Gesellschaft für Informatik (GI)

publishes this series in order to make available to a broad public recent findings in informatics (i.e. computer science and information systems), to document conferences that are organized in cooperation with GI and to publish the annual GI Award dissertation.

Broken down into the fields of
• Seminar
• Proceedings
• Dissertations
• Thematics
current topics are dealt with from the fields of research and development, teaching and further training in theory and practice. The Editorial Committee uses an intensive review process in order to ensure the high level of the contributions.

The volumes are published in German or English.

Information: http://www.gi-ev.de/service/publikationen/lni/

ISSN 1617-5468
ISBN 978-3-88579-234-5

IMF2008 is organized by the working group SIDAR (Security – Intrusion Detection and Response) and took place on September 23 - 25, 2008 in Mannheim. This volume contains ten accepted papers and two papers of invited speeches.To assure scientific quality, the selection was based on a strict and anonymous review process.The papers cover the subjects incident management, incident detection, , IT forensics, and IT forensics education.

Oliver Göbel, Sandra Frings, Detlef Günther, Jens Nedon, Dirk Schadt (Eds.)

140

GI-Edition

Lecture Notes in Informatics

Oliver Göbel, Sandra Frings, Detlef Günther, Jens Nedon, Dirk Schadt (Eds.)

IMF 2008 IT Incident Management & IT Forensics

Conference Proceedings
September, 23 – 25, 2008, Mannheim

Proceedings

Oliver Göbel, Sandra Frings,
Detlef Günther, Jens Nedon, Dirk Schadt (Eds.)

IMF 2008

IT Incident Management & IT Forensics

**Proceedings of the 4th International Conference on
IT Incident Management & IT Forensics
September, 23 – 25 , 2008
Mannheim, Germany**

Gesellschaft für Informatik e.V. (GI)

Lecture Notes in Informatics (LNI) - Proceedings
Series of the Gesellschaft für Informatik (GI)

Volume P-140

ISBN 978-3-88579-234-5
ISSN 1617-5468

Volume Editors
Oliver Göbel
Stabsstelle DV-Sicherheit der Universität Stuttgart (RUS-CERT)
Breitscheidstraße 2, 70197 Stuttgart, Germany
Email: goebel@cert.uni-stuttgart.de
Dirk Schadt
Spot Consulting
Sandra Frings
Fraunhofer Institut für Arbeitswirtschaft und Organisation IAO
Detlef Günther
Volkswagen AG, ISSO
Jens Nedon
ConSecur GmbH

printed by Köllen Druck+Verlag GmbH, Bonn

Preface

Information technology has become crucial to almost every part of society. IT infrastructures are critical to the world-wide economy, the financial sector, the health sector, the government's administration, the military, and the educational sector. Due to its importance the disruption or loss of IT capabilities results in a massive reduction of operability.

Hence, IT security is continously gaining importance and has become technically essential to IT infrastructures.

Although security usually gets integrated into the design process of IT systems nowadays, the process of maintaining security in IT infrastructure operation still lacks the appropriate attendance in most cases.

Especially the capability to manage and respond to IT security incidents and their forensic analysis is established in the rarest cases. The quickly rising number of security incidents worldwide makes the implementation of incident management capabilities, targeting the mitigation of immediate consequences to the own infrastructure, essential.

Also, the need of subsequent forensic analysis of selected cases to gather evidence on the incident's details and work up the information for law suits or to avert unwarranted liability claims of aggrieved third parties is constantly growing.

In order to advance the fields of IT-Incident Management and IT-Forensics the special interest group *Security - Intrusion Detection and Response* (SIDAR) of the German Informatics Society (GI) organises the annual *International Conference on IT-Incident Management and IT-Forensics* (IMF), bringing together experts from throughout the world, to discuss state of the art in these areas. IMF promotes collaboration and exchange of ideas between industry, academia, law-enforcement and other government bodies.

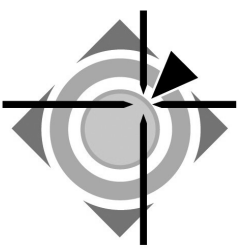
IMF 2008 is supported with keynotes by the president of the German Federal Office for IT Security (BSI), the U.S.Army Research Laboratory and the German Federal Office of Criminal Investigation (BKA),.

The organizing committee would like to thank all persons who helped in realizing the conference, especially the authors whose papers and presentations make the essence of the conference, the members of the program committee who reviewed and evaluated the papers submitted and whose professional competence ensures the scientific quality of the program, as well as the sponsors who supported the conference.

Oliver Göbel, Sandra Frings, Detlef Günther, Jens Nedon and Dirk Schadt

Program Committee

Susan Brenner, University of Dayton, USA
Klaus Brunnstein, University of Hamburg, Germany
Jack Cole, U.S. Army Research Laboratory, USA
Andrew Cormack, JANET, UK
Ralf Dörrie, Germany
Sandra Frings, Fraunhofer IAO, Germany
Oliver Göbel, RUS-CERT, Universität Stuttgart, Germany
Detlef Günther, ISSO, Volkswagen AG, Germany
Vijay K. Gurbani, Bell Laboratories, Alcatel-Lucent, USA
Bernhard Hämmerli, ACRIS GmbH, Switzerland
Alexander Herrigel, Secude Int. AG, Switzerland
Klaus-Peter Kossakowski, DFN-CERT, Germany
Thorsten Lieb, Avocado Rechtsanwälte Frankfurt, Germany
Jim Lyle, NIST, USA
Robert A. Martin, MITRE Corporation, USA
Ralf Moll, LKA Baden-Württemberg, Germany
Jens Nedon, Consecur GmbH, Germany
Henning Pagnia, Berufsakademie Mannheim, Germany
Hartmut Pohl, FH Bonn-Rhein-Sieg, Germany
Jason Rafail, CERT/CC, USA
Dirk Schadt, Spot Consulting, Germany
Mark Schiller, Stratton Security Ltd, UK
Marco Thorbrügge, ENISA, EU
Steven Wood, Alste.Technologies GmbH, Germany
Stephen Wolthusen, Royal Holloway, University of London, UK,
Gjovik University College, Norway



Organization

Organizing Committee:	Sandra Frings, Fraunhofer IAO, Germany Oliver Goebel, RUS-CERT, Universität Stuttgart, Germany Detlef Guenther, ISSO, Volkswagen AG, Germany Jens Nedon, ConSecur GmbH, Germany Dirk Schadt, Spot Consulting, Germany
General Chair:	Dirk Schadt, Spot Consulting, Germany, dirk.schadt@spot.net
Program Chair:	Oliver Göbel, RUS-CERT, Universität Stuttgart, Germany, goebel@cert.uni-stuttgart.de
Sponsor Chair:	Dirk Schadt, Spot Consulting, Germany, dirk.schadt@spot.net
Local Organization:	H.J. Müller, Berufsakademie Mannheim, mueller@ba-mannheim.de

In Co-Opereation with:

Institute of Electrical and Electronics Engineers, Inc.
http://www.ieee.org/

IEEE Computer Society
http://www.computer.org/

Stabsstelle DV-Sicherheit der Universität Stuttgart (RUS-CERT)
http://cert.uni-stuttgart.de/

Fraunhofer Institut für Arbeitwirtschaft und Organisation (IAO)
http://www.iao.fraunhofer.de/

ConSecur GmbH – security & consulting
http://www.consecur.de/

Spot Consulting
http://www.spot.net/

Hosted by:

Berufsakademie Mannheim
http://www.ba-mannheim.de/

Supported by:

European Network and Information Security Agency
http://enisa.europa.eu/

International Information Systems Security Certification Consortium, Inc.
http://www.isc2.org/

Sponsored by:

Alste Technologies GmbH
http://www.alste-tech.com/

Keynotes/Invited Speakers

Udo Helmbrecht

Präsident des Bundesamtes für Sicherheit in der Informationstechnik (BSI)

New Challenges for IT-Security Research in ICT

Jack Cole

U.S. Army Research Laboratory

Incident Management and Forensics Put in Context

Fred-Mario Silberbach

Bundeskriminalamt (BKA)

Investigations and Prosecution in Cases of Computer Crime – Overview on the National and International Situation

Felix Lindner

Recurity Labs GmbH

Network Infrastructure Forensics

Table of Contents

Keynote

New Challenges for IT-Security Research in ICT

Dr. Udo Helmbrecht

Präsident des Bundesamtes für Sicherheit in der Informationstechnik (BSI)

Godesberger Allee 185-189
53175 Bonn

bsi@bsi.bund.de

Threats in information and communication technology (ICT) have substantially increased over the last years. The dependence on modern communication technology in professional and private environments results in an increase of IT risks. Trojan horses and DoS attacks performed with bot-nets are the most challenging attacks we face today. But we are also confronted with new ways to attack: data encrypted today with current state of the art technology and stored for long periods may be attacked using quantum computer technology then possibly operational. There may be new mathematical possibilities to crack classic cryptography. New technological possibilities to eavesdrop using stray radiation may come up. Observing the overall development reveals that the quality of attacks increases continously. Over time technological possibilities of single individuals will vastly exceed the ones of government agencies today. Thus we need cutting edge ICT-security research to be prepared for upcoming threats. The talk provides an overview on current technological trends and proposes the creation of a new culture of cooperation of the security and academic community.

Invited Talk

Incident Management & Forensics Put in Context

Jack Cole

U.S. Army Research Laboratory

AMSRD-ARL-CISD, APG, MD 21005-5067
cole@arl.army.mil

Incident Management and IT Forensics (IM&F) is a member of a family of activities that interdependently support and affect each other in what superficially appears to be a linear relationship. Immediate family members include intrusion detection, information security, information assurance, and decision support. In this interdependent context members suffer and benefit as well from various external or global factors. Each member is affected by perceptions of worth, other aspects of the social and cognitive domains, and by inherent limitations of both IT and of humans.

The worth of successful operations, transparent to others because of their success, is often falsely perceived at best as regrettable overhead, at worst as unrelated to higher level enterprise goals, and therefore funding support is suboptimal.

Identifying network attacks and determining their extent and causes relies on human judgment in examination of multiple views of seemingly disparate events and facts, developing belief that these things together constitute an attack, and weaving these bits of information into a visible tapestry of the attack.

While fairly successful, such approaches are often developed experientially, sometimes lack the rigor of formal analysis and concepts, and may be improved by application of technologies now used in seemingly unrelated areas. Such is the promise of "network science", a relatively new field, to bring renewed interest in rigor and formality and to revive consideration of the social and cognitive domains in developing information networking and security and other areas.

Among the most significant limitations of IT is simply the fact that most data, perhaps 80%, is undeliverable in any meaningful time frame, and that the cognitive process begins only after delivery!

Human limitations are of course many, but the capability of humans to read text, hear speech, and accept information visually is at present several orders of magnitude less in terms of input rates than the flow rate of multiple terabytes of real-time network data to be analyzed, even discounting stored information. With poor funding, hiring thousands of analysts is out of the question, but even robotic analysts can not measure up. Understanding the social and cognitive aspects, human limitations, and intrinsic IT limitations that affect IM&F and its siblings can lead to better tools and strategies.

Awareness of this context and application of some ideas from network science may permit IM&F to meet growing demands for its services.

Investigations and Prosecution in cases of Computer Crime – Overview of the National and International situation

KOR Fred-Mario Silberbach

Bundeskriminalamt (BKA)
SO43 Analysis, Investigations, High-Tech-/
Computer Crime, Random Internet Searches
65173 Wiesbaden, Germany
fredmario.silberbach@bka.bund.de
so43@bka.bund.de

1. Crime Statistics and Developments / Trends

Regarding the Crime Statistics (Polizeiliche Kriminal-Statistik - PKS)[1], which is published annualy by the Federal Criminal Police Office (Bundeskriminalamt – BKA), could lead to the following conclusion: The real situation of Computer Crime is not very clear.

The definition for Cybercrime, which is written down in the Cybercrime Convention of the European Council, still has not been ratified by Germany. So there is still no legal definition for computer related crime in Germany.

For a better understanding about the statistics concerning computer crime it is necessary to point out that the national crime statistics of the BKA differentiates between

- Computer Crime in a so-called common sense, that contains all crimes where the Internet was used as an instrument for committing the crime and
- Computer Crime in a so-called closer sense, that contains offences like Computer Fraud, fraud by accessing communication services, forgery of evidentially data, data espionage, alteration or sabotage of data and computer sabotage.

In 2007, in Germany 179.026 crimes were reported to the BKA where the Internet was used – in a common sense - as an instrument of crime. This means an increase of 8 % compared to 2006 (165.720). 73% of all reported crimes were fraud offences and 40% of the total number of these offences (71.876) concerned fraud by obtaining goods.

The number of computer crimes in a closer sense has risen from 29.155 in 2006 up to 34.200 crimes in 2007, which is an increase of about 17,3%[2]. A further rise in the next years must be expected.

Phishing is not a crime which is officially listed in the German Penal Code. Phishing describes a modus operandi which consists regularly of the listed crimes data espionage, data alteration, computer fraud, obtaining goods by fraud and money laundering. Therefore, the numbers of Phishing cases are also not listed in the above mentioned National Crime Statistic.

[1] Bundeskriminalamt, 2007, Polizeiliche Kriminalstatistik, p. 236, Wiesbaden/Germany
[2] see above

To get an impact about these numbers it is necessary to evaluate the criminal cases which are reported by the Criminal Investigations Departments on an additional Case Reporting System.

Concerning Phishing in connection with online banking, in 2006 3.500 cases were reported to the BKA with an average loss between 2000 and 3000 € (total loss between 7,000.000 and 10,500.000 €). In 2007, already 4.200 cases were reported with an average loss of 4.500 €[3] which means a total loss of 19,300.000 €. But not every case has been reported to the BKA and a considerable higher number of cases and a higher loss must be estimated - only in 2007 and only in Germany.

At the end of 2007 and the beginning of 2008 in Germany also the last of the big German bank corporations implemented the iTan (individual Transaction-Number), an updated and saver mechanism for online banking instead of the PIN/TAN-method. The positive result: In the first seven months of 2008 the number of Phishing cases decreased to 1.100 cases which means a reduction of about 54% [4]. But now the BKA notices a higher intensity of attacks against banks which are using the iTAN-method. Actually there exist at least three Trojan-Families which attack the in Germany commonly used iTAN-method. One of these families focuses only the German Market and attacks exclusively German banks. And mostly, this modern malware is not detected by signature-based antivirus-software.

In the last years, the offenders could catch credit or debit card data, ebay or banking account information because the computer users – the prospectively victims - clicked on links in received Emails and entered their login data at faked websites. But the situation has changed. Phishing via Spoof- and Fakesites has nearly vanished in Germany and throughout large parts of Europe. Malware is hitting the market massively. 99% of the Phishing cases reported to the BKA now is being done by malware. Partly, this malware still is downloaded to a computer because the user opened an attachment. But the development shows an increasing rate of drive-by-infections, that means, when users follow a link to an infected homepage or when users only visit infected homepages from innocent companies, restaurants and internet-services. This modern malware is evolving quickly and aiming for multi-usage business models. It is self-upgrading, built-on-demand, it does contain not only Keylogging- and "Man-In-The-Middle"-functionalities but also Proxy-, Remote- and Spam-capabilities.

In an investigation of the BKA, in September 2007 a group of 10 offenders was arrested, which had caused a loss with trojan-based Phishing of nearly 700.000 € in a period of 18 months.

But Phishing is not limited to banks anymore. Malware, infected computers and networks can be used in more than one way to generate money. Phishing now is about losing the digital identity in its entirety. This contains e.g. the access to Email-Boxes, to ones company network and/or distributed resources, to online merchandising services (e.g. ebay, amazon) and to social networking portals (e.g. stayfriends.de, xing.com).

[3] Bundeskriminalamt, 2008, Kriminalpolizeilicher Meldedienst – KPMD, Wiesbaden
[4] see above

Stealing data of credit cards becomes more and more popular. In 2007, the online-gaming portal „Steam“ was successfully compromised and millions of valid sets of credit card-data had gone lost. Rumours are talking of more than 10.000.000 sets of credit card data. Only six hours after this security breach, more than one million of those credit card data-sets were offered in the so called Underground Economy by specialised resellers. In another case of 2007, a company lost 45.700.000 datasets of valid credit and debit cards.
These just mentioned cases are only two examples among many others. As direct consequences, many banks and/or online merchants worldwide are being hit by fraudulent transactions pointing back to those incidents.

Stolen data are sold and bought in the just mentioned Underground Economy. For example, datasets of German credit cards with a limit of 5000 € are sold for 2€ each, US-cards are sold for only 1 € - and an additional discount for buying great numbers of datasets is also possible.
The Underground Economy is furthermore a global market where services around digital identities are traded. And the amount of stolen data is searched and filtered by special software (“Harvester”) to generate single information that can be used by special interested offenders to commit other crimes. Today, it is absolutely no problem to find, to hire or to buy components or services like malware, financial agents, spam sending services, anonymized or encrypted communication channels, false identities, credit card data, login / access data for all sorts of digital identities, anonymized internet based payment-systems, which also can be used for money-laundering, and exchange offices for digital currencies.
Knowledge and resources are traded and paid with digital currencies like E-Gold and Web-Money.

In spite of all the above mentioned data, information and examples, a considerable dark field must be assumed. Regarding present hints, a successful invasion of a computer very often is not noticed by the user or owner. And especially in those many cases of noticed attempted invasions or completed invasions but without financial loss, the user do not complain the attack to the police. Last but not least, attacked companies regularly do not report attempted attacks or successful invasion to the police. The reasons might be the expected and feared loss of reputation, the confidence among their private and commercial customers and also questions of liability.

2. General Conditions for investigations and prosecution of Computer Crime in Germany

The “typical user”
Due to statistics of the Federal Statistical Office, in 2007 round about 28,300.000 Computers (PC, Notebooks, PDA´s) were used in private households[5]. A lot of these computers are bought by “normal users” in department stores, in internet-stores or internet-auctions and also in discounter stores. In most cases these computers are equipped with a timely limited security software or only with test- or beta-software-versions. In many of our known cases the victims were using the computer for surfing in the internet, for doing online-banking, for chatting, for changing files and for all the other interesting things – but without any or a sufficient protection of their computer. Very often there is still no or not sufficient knowledge about the risks of using the internet and the possibilities to protect the computer effectively. And also users with an increasing or outstanding knowledge

[5] Statistisches Bundesamt – Destatis, 2008, Wiesbaden

about risks and measures become more and more victims because the offenders develop new and very professional malware and techniques to steal data and information.

Darkfield Economy

A base for developing strategies and measures to fight against crime are the statistics about reported crimes. Of course, sometimes the validity of those information is limited and it has to be added by estimates or results of research to get further information about the dark field and to get an impact about the real situation of a phenomenon.
But concerning the situation of computer crime in the German economy it must be noticed: it is very dark! The BKA just has little information about quantity and the quality of computer crimes that are committed against economic companies in Germany and also about the caused loss. This dark field „Economy" makes it nearly impossible to estimate the real situation of computer crime, to develop strategies and enforce measures against those crimes and also to advise the politics. It is obvious for everybody: to reach a target or to fight against an enemy it is necessary to see the target or to know the enemy.

Legal possibilities and limits

Other important aspects are the legal possibilities and limits for investigations in cases of computer crime.

In March 2008 the German Constitutional Court has confirmed in a preliminary decision the general obligation for the suppliers of telecommunication services to preserve traffic data so that the police or other authorities can demand and collect those data in cases of serious crime. But from the point of view of the High Tech Crime Unit of the BKA it is necessary, that the Constitutional Court confirms the legitimacy of the complete regulations of the specific rule.
For starting investigations and to solve cases of computer crime the existence of traffic data is essential. Without any traffic data, in many cases it is useless to start the investigation and the file has to be closed directly after a case has become known to the police. When e.g. hackers attack the server of a company to steal sensible information, the IP-address at the moment of the attack regularly is the only starting point for further investigations. In these many cases of Phishing, the IP-addresses of the offenders – except the bank account data of the financial agents in Germany - are the essential information just to start and to move one step closer to the offender.

A long period of time, the internet service providers were not liable but also not yet authorized to preserve traffic data.
Due to Directive 2006/24/EC of the European Parliament and of the Council on the "Retention of data generated or processed in connection with the provision of publicly available electronic communication services" (March 2006), the supplier of telecommunication services in Germany have to preserve traffic data for a period of six months. Several times not only the BKA has demanded this result of an European agreement which – by the way – in Germany stays 18 months under the period of maximum 24 months which is allowed by the above mentioned European directive.

In this context it is very important to point out, that the preservation of traffic data is not only necessary to clear up crimes – it is also necessary to avoid crimes or to prevent danger in general.

At the end of 2007 there was a case, while in a running chatroom session in Germany a person had announced a run of amuck. The lack of stored IP-address-data here but the knowledge about an Email-account hosted in a foreign country was the only way to get more information in this situation. Due to the lack of further information about the owner of the email account, the foreign National Police Office had to seize the content of this account. They mailed it to Germany and by evaluating the content it was possible to identify the responsible person. The result of many hours of investigating, phoning and mailing during the night and a part of the week-end: The person could be found – but only luckily. In this case, the person did not intend to run amuck in reality. But other announcements – also from people who want to commit suicide - might have a real background. Should it depend only on a lucky chance to clear those situations?

Regarding the task of the police, to fight against crime and to find evidence with different criminalistic methods, the present problems and challenges, based on experiences on different cases investigated by the police, can be described as followed:

Technical development

Doing searches today, the police finds and seizes more and more technical equipment like personal computers, notebooks, PDA´s, mobile phones, digital cameras, mp3-players etc. Computers with a capacity of one terabyte and more are not rarities any more. Only in one case, a seized computer with a hard disk containing 2,5 terabyte of data in 300,000 files had to be evaluated.

Today, offenders communicate regularly with mobile phones, by Voice over IP (VoIP) or via the Internet, very often also using Internet-Cafes, Call-Shops or not-protected WLANs of other people. It must be noticed that offenders save their information, which are evident for the police, increasingly not only on their computer at home but also in the webspace. And it is also not a secret that anonymity and encryption of communication and at the storage of data takes place more and more. The decryption of seized IT-evidence and communication data demands a high personal and technical effort or it is increasingly impossible.

There is an increasing use of broadband internet access with a data transfer rate nearly 2000 times faster than an (old) analogue line. The consequences: Only one interception can block a major part of the interception and data storage capacities of a big Criminal Investigation Department. And these amounts of data also must be verified and evaluated, in general by police officers. But the number of police officers does not increase as much as the amount of data.

Training and Equipment

For doing investigations effectively and with the correct criminalistic and forensic methods, in general well qualified personal and also modern hard- and software is needed.
But do we already have such experienced police officers in every police station or Criminal Investigation Department in the country? Officers, who exactly know what to do when an victim, e.g. in a Phishing case, comes to them to file a complaint? Or in cases, when network administrators, that have noticed unauthorized data traffic in an obviously infected and captured server, want to give the police a little chance to intercept this data traffic in order to find the offenders instead of just stopping any network activity? No, but the police is working on it, although the police has many other duties in order to protect the citizens and to fight against other forms of crime. The police in Germany has recognized this problem and meanwhile founded special High Tech Crime Units at different police

offices – yet mostly at bigger Police or Criminal Investigation Departments. But in the federal structure of Germany the situation between the sixteen National States still is different.

These problems - or challenges - in general also concern the judicial bodies. Single statements of prosecutors and judges and also experiences of investigations showed partly a lack of knowledge about new technologies, a lack of acceptance for new challenges and also a lack of willingness to go new ways for the handling of such cases at court.

3. The International Situation

The just mentioned examples to show the general conditions for investigations and prosecutions in Germany and the hereby following problems for the fight against computer crime become another dimension when the cases become international. And it is obvious, that there are only a few cases or nearly no cases where the perpetration occurs only within the national borders.

In an essential part of the cases of computer crime it is normal to find

- E-Mail-Accounts of offenders at foreign Internet Providers or Mail Services or in Germany offered internet services but where the content of mails is stored on foreign servers
- Malware stored on foreign servers to get downloaded in order to infect or capture computers or networks in Germany
- IP-Adresses from foreign servers when attacks were enforced and noticed
- Foreign servers where stolen data or tools to commit further crimes (e.g. like Rock-Phish) are stored, changed and dealt with.

These are just a few examples to show the international character of Computer Crime. But while enforcing an investigation for the police or the prosecutor it becomes interesting now.

To improve the change of information about E-Mail-Accounts, IP-addresses and other stored data, in 1999 the G8-countries published "Principles On Transborder Access To Stored Computer Data"[6]. Among other things, these principles declare for example, that the different states

- shall ensure their ability to secure rapid preservation of data stored in computer-systems
- may request each other to secure rapid preservation of data stored in such systems
- shall take all appropriate means, in accordance to its national law, to preserve data as expeditiously as possible.
- shall execute requests for expedited legal mutual assistance in accordance to its national law as expeditiously as possible.

To ensure the implementation of those principles, the G8-countries first installed a network consisting of single point of contacts in each country with a 24 hour 7 days a week presence. In Germany, this single point of contact is within the High Tech Crime Unit of

[6] G8 - Ministerial Conference on Combating Transnational Oranized Crime, October 1999, Moscow

the BKA. These G8 24/7 points of contact are provided for investigations involving electronic evidence that require urgent assistance from foreign law enforcement. The introduction of the so-called "preservation order" provides the members of the network with an instrument that helps to freeze volatile data in a target country before an official letter rogatory for data preservations will be sent through the appropriate channels – a process that is time consuming and which normally takes longer than the usual storage of data.

This network has grown during the last years and actually there are similar contact points in 50 countries worldwide. But this shows also, that many countries still don't have installed such units. To communicate with those states in cases of computer crime, it is necessary to use the normal Interpol channels. In general, the communication in such cases needs more time and sometimes it is not clear, if, when or from which foreign unit exactly you will get an answer for your request or not and whether the results justified the efforts before.

To ensure that the information which has been exchanged through appropriate Interpol channels reaches the specialized police units with the least possible delay, Interpol has compiled a list of National Central Reference Points (NCRPs) for computer related crime. To date, nearly 120 National Central Bureaus of Interpol have designated such NCRPs. These NCRPs are also an essential prerequisite for the establishment of an early warning system.

The introduction of the above mentioned measures were important first steps to improve the fight against Computer Crime. But a continued international harmonization of the legal base in the different countries is furthermore necessary.

To receive information from other countries, for example information about the subscriber of IP-addresses in an actual Phishing-case, first of all it is necessary that the crime, that happened in one country and which was the starting point for a request to a foreign country, also is a punishable act with regard to the foreign penal code. Further it is necessary, that the domestic law in the countries regulates e.g. the preservation of traffic data and the obligation for Internet Service Providers to hand out requested information immediately to the police.

As far as these requirements are not met, in many cases of computer-related crime it is even not necessary to send requests to different countries. And also in Germany it is still necessary to answer many countries, that they will not receive a suitable answer to their request concerning traffic data because of the present legal situation at the preservation of traffic data.

4. Needs for Action

To improve the fight against computer crime there are various measures which should be enforced.

It is necessary to establish specialized units at the police offices and to implement specific training programmes. In Germany, the BKA - in cooperation with the State Police Training Institutes – is currently setting up a national training programme for first responders, case officers and IT forensic experts. It is also necessary to think about setting up public

prosecutors offices specialized in IT-crime and to establish specific training not only for law enforcement offices but also for prosecutors and judges.

Concerning the legal bases, a wide spread ratification of the United Nations Cyber Crime Convention is necessary to improve the international cooperation by establishing unique standards in all the member states in the world.

The need and the importance of the preservation of traffic data for the investigation in cases of computer crime already has been described. It now depends on the final decision of the German Constitutional Court to give the police this necessary instrument.

A better and closer cooperation is needed, on an international and a national level, and also between law enforcement offices, IT-based offices, the Internet Economy, banks and other private organizations and institutions.

On the international level there is now a working party of the Council of Europe "Cooperation between Service Providers and Law Enforcement against Cybercrime". It consists of participants from law enforcement offices and the industry and develops common guidelines for the necessary cooperation in the future.

The biennially Interpol conference on Cybercrime is a useful platform to meet the counterparts of the BKA from many other countries, to exchange information about the phenomenon and to develop strategies and measures against this sort of crime.

In Germany a closer cooperation between the BKA and the Federal Office for Information Security[7] on technical issues has been implemented.

It is also necessary to raise the awareness among the public and also among parts of the economy concerning the threat of computer related crimes and the possibilities to react on it.

The "Programm Polizeiliche Kriminalprävention der Länder und des Bundes" (ProPK) is a cooperation between the Federal Gouvernment and the National States in Germany to prevent of crime. Together with the ProPK, the BKA develops and publishes the so called "IT-Newsletter" with information for the people in Germany about new trends and how to minimize their risk of becoming victims of crime.

Very important is a better information exchange – especially with the economy. The police is not the enemy of the economy. But a better information exchange with respect to the respectively needs, possibilities and limits can help the police and the economy to provide a better and safer environment for "their customers". The implementation of a frequently information exchange with representatives from the BKA and several german banks is only one opportunity to go into the right direction.
The participation of different representatives of the economy in a current project group of the german police to develop a common strategy against computer-related crime is another example to improve the national cooperation.

[7] Bundesamt für Sicherheit in der Informationstechnik – BSI

5. Outlook

In the time of a fast developing technology and also of fast developing methods for committing crimes transborder, worldwide and within milliseconds, the society must give the state and his police the necessary instruments to fight against this crime.

Due to the constitution, the state is responsible for the safety of its citizens. The German Constitutional Court (Bundesverfassungsgericht – BVerfG) has emphasized several times the peremptory need for an effective criminal prosecution and for a fight against crime. There is a public interest in a complete - as possible - establishment of the truth. Of course – not at any price. The State has to respect the constitutional rights of its citizens.

There are critics and statements, that the State only wants to collect more data and information to improve the possibilities of observing and intercepting its citizens or to control the whole society. It is also asserted, the State would hurt the constitutional rights of its citizens.
But these critics should compare the fast speed of the technical development and the up to date grown possibilities of fast and worldwide communication in the area of Computer Crime on the one side with the development of the legal possibilities of the State on the other side. And the critics should also take into consideration, that the demand for respecting the constitutional rights of the citizens means also, that the State has to care for those citizens, which have a legitimated claim to get protected by the State.

The president of the BKA, Mr. Ziercke, expressed the willingness of the BKA and the Police of the German National States to carry on an open and fair dialogue based on the intelligence about the situation of crime[8].
With relation to questions directly concerning the phenomenon computer crime, also the High Tech Crime Unit of the BKA is open for dialogues and discussions.

[8] Jörg Ziercke, 2007, „Polizei in der digitalen Welt“, Autumn Conference Bundeskrimimalamt, Wiesbaden

Network Infrastructure Forensics

Felix Lindner

Recurity Labs GmbH
Wrangelstr. 4
10997 Berlin, Germany
fx@recurity-labs.com

Abstract: Incident identification, response and forensic analysis depend on the ability to extract meaningful evidence from the suspected system. Such tools do not exist for network infrastructure equipment. The significantly increased attack resilience of common general purpose operating systems poses a surprising new challenge to forensics, as attackers will likely shift their attention back towards network infrastructure control. The paper discusses the importance of network equipment forensics, the anatomy of devices and the attack types encountered. Finally a method for performing forensics on a widely used type of network equipment is presented.

1 Introduction

According to the definition, the goal of computer forensics is to explain the current state of a digital artifact [Wi08]. Computer forensics is normally performed on common desktop and server computer systems, digital media and data items. In the past, very little attention was paid to computer forensics on network infrastructure equipment.

This paper first discusses the need for forensic analysis methods and supporting technologies for network equipment and will show why the previously neglected capabilities need to get build up quickly. Following, the paper classifies the types of network equipment commonly encountered and the types of attacks that need to be detectable. The paper concludes with a comparison of the currently used evidence gathering mechanisms to a method developed by the author.

2 Network Equipment Forensics

Forensic investigators should be able to extract evidence from any sufficiently complex electronic device in a regular computer network. As new device classes become common on a regular basis, forensic investigators have to identify the means of performing evidence gathering, examination and analysis on the new device classes. Recent challenges for investigators include various mobile phone platforms, MP3 players and portable storage devices. New capabilities have to be developed constantly as users store relevant data in more diverse places, distributing information important for investigations.

Digital forensic procedures have so far largely neglected the device class of network equipment. For the purpose of this paper, we define network equipment as:

> A physically enclosed component of a communications network that is built for the purpose of performing programmed transport actions on data, based on information within the transported data.

The definition encompasses all so-called "active network gear" with the exception of simple repeater stations or otherwise "non intelligent" equipment, but excludes multi-purpose operating systems that could perform the same task.

We will now review the requirement to have analysis methods and supporting technologies for network equipment.

2.1 Attacker Focus Shift

The need to perform digital forensic investigations on a device class is usually created by users storing personal information on devices of this class. Within this category also falls the case of the device class becoming a target of malicious activity by itself, as attackers must inject data into the device to perform directed attacks. Alternatively, the need for digital forensics can also arise when the device class exhibits detectable traces of user data or communication. The later argument holds true for network equipment by its very nature, the former requires inspecting network equipment as a target more closely.

Directed attacks on network equipment were the norm in the early days of communication networks, when sender or sink devices on the network were either dumb or closely monitored equipment in the hands of experts. Directed attacks on network equipment were declining in the more recent past, as the connected computer systems with common operating systems became increasingly powerful, interconnected and commonplace. The effort required to successfully attack a network device was now comparatively large in contrast to the development of an attack against a common operating system platform or widely used software. Additionally, successful intrusions into network equipment are not as easily detectible as intrusions on multi-purpose computer systems are. Accordingly, digital forensics has had little demand to perform investigations on network equipment and the numbers on un-detected attacks are probably much higher.

In the recent years, fueled by public demand to stop large scale intrusions and worm outbreaks, the vendors and publishers of common operating systems have significantly improved the protections shipped with their products. Security features, mitigation technologies and better control over privileges have become an important aspect of any operating system deployment decision made, from professional applications down to home users. As security becomes a market factor, competition drives significant improvements on all major operating system platforms, compilers and libraries. Additionally, a plethora of third party security products is available and marketed to close the remaining gaps left open by default installations and hard to secure application software.

The success of this market-wide shift towards more secure software and installations can be easily verified by the drastically lower numbers of newly detected vulnerabilities in operating systems. Another indicator is the apparent end of the era of worms that rely on software vulnerability exploitation for propagation. Not a single large outbreak has been observed in the last years on platforms that used to be notoriously infected just a few years ago.

It can, however, not be shown or even expected that the total number of attackers or their skill level decreased over the same period of time. To the contrary, cyberspace warfare has become an integral part of many first world nation states' military operations. Offensive cyber-warfare groups have now to prove their usefulness. Privately operated groups, mainly illegal, have also banked on business models involving intrusions into computer systems.

This produces a gap between the increase in efforts required for a successful intrusion and the goals of such groups, may they be government sponsored or privately run operations. Since network equipment provides significantly less resilience against attacks than modern operating systems on desktops and servers, it is the opinion of the author that the focus of professional attack groups will shift towards network equipment again.

2.2 Network Equipment Attacks in the Wild

Very little is publicly known about successful attacks on network equipment. While a number of articles have been published in so-called underground magazines [Ph00] on different procedures to perform attacks and establish control of network equipment, intrusions remain mostly undetected and hence undocumented.

Reports about the abuse of illegal access to network equipment are only recently disclosed publicly. [Di08] discusses a case of blackmail in which the attacker(s) obtained control over the router network and blackmailed the legitimate owner of the network for restoration of their access without network downtime. The alternative for the legitimate owners would have been an entire network shutdown and incremental restoration and restart procedure, potentially requiring several days of downtime.

When inspecting publicly accessible information on attacks in general, it can be observed that the real number of attacks on network equipment is not to be neglected. In [BC04], an open HTTP proxy server is inspected for malicious activity performed through its service. 0.94% of all activity targets Cisco routers. When searching for information on a single Cisco IOS vulnerability [CVE01], sources such as [Te06] show that it is actively scanned for and exploited in the wild five years after the vulnerability has been reported and fixes are available. Since attackers rarely scan networks for vulnerabilities that they consider extinct, reports like [Te06] indicate clearly that the attackers consider it likely to still find vulnerable equipment.

2.3 Underlying Threats

To understand the underlying threat of compromised network equipment, it is necessary to review the basic functionality of today's communication networks. Almost any network today uses the TCP/IP protocol suite [Po81], which by design relies on intelligent network nodes taking decisions on the transport of user data from one point in the network to another.

The communication endpoints in IP have very limited abilities to influence the path a segment of the data (packet) takes from the sender to the sink. With the rapid expansion of the Internet, all remaining functionality of sender-decided paths was blocked in the network to prevent endpoints from influencing the availability of specific network paths.

The intelligent network nodes required by IP are network equipment. They rely on so-called routing protocols to communicate network paths to each other. Without this communication, the nodes cannot fulfill heir role. All routing protocols support their implementation without authentication between the peers taking place, only some of the more widely used protocols even support authentication. The inherent problem, even with authentication deployed, is the full trust relation of nodes within the communication network. Compromising a single node can in some cases already yield full control over the communication path decisions in the entire network.

In reaction to the inherent insecurity of today's computer networks, a variety of security protocols on top of the existing infrastructure has been proposed and deployed (e.g. [DR06], [DD03]). Common between all the security protocols is the protection of the communication contents (integrity) and un-tempered sender identification. Additionally, most of them offer confidentiality through encryption.

The security protocols protect their payload, but cannot influence the communication path taken by the individual packets, as the communication infrastructure is transparent to the security protocol. Accordingly, they can only offer to discard the communication data items (packets) when they were modified by an unauthorized party during transport. None of the communication protocol suites in wide use today offers the ability to choose a different communication path or channel when the current channel is misbehaving maliciously, and none of them could, as it would open the possibility to attackers for direct misuse of this very functionality.

2.4 The Goal of Network Forensics

Based on the observations above, the goal of network equipment forensics must be the ability to extract meaningful and complete evidence from network equipment. Analysis should enable the legitimate operator of the network equipment to identify a successful intrusion and show beyond reasonable doubt that an intrusion occurred. Additionally, it should be possible to identify the sender of data that can be shown to have a relation to the intrusion.

3 Network Equipment Anatomy

Network equipment, as any other computer system, is built by the basic blocks of at least one processor (CPU), several types of volatile random access memory (RAM), permanent (non-volatile) storage and interfaces to other systems. In contrast to multi-purpose computer systems, network equipment makes use of the permanent storage almost exclusively at boot time, as the storage provides the system software and initial configuration. Once started completely, all information and state is kept in RAM, with the only exception being configuration changes that are written back into permanent storage and minor data items in small non-volatile memories. Additionally, network equipment often has separate memory banks for special purposes, such as local module memory or network interface memory, all of which is completely volatile.

As outlined, almost all evidence is volatile data, which is a significant difference to general-purpose computer systems, where volatile data is only a small part of the evidence collected. The forensic investigator needs a way to preserve a snapshot of the volatile data from the device, as it contains almost all the evidence there is. The potential to collect the evidence depends on the device class of the network equipment examined. Two general device classes can be identified:

3.1 The Monolithic Device Class

Older network equipment is generally built as a combination of custom hardware designs and software specifically developed for the hardware platform. Since the network equipment market has significantly longer product cycles as other market segments in information technology, the majority of deployed network equipment still follows the monolithic design approach. Prominent examples of monolithic system designs are routers and switches from vendors like Cisco Systems (IOS based) and HP.

The majority of monolithic devices run software that is termed “firmware”. The term is used to indicate that the software is not a modular operating system, but rather a single binary program compiled into self-contained executable code, with the goal of minimizing the required amount of resources. Along with this design decision, vendors usually abandon concepts such as kernel and process separation, virtual memory protections and concurrent scheduling.

It should be noted that monolithic devices have no ability to recover from memory corruption, the by far most common critical software fault class in such environments. Due to the missing memory boundaries and process separation, the system cannot terminate and restart selected parts. Therefore, the only solution is to restart the entire device, which appears to be the generally accepted way of handling memory corruption. Forensics must deal with this behavior; especially since attacks on vulnerable code (e.g. buffer overflow exploitation attempts) cause memory corruptions.

Monolithic device firmware must support the collection of evidence through an explicitly implemented functionality, as no standard interfaces exist to introduce the feature. The functionality available is often designed for the vendor's developers and not for a forensic investigator. At the time of this writing, the author is not aware of a single monolithic network device that explicitly offers evidence collection functionality for the purpose of forensic analysis. However, developer functionality, although largely undocumented and unsupported, can be used to achieve the same goals of full volatile date collection.

3.2 The Embedded OS Device Class

The second large device class uses existing operating systems designed for embedded platforms and implements the desired functionality as part of the system. Those systems generally do have a separate kernel and user land processes, concurrent schedulers and virtual memory with inter-process boundaries. They also provide more or less open interfaces, as the embedded operating system is normally not implemented by the same vendor and is therefore designed for flexibility rather than a unique use case. Typical examples of this class are Juniper routers running a FreeBSD based system and the large group of home network equipment (e.g. wireless access points and DSL routers) running Linux.

Devices utilizing an embedded operating system as core provide better stability, as processes are usually separated from each other and can crash individually. All embedded operating systems support post mortem analysis of individual processes, which can be used by the forensic investigator to analyze and identify the cause of the crash. Unfortunately, access to post mortem analysis information is not exposed to the user or administrator by most vendors. Therefore, the forensic investigator must find other ways to access the information.

While vendors of this device class commonly do not support the addition of runtime code to their devices, it is considerably easier to do than with monolithic devices. This fact can be exploited by attackers and forensic investigators alike. While the attacker will strive to embed binary code into the operating system kernel or a critical process, the forensic investigator can use the build-in crash dump and logging mechanisms to obtain evidence from the device.

3.3 Access to Evidence

Wile the device classes are extremely different by software design and architecture, the forensic investigator will in both cases need to identify post mortem analysis features usually left for the developers of the device. Depending on the device, the feature will likely provide any of the following:

- External debugging access to the device for the purpose of obtaining a partial or complete memory dump
- A preconfigured location for full memory dumps
- A fault analysis summary in text form
- An on-system kernel debugger
- A network accessible kernel debugger

It is common to all methods mentioned, that a configuration and preparation of evidence collection must be performed before the to-be-observed event takes place. Currently, the only option left when the necessary preparations have not been completed before the event, is the immediate mirroring of the device's memory contents using methods similar to [Ap08], which is impractical for even the advanced forensic investigator.

It should be noted that none of the methods discussed in this section is currently in wide use or part of any best practice recommendation.

4 Types of Attacks

Network equipment is generally targeted by the same attack types as common operating system platforms are. However, in contrast to the later, network equipment is most commonly attacked using modified or specially crafted network communication messages. This includes modification of address caches of the ARP protocol, Domain Name Resolution or neighbor information about other network devices. Within the same class of attacks are injections of invalid information on network communication paths, commonly referred to as routing protocol attacks, where the attacker emulates network equipment himself and becomes part of the communication infrastructure. This type of attack is easily recognizable with existing tools (see 5).

The same type of attack can also be applied to influence the network equipment directly with the goal of achieving administrative access to it and being able to load modified binary code or configuration settings. For the attack to succeed, the network equipment must at some point in time request either parts of its binary code or its configuration over the network. Commonly, such requests happen at boot time and involve entirely insecure protocols, such as TFTP, in the process. If the attacker is able to redirect the communication path for the device to a system he controls, the attacker can provide arbitrary code or a configuration that will be loaded onto the device. Changes to the configuration are in almost all cases recognizable by existing tools and procedures, while changes to the runtime code require evidence collection methods as described in 3.3 and further analysis.

The second important attack type against network equipment is the exploitation of vulnerabilities in services provided by the equipment itself. The network equipment offers network accessible services, such as remote management access, protocol translation, protocol de-capsulation or informational services such as network time. Since the exposed services have to handle requests from the outside, they can be attacked by a malicious party. The types of vulnerabilities discovered in services on network equipment naturally mirror those found in other software, namely buffer overflows, integer overflows and format string vulnerabilities. All of them allow the attacker to corrupt the memory of the process exhibiting the vulnerability, hereby diverting the execution of binary code in a way favorable to the attacker.

While on multi purpose operating systems, the attacker usually injects binary code that, upon execution, will open a command line interface towards the network (so-called “shell code”), the approach taken is different with network equipment, as the attacker will want to change the behavior of the device in a non-obvious but permanent way. Therefore, the attacker modifies the binary code of the network equipment to divert execution into a small chunk of code provided by the attacker, hereby introducing additional behavior options. [Mu08] provides a discussion of such modification for Cisco IOS devices. It is not possible to discover modifications of the binary code with the regularly exposed user interface, even for the experienced network equipment operator.

5 Current Evidence Gathering

The gathering of evidence from network equipment is today limited to different types of remote monitoring and management.

5.1 Simple Network Management Protocol

The globally accepted standard for device management is the Simple Network Management Protocol (SNMP), as it allows querying exposed information from the device as well as changing the configuration settings at runtime remotely. While the protocol supports the generation of messages to a centrally located message sink in the network upon specific events, most information is obtained via query requests to the device. This limits the amount if information the operator is able to collect, since the network bandwidth is degraded by every query and therefore not available for customer traffic.

Another limiting factor, from a forensics point of view, is the type of information exposed via the SNMP Management Information Base (MIB). Vendors and network operators alike focus naturally on the network specific information and only include general system state information in the MIB. The internal state of the operating system software is not regarded as important or useful to the network operator. Therefore, SNMP rarely exposes data of that type.

5.2 Syslog

Another widely deployed information gathering approach is syslog. Syslog is a simple and purely event driven protocol that transports logging information to a remotely located host, where they can be stored in text files. Configured correctly, syslog allows for slightly better monitoring of the network device's software states than SNMP, as events can be generated by any part of the software and are free-form text strings. However, the information provided by most network equipment is still a small subset of the internal state.

5.3 Debug filters

A third method of evidence collection is employing vendor specific debug output filters. Usually available on the local or remote console of the device, vendor specific debug output can be enabled for specific subsystems of the software. This often provides more detailed information than SNMP and Syslog, at the expense of the device's performance. Enabling all available debug information could provide enough evidence to investigate an intrustion, but is rarely practical, since it degrades the device's performance to a fraction of its regular throughput.

5.4 Looking Glasses and Monitors

To monitor the data network equipment uses to make communication path decisions, operators use software on external hosts that participates in the routing protocols or uses mechanisms like SNMP to query the respective tables from network equipment. These so-called looking glasses and routing protocol monitors provide the operator with an overview of the network's state and greatly simplify the identification of misbehaving network equipment.

5.5 Traffic Accounting

Network operators must monitor their devices and bandwidth for signs of overuse. They also must be able to correlate network traffic to customers, since most billing models depend on the traffic generated. Therefore, almost all larger networks make use of accounting information generated by the network equipment and sent off to centrally located accounting systems. An example of such functionality is Cisco System's NetFlow.

Accounting records may include data on traffic type, source network addresses and physical ports used. In correlation with other data available, accounting records can be a valuable source of evidence, since they are collected for billing and therefore the collection mechanism has to fulfill requirements that make billing legally enforceable. Accordingly, these mechanisms are implemented so that tempering with the records is detectable and access to them is closely monitored, giving the forensic investigator a rare case of solid evidence.

5.6 Summary of Current Methods

The currently widely deployed methods allow the detection of changes in the communication paths of the network. The evidence collected by the methods described can be sufficient to show the source of the communication path changes. However, it is often hard to establish if the sender intentionally or inadvertently caused that change. This is a design limitation of currently deployed communication networks. However, it can be stated that evidence collection and therefore the respective analysis can be conducted in reasonable depth, if the attack only influenced the communication path logic of the network equipment.

Whichever of the currently deployed methods is considered, none of them provides detailed enough evidence to perform an analysis regarding attacks that affect the network equipment's software. It is therefore hard to identify malicious network equipment software changes or single out software failures caused by intentional attacks against those caused by functionality issues. In the rare case that logging information provides enough indicators to conclude an attack could have taken place, the forensic investigator is currently at a loss for means of collecting the full evidence for further analysis.

6 Forensics for Monolithic OS Designs

To augment the methods described in 5, the network equipment family with the largest market share, Cisco System's IOS routers and switches, was analyzed for possibilities of significantly improved evidence collection mechanisms and procedures. The focus is on a separation of the evidence collection and analysis steps.

Evidence collections should be simple and easy to understand, preferably using a mechanism comparable to accepted methods such as hard drive imaging. The result should be unaltered and complete. The analysis tools, working on copies of the evidence, shall allow the dissemination of the evidence to a point where intrusions can be clearly detected and shown, including any modifications of binary code or critical data structures.

6.1 Evidence Collection through Memory Dumps

As already mentioned in 3.3, access to evidence in closed network equipment is normally achieved by means originally developed for the vendor's developers. In the case of Cisco IOS devices, the functionality used is the creation of full system memory dumps. The devices can be configured to write an unaltered memory dump of all mapped system memory onto a flash memory card or a remote host.

The advantage of the method is the ability to create memory dumps at any point in time during runtime of the device as well as upon critical software failures that cause the system to restart. In the event of a system restart, the memory dump is the only way to preserve any evidence from volatile memory.

The disadvantage from a forensics point of view is the reliance on functionality of the potentially compromised system to obtain the evidence. Technically, the modified device software can write arbitrary data instead of the memory dump, if the right code region has been modified by the attacker. Additionally, carefully crafted device crashes can leave the device in a state that doesn't allow it to write memory dumps anymore, as all information about memory organization and structure is already lost.

6.2 Evidence Collection through GDB

Cisco System's IOS devices offer another undocumented developer access to their system: the GNU Debugger (GDB) serial debug protocol, a widely used text based protocol for driving embedded kernel debuggers. The protocol relies on a GDB debugger stub in the target device; a very small code element that handles basic commands received from the serial console port.

Using the GDB debugger stub, the forensic investigator can obtain partial or full memory dumps from an IOS device by connecting a respective dump tool and halting the device's execution. Only the very small GDB stub will be executed when the device is in kernel debugging mode.

This presents another method of obtaining on-device evidence. The method's clear disadvantage is that it cannot be executed automatically upon critical system failures and requires a specialized software tool to be connected through the serial console port. Additionally, the method is extremely slow and therefore time consuming. Its advantage lies in the possibility to query arbitrary memory addresses and as an alternative for the full memory dump process, due to the GDB stub's functionality of halting execution and reporting CPU exceptions directly to the connected inspection tool, presenting a way to capture otherwise unhandled cases.

6.3 Analysis

Evidence colleted using the methods presented in 6.1 and 6.2 consists of a raw copy of the system memory. To analyze the obtained data, an analysis tool must be able to determine the former memory structure and gradually recreate abstraction from pure data. Based on the recreated data structures and their relation to each other, the analysis tool must be able to distinguish patterns of abnormal behavior, memory corruptions, modified binary code and additional functionality running on the inspected device at the point of the evidence collection. Such analysis tool has been developed and made available for Cisco IOS devices [Li08].

6.4 Abstraction Recovery

For the initial recreation of the memory layout, an analysis tool can utilize information from the device's software. The device software is in almost all cases available as separate package, allowing users and operators to upgrade their devices. The software distribution contains all information about the internal memory layout of the device. Using reverse engineering techniques, an analysis tool can identify the intended memory layout of the running system and compare it to the obtained evidence as well as to the limits the CPU platform of the device imposes on memory layouts.

Given a successfully reconstructed memory map, the analysis tool can search for patterns of well known data structures. The information base must be created manually from reverse engineering results, as the device vendor usually does not publish much detail about internal data structures. When data structures are found, their information can be added to a dynamic knowledge base, which in turn can provide more abstract information to higher level analysis code.

6.5 Backdoor Detection

Once sufficient abstraction and detail is built up, the detection of attack footprints can be performed. The tests performed are, by nature, very specific to the device type and device software employed. A generic test applicable to all device classes is the comparison of the binary code segments identified in the evidence to the same segments in the device's original firmware. If a difference is detected, the runtime code was modified. Since self-modifying code is rarely used on embedded devices in general, the differences highlight areas for future reverse engineering to the forensic investigator [Ci08]. Other detections include the inspection of process stacks for signs of code redirection into non-code areas of the memory, heap integrity checks and extraction of background scripts.

6.6 Traffic Extraction

Given a full memory dump of the device in question, a successfully implemented advanced forensics method is the extraction of data packets residing in the device's memory at the time of the evidence collection. In case the evidence was produced due to a crash of the device, this greatly increases the chances to identify both the offending data packet and the source of the offense.

In a first step, responsible data structures are inspected for information on the memory location of network protocol packets. Identified data packets can easily be extracted into a common file format for network traffic analysis, such as the widely used PCAP format, and analyzed by the forensic investigator further in publicly available tools (e.g. Wireshark).

Once this list is obtained and verified, the process stacks and other user memory areas can be inspected for data that directly references the responsible network data structures. If the device's firmware tracks ownership and references of memory blocks, these can also be used to reference code functionality to data handled. If relations between binary code of the device's software and a data packet can be shown, the forensic investigator can inspect the disassembled binary instructions for signs of a vulnerability exploited by the data packet in question. This allows to uniquely identifying vulnerabilities exploited, even if they are not previously known to the vendor or the general public.

7 Conclusion

The Internet Protocol based infrastructure used worldwide today is unable to cope with single compromised network nodes by itself, not even with manual intervention. Security protocols allow the detection of mischief in such networks, but don't present a way to resolve the issue once detected. Given the tremendous importance of network communication and availability in today's world, it can be argued that users will be forced to accept to work across compromised infrastructure if the alternative is to be offline.

Forensic investigators must be urgently put into the position to perform on-demand in-depth analysis of potentially attacked and compromised network equipment, as it is likely that attacker focus will shift in that direction. The lack of methodologies and tools for performing forensics on network equipment must be overcome to be able to detect, analyze and trace infrastructure attacks as well as attackers.

The methods presented in this paper have been shown to work well for the specific subset of Cisco IOS network devices, but may be applicable to a large range of monolithic devices, given comparable memory access methods. The incremental analysis approach allows producing compact and targeted information for the forensic investigator and preserves the evidence in its unmodified form. While the developments in this area are fairly recent, the author believes that network infrastructure forensics abilities will become crucial in the future.

8 Bibliography

[Wi08] Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Computer_forensics

[Di08] Jerry Dixon, National Member Alliance's Vice President for Government Relations, Infragard: Keynote - Quest for the Holy Grail, BlackHat Briefings Washington DC, 2008

[Ph00] Anonymous: Phrack Magazine, Volume 0xa Issue 0x38, 0x10[0x10], 05.01.2000

[Fx02] FX: Burning the bridge: Cisco IOS exploits, Phrack Magazine, Volume 0x0b, Issue 0x3c, Phile #0x07, 28/12/2002

[BC04] Brown, D.; Clore, E.: Honeynet Scan of the Month 31 submission, http://honeynet.opensourcecommunity.ph/scans/scan31/sub/doug_eric/, 2004

[Te06] Anonymous: Forum entry “GET /level/16/exec/-///pwd HTTP/1.0 is this a hack attempt?”, http://www.techienuggets.com/Comments?tx=420, 2006

[CVE01] Common Vulnerabilities and Exposures: CVE-2001-0537 “IOS HTTP Authorization Vulnerability”, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0537

[Po81] Postel, J (Ed): Internet Protocol, DARPA Internet Program, Protocol Specification (STD0005), September 1981

[DR06] Dierks, T.; Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.1, RFC4346, April 2006

[DD03] Doraswamy, N., Harkins, D.: IPSec – The New Security Standard for the Internet, Intranets, and Virtual Private Networks. Second Edition Auflage. Prentice Hall, 2003

[Ap08] Applebaum, J. et. al.: Lest We Remember: Cold Boot Attacks on Encryption Keys, Princeton University, Electronic Frontier Foundation, Wind River Systems, 2008, http://citp.princeton.edu/memory/

[Mu08] Muñiz, S.: Killing the myth of Cisco IOS rootkits: DIK (Da Ios rootKit), Core Technologies, March 2008

[Li08] Lindner, F.: Developments in Cisco IOS Forensics, January 2008, http://www.recurity-labs.com/content/pub/Recurity_Labs_Whitepaper_Cisco_Forensics.pdf

[Ci08] Detected Cisco IOS rootkit using text segment comparison: http://cir.recurity.com/cir/case.ashx/120EF269A5BC2320730E60289A4B84D9047CECEE/report-detailed.html

A Forensic Computing Framework to fit any Legal System

SSG Steven W. Wood
US Army, Retired
steven.wray.wood@us.army.mil
"The opinions or assertions contained are those of the writer and are not to be construed as official or reflecting the views of the United States Army."

Abstract: The demand for examinations of digital evidence is on the rise the world over. The majority of the academic work in this field comes from the United States and is heavily geared toward the American legal system. This makes using such a framework very difficult in other countries where the legal system can be fundamentally different. This paper proposes an extended framework that can be used in any jurisdiction in the world. It is our goal to take the existing state of the practice and add a level of abstraction that will increase its usefulness.

1 Introduction

The processing of digital evidence is a relatively young and extremely dynamic scientific field. While great strides have been made in the development of standards and procedures there is still not even an agreed upon name for what it is that we do. The author has come to prefer the term *forensic computing*, which is based on the experiences gained over the last 9 years of doing this work full time. Forensic computing is generic enough of a term to encompass all digital evidence that might need to be examined, yet detailed enough to prevent confusion as to what is being done. Stated in simple terms a computer is used to process evidence that will be used in a court to answer a legal question. Sheldon used the term forensic computing in his look at the diversity of the field. [SH01] Even if we examine a cellular telephone for example, we will need a computer with special software to complete the work. The same could be said for looking at evidence contained on the hard drive of a networked refrigerator.

There are numerous frameworks in the literature of which no single one seems to cover all of the areas that are encountered on a daily basis. The majority of these frameworks seem to be written from the perspective of a single person who is tasked with working an incident from the very beginning to the end. In the real world this is rarely the case, with many people working together to complete the larger task. Current frameworks are further not flexible enough to be applied to differing legal systems, as they seem to be written exclusively for use in the United States. Finally the majority of these frameworks are geared toward purely computer crime incidents when in fact examinations are performed for many other crimes on a regular basis such a rape, murder or terrorism.

This paper will work to define a new framework based on the experiences gleaned from over 1000 cases processed in a government contracted computer crime lab.

1.1 Background

Numerous models have been proposed for digital forensics over the last decade. In a paper written by Pollitt [PO01] there are no less than 15 models that are examined. The common theme between each of these models is that they all appear to be written overwhelmingly for the United States legal system.

It must be noted that many of the biggest names in the traditional forensic fields (Mathieu Orfila and Alphonse Bertillon of France, Hans Gross of Germany and Albert S. Osborn of the United Kingdom to name just a few) were not only non-American, but they performed their work in their respective home countries. A truly universal framework would therefore need to be abstract enough to apply in any country in which it was to be used.

There are numerous scientific principles when dealing with any discipline of forensics that are universally applicable regardless of the country or jurisdiction they are being used in. The difference that we will see is in how these principles are applied, which will be done in accordance with the local legal code and current jurisprudence at that location. While many legal systems are similar in nature, they can vary greatly in content and interpretation. This is evident from the fact that in some countries pictures of children being sexually abused are still not illegal. [ICM01] We cannot take for granted that the way things are done in the United States, or wherever else one lives, is the same around the world.

This paper proposes a new framework for forensic computing professionals that will offer the following:

- International applicability by ensuring enough abstraction to allow any legal system to be supported.
- The framework will give the ability to improve completion times of cases by optimizing the workflow.
- The framework will concentrate on what needs to be done and not how the person should do it.
- The framework will be equally applicable to any type of examination or investigation being conducted whether law enforcement or civil in nature.

1.2 Outline of the Paper

Section 2 of this paper will cover the overarching forensic principles that will surround all of the work being done by the examiner. Section 3 will discuss the framework itself and go into detail on the steps that will need to be taken by the examiner when working a case. Section 4 will touch briefly on the concept of skill levels that are for future work with the framework. Section 5 looks at a test case using the new framework. Section 6 talks about the benefits of the framework. Section 7looks at the future work planned by the author and Section 8 is the conclusion of the paper.

2 Forensic Principles

The model which we are proposing will consist of 6 general scientific principles that can be mapped to anywhere in the world and the corresponding legal code. These principles are not part of the framework itself in that they are not a specific step taken every time the framework is used. Instead they are always present in the background and must be observed by the practitioner at all times. The concept of principles in a forensic framework was proposed by Beebe and Clark. [BC01] These principles are to be thought of as continually running processes that have no clear starting or stopping point. It is fair to say that these principles wrap around the scientific process being performed in the actual steps of the framework. This abstraction shifts the framework from telling the user *how* to do something to telling them *what* they need to do.

We find it important to include these principles to ensure a form of quality control that is always present when the framework is being followed. The principles are not concrete steps that are performed by an examiner but are abstract in nature. How they are carried out, and to what standard, will be decided based on the location of where the work is being performed. The framework however does dictate that the principles are to be used.

2.1 Overview

We can see in the below figure what the principles look like. You can see that there is nothing new or earth shattering here. We are simply taking these common concepts out of the physical steps done by an examiner and abstracting them. This makes it possible to adapt to any set of rules or regulations that need to be followed.

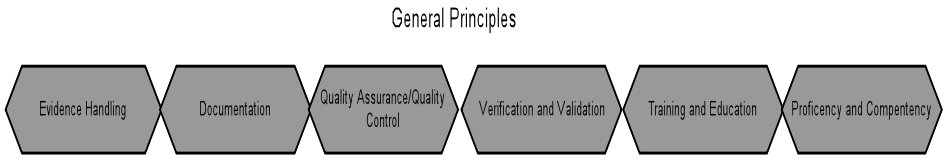


Figure 1: The six General Principles.

2.2 Evidence Handling

This is clearly one of the most important parts of any forensic process so why would it be listed under general principles? That is because it is not feasible to cover every possible combination in a framework for every country in the world. It must be the responsibility of the practitioner to make sure they are in compliance with the applicable laws and procedures of their jurisdiction.

What if for example one jurisdiction requires that two copies of all evidence be made on-site? What if another jurisdiction requires simply that the person making the digital copy of the evidence needs to take a picture of what they are doing? What if you are working a strictly intelligence-related case where evidence handling has no bearing at all? Any framework that tried to capture all of these different scenarios would be overly complicated and simply unusable in the real world.

Being a general principle puts the responsibility entirely onto the practitioner to meet the current standards of their jurisdiction. They are the ones who will eventually have to justify their actions in front of a court of law. The standard they will be held to will be the one that is in effect for that jurisdiction and for the type of investigation being performed.

2.3 Documentation

This extremely important part of the forensic process must be considered a principle because how it is done may not only vary between countries, but even between labs in the same jurisdiction. Generally not only must paperwork be accurately filled out and maintained, but the form and content will depend on local requirements. There is no magical “universal forensics form” that has been agreed upon by everyone worldwide that we know of. Information that needs to be recorded at one lab could actually be unnecessary for the next. The examiner will have to follow the rules where they are at and not just make it up as they go along.

The critical requirement here is that the forensic process can be recreated at a later time based entirely upon the documentation that has been kept. A typical forensic lab will not only have hundreds or thousands of cases that they work each year, but it could be many months, or even years, until a case goes to trial. Without the ability to look at the documentation there would be no way to accurately testify as to what was done. The documentation is the only permanent record an examiner will typically have.

2.4 Quality Assurance/Quality Control

Due to the fact that most labs will work differently in performing even the same tasks, there is no single answer for Quality Assurance/Quality Control (QA/QC). The QA/QC program will ensure that the work performed in the lab is in accordance with their documented policies and standards. Results will need to be checked for accuracy and any noted deficiencies will need to be fixed in the processes. This is a very important function needed to maintain the integrity of the entire forensic process.

It is also a responsibility of QA/QC to analyze processes and make recommendations for improvements as needed. Time brings with it new technology which will need to be incorporated into the operating procedures. These will need to be extensively tested and well documented. Experience can also lead to minor changes in certain scientific testing procedures and QA/QC will need to track these and ensure these are put to paper.

2.5 Verification and Validation

This principle differs from QA/QC in that it is not interested in examining policies and procedures but strictly concerns results of tests. If for example a SHA-256 hash value is calculated for a file there must be a way to make sure that it is correct. QA/QC will need to have policies and procedures in place for making that possible. Going through those steps to actually check the results will be the verification.

Validation deals with the results and output of the various tools and procedures being utilized. It is the sole responsibility of the practitioner to make sure that their tools and results are correct. When new versions of software are released they must be validated to make sure changes to them do not produce unexpected results. Validation is an ongoing process and not some single step in performing a forensic examination.

Validation will also need to be conducted on the tools being used to perform the examination. When using any software many events may cause the program to enter an error state in which it can produce unexpected results. These unexpected results may or may not be correct. If the examiner uses these results as the basis for making a finding it is highly possible that it will be wrong. This means that for each new version of Windows for example, tests will have to be conducted to see if the forensic software tool being used produces the same results as before. Changes to the underlying operating system can cause us to see different results even when using the same version of the examination software.

The same validation must be conducted on the forensic tools themselves used by the examiner. Vendors tend to release small updates to their software on a periodic basis. It must be noted that most of these updates are done to address bugs found during the use of the product. Validation will show if the fixes cause any other output to change. During validation it is also possible to determine, by using known sample data, if a new tool is working as it should. No one would want to just open the box of a new tool and jump right into doing casework with it. What if you did and found out later that a flaw made all of your results invalid?

2.6 Training and Education

There is no end to the learning process regardless of the forensic specialty being practiced. Researchers find new ways of performing existing tests while new technology will bring with it new possibilities. DNA testing is a prime example of this concept. Without continued training in the newest techniques a practitioner will not learn about the changes in the science. Attending training from a variety of providers can give the practitioner a broader knowledgebase from upon which to draw.

Under this principle would also fall the attendance of conferences related to the practitioners primary forensic specialty as well as general forensic-themed gatherings. These events are a great way of exchanging ideas with peers and learning what others are doing. The open exchange of information is a wonderful way of getting ideas looked at by other professionals in the community as well as getting their opinions to any questions you might have.

In forensic computing it is best to attend training covering general subjects as well as specialized courses covering software suites. This gives the student a solid foundation on which they can build their knowledge for using the individual tools. The Certified Computer Examiner Bootcamp would be a good choice for the general requirements. [CCE01] This weeklong class covers basic forensic practices as well as showing the student what is happening at the lower levels of the computer. Guidance Software offers numerous classes that will give a candidate the deeper understanding of how the tools being used work. [GUI01] These classes are geared toward the EnCase line of products but the knowledge gained can be applied to other tools as well.

2.7 Proficiency and Competency

This section is designed to ensure that the person doing the forensic work is skilled and able to perform to standards. What this means will vary from jurisdiction to jurisdiction. Be it a requirement for government licensing or mandatory yearly training, it would be captured under this principle.

There are numerous vendor-specific as well as vendor-neutral certifications available in the market. These certifications show that a person has the knowledge and ability to use a specific tool correctly. While not a perfect solution they are what we have available to us at the moment. Until something better does come along we will need to use these forms of skill measurement to their fullest potential. It is best for a practitioner to have a variety of certifications from multiple sources.

The examiner will need to keep documentation current that proves what they know and how well they are at doing their job. Many people have years of practical experience yet have never obtained a certification or college degree to prove it. This does not mean however that they are not highly qualified. Alternate evidence will need to be gathered in accordance with where the person is working. A comprehensive list of all work preformed is a prime example of such proof.

3 A Process Framework for Forensic Computing

The framework consists of 8 distinct steps that will cover the actual work that is to be performed. These steps will map more to what needs to be done and worry less about how it must be done. If a framework dictates how an examiner is to do their work they have no flexibility to adapt to the situation they find themselves in. This could make a framework unusable in an entire country if it is contradictory to the prevailing laws. Concentrating on what needs to be done, and letting the principles of the framework guide the how, makes it possible to overcome these limitations.

No investigation is conducted in a vacuum and the forensic examination of a computer is no different. There are steps that will need to be taken to ensure a complete, fair and reproducible examination. Most examiners will say they are afraid of missing information that might prove a person's guilt. We see it slightly different and think it would be worse if an exam was stopped too soon and evidence of a person's innocence was missed.

It should be obvious to most readers that there is no incident response phase included in this new framework. It was decided to leave this out because by far the majority of examiners will not ever be involved with this. Generally a victim detects that one of their systems has been compromised and then report it to the Police. The Police will get all of the necessary legal paperwork together and make a copy of the affected system. Once that is done the image of the system will be turned over to the lab for examination. Alternatively the victim could contact a third party vendor who then comes on-site to examine what happened. Either way the examiner will first become involved well after the incident occurs.

The steps which are detailed below are designed to be performed in the order in which they appear. This does not mean that in special situations some steps will change order or that some may even be left out completely. Each investigation is unique in the evidence it holds, what the person is accused of doing and what facts are being looked for. A case where we are trying to find pictures of abused children stored on a camera is very different from one where we need to find a deleted email on a server sitting in another country. That being said we still want to try and cover as many eventualities as we can without overly regulating what is happening. Forensic computing is much too fluid of a process to try and use any type of checklist to measure progress.

The below steps are similar to the process outlined by Carrier and Spafford. [CS01] In forensic computing we may be dealing with bits and bytes stored as either a 0 or a 1, but the actual work truly is no different than a traditional crime scene. Similar rules and procedures will need to be observed.

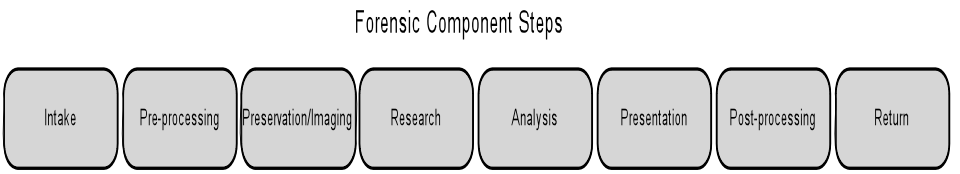


Figure 2: The eight Forensic Component Steps.

3.1 Intake

This is a step that will be taken before any work can be done. Regardless of the device being tested, or the procedure being done, it must be given to the examiner first. If a computer is the object it will need to be physically delivered. Even if a forensic image of the physical computer is stored on a hard drive as a file, it must still be given to the examiner.

The general principles above will dictate what paperwork is needed and how the evidence will be handled. We just need to know that no matter what it is we are doing, or what it is we are examining, we must physically take possession of it before work can begin. The item to be examined can be transported to us via a third party or we may seize it onsite with a search warrant. Once possession has been established the next steps can begin. There is nothing stopping the examiner from making copies of the evidence onsite under Step 3 and then bringing it back to the lab for examination and beginning with Step 1 once they have returned.

3.2 Pre-processing

These are the steps taken before the full investigation or examination process gets underway. Generally it will consist of gathering information needed by the examiner to complete the following steps of the framework. Reading the case file to get a full understanding of what is going on would be the most obvious example. A detailed history of the accusations being made can provide vital clues that will let the examiner narrow their focus quicker.

Another common example would be researching the technology to be examined. If presented with a new laptop model for the first time an examiner would need to research how to take it apart before grabbing the screwdrivers. The same can be required if the examiner is faced with an exotic hardware RAID controller which they have never seen before. Some of these controllers will not work without proprietary drivers from the manufacturer being used.

This process would also need to be taken if some special software package needed to be examined. In many economic crime investigations a customer management program or order processing system will need to be examined. While most of these programs will have the same output when given the same input, how they accomplish this under the hood can be very different. Knowing how the tool works means we can detect any attempts to circumvent internal controls or regulatory requirements. The author has seen suspects who have paid to have software modified to hide the true amounts in their bookkeeping systems when viewed by the tax authorities.

3.3 Preservation/Imaging

The evidence being examined must be protected to maintain it in the condition in which it was at the time of seizure or receipt. While an examiner may not always be the person who made the image of the device under examination, they must implement effective controls over the digital evidence as soon as they have it in their possession. The most common task to perform here will be the imaging of computer hard drives and other digital evidence. Hardly a case will be seen without a digital storage device of some sort in it. Other devices such as USB, Firewire and other storage media will also be captured in this step.

Actions to ensure that no changes to the original media must be taken independent of which tools or methods are being used. This can include such things as hardware write blockers, software write blockers and/or forensically sound boot media. No method is ever entirely 100% safe so proper procedures must be followed to minimize the risk of a write being made to the media.

3.4 Research

This phase begins once the examination has started and is one of the preliminary steps that will be taken. Once the examiner has had the opportunity to look at the evidence that is available a plan of attack can be developed. To do this properly the examiner will need to know what it is they are dealing with. Getting detailed information on the data types being looked at will need to be accomplished.

Under this step will also be determining what the subject of the examination is. Knowing the facts of a case is not technically necessary for a successful examination but will greatly improve accuracy and reduce time to completion. Getting information as to who is involved and what they are suspected of doing allows a plan to be made for the next step. If there are witnesses statements as to how the crime supposedly transpired it can help speed along the process.

3.5 Analysis

This is listed as a step but in reality it is a linear process. It starts at one end (the left hand side) with a huge amount of data that needs to be examined. Working from the pile on the left towards the right side the data is examined, leads are developed and followed and questions are asked and answered. Data that is not relevant will be excluded and additional evidence that needs to be obtained can be added. [FS01] The process continues to the right until the only thing left is the relevant information.

The actual process will differ between examiners based on their training and experience level. A person from an IT administration background will do this much differently than a person from an intelligence background for example. While the results may be nearly the same for both of them, the person with the intelligence background will likely be much faster and have a better chance of finding links between diverse tidbits of information that they observed.

3.6 Presentation

No matter what work was performed the results must be given to the requestor in some physical form. In many jurisdictions this can be in electronic form. In Germany we commonly store the results on electronic media such as Blu-Ray disks but must always submit a printed version. This is due to the fact that German Criminal Procedure requires all reports to be in a printed format. If it cannot be placed in the case file it does not exist.

There are rarely exceptions made to this rule even when many pages need to be printed. The author once had to submit a report to a State Supreme Court that detailed transactions contained in a large SQL database. Even by shrinking the report down and fitting 10 transactions on a single A4 page in landscape mode, the report consisted of around 17,500 pages.

The important thing to remember is that you must make the results understandable and put them in a form requested by the person submitting the work. It must further meet the legal requirements of the jurisdiction. You might do the best work in the world and crack the biggest case ever, but if you cannot get others to understand the facts it was all for nothing.

3.7 Post-processing

When all the analysis and presentation work is completed it will be necessary to perform the post-processing steps. These will differ from lab to lab and typically entail archiving the evidence and work product from the case. Local policies will dictate what needs to be done which can range from writing the data to CD or DVD to storing it in an encrypted container on a secure SAN.

In crimes involving graphics of children being sexually exploited sometimes the courts will order the offending material to be removed and the computer be returned to the owner. This happens mostly when a third party is involved or there were only one or two pictures total. In this case the examiner will need to forensically sanitize the relevant devices and verify the process before the work is completed.

3.8 Return

Once an examination is completed the evidence must be returned to the person who submitted it. The evidence will need to be handled in accordance with local policy and relevant legal codes. Every jurisdiction will be different and even a single jurisdiction could have several conflicting rules based upon the crime being dealt with.

In Germany a computer containing images of children being exploited will typically be seized by the government as a tool used in a crime. The same can happen in a fraud case where a computer was used to propagate the fraud. In the event that there is no evidence that the person did what they are accused of, the device will be given back to them. This situation will also occur when a person is found not guilty at trial.

Either way after all the work is completed the items received in Step 1 will need to be given back.

4 Skill Levels

Beebe and Clark made the point that most frameworks today are single-tier. [BC01] They propose having sub-phases that fall under these top-level phases. This theory is more in line with how work in forensic computing is actually performed. When doing imaging of a system for example there are multiple methods of differing difficulty levels that can be used. Having the multiple sub-phases allows the user to escalate through them until they are able to complete their task.

This section is for the future expansion of the model to allow us to extend this concept and to be able to measure the performance of practitioners. No single person can know everything nor can they be good at everything that they do know. It is for this reason that we need to break tasks down into different skill levels to make the testing of forensic computing professionals possible. The tasks that will be captured in these levels will need to be concrete in nature. This means the steps will consist of such things as making an image of a floppy disk. What is expected will be defined and well as what results need to be observed. Only after these concrete tasks have been defined can any true measurement take place.

In making an image of a floppy disk we could have the following methods map to the appropriate skill level:

Skill Level 1 – Use an automated floppy imaging device.

Skill Level 2 – Use EnCase to image the floppy.

Skill Level 3 – Use Anadisk to dump the contents of the floppy.

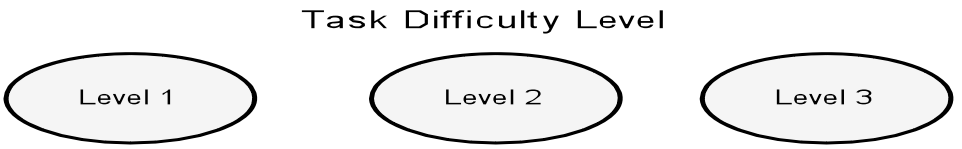


Figure 3: The three Skill Levels.

5 Test Case

In an effort to clarify how this framework functions we will use an example to demonstrate how the pieces work together. An investigation has been launched after a radio unit responded to a disturbance call at a multi-family house and discovered the body of a woman. The body was located in the bedroom of the top floor apartment. There was no evidence of forced entry and a laptop computer was found in a rucksack inside the closet. The apartment has a telephone and Internet service.

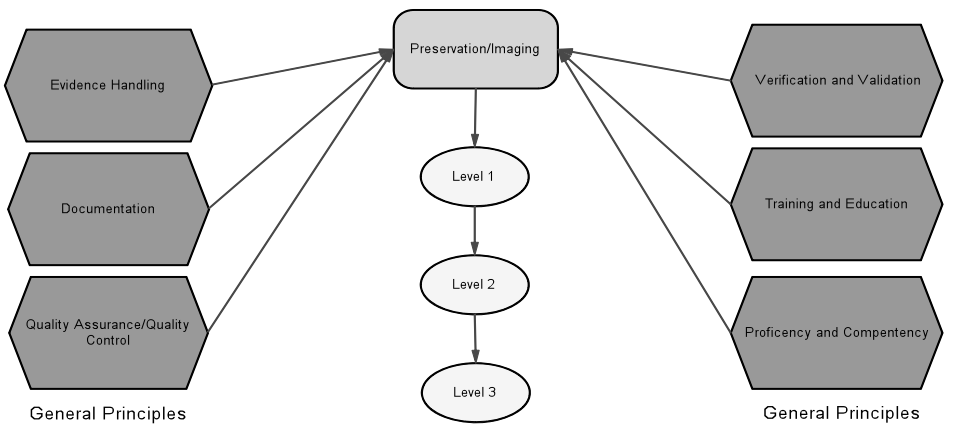


Figure 4: How it all fits together.

5.1 Forensic Process Step

The actual physical step we need to perform in this example is preservation/imaging. In this example the homicide detective took the laptop into evidence and transported it back to the station. At the station the detective personally brought the laptop to your office where you signed for it. Researching in the Internet you found out how to take the laptop apart and remove the hard drive for imaging. We will see below how the principles relate to the physical act of forensically copying the drive.

5.2 General Principles

Here we will look closer at the general principles we will follow when making our image of the hard rive removed from the laptop.

5.2.1 Evidence Handling

When the laptop was taken out of the evidence room the barcode on the evidence bag was scanned by the examiner as leaving that location. When the examiner gets to the imaging station the evidence management system was updated with the new location of the evidence by scanning the barcode again.

5.2.2 Documentation

Lab policy requires an imaging work form to be completed for the hard drive. This form is used to capture the model and serial number of the evidence among other relevant data. Where the image file is saved on the local computer file system will be noted and the status of the hash verification of the image will be written down. When done the form will be added to the case file.

5.2.3 Quality Assurance/Quality Control

QA/QC dictates that the hardware write blocker used for the imaging process will have initial and periodic testing. This testing will make sure that the device copies all data as expected and prevents all writes to the source media. QA/QC also covers verification conducted on the software tool used to make the image of the source drive.

5.2.4 Verification and Validation

These are the actual steps taken that are called for in QA/QC. A test set could consist of hashing the contents of a sample drive and then hooking it to a computer with the write blocker. Then the user can try to write files to the protected drive and then run another hash of the drive after rebooting the computer. If the values match the blocking device passes and can be used.

5.2.5 Training and Education

When a new tool is to be implemented the lab policy dictates that an examiner must be trained on it before they can use it. Examiners who have tested and validated the tool for the others can conduct internal training. When they have been trained to lab standards they will be able to conduct their own validation of the device.

5.2.6 Proficiency and Competency

This will capture the certifications the examiner has obtained. It will also be satisfied with the presentation of training records and internal testing results. The critical thing here is to show that the person doing the examination is skilled enough to do the work.

6 Framework Benefits

The major benefit of this new framework is that it can be used for investigating any crime where the processing of digital evidence is required. It is equally important that this framework is able to be used anywhere in the world regardless of the underlying legal system. By focusing more on results instead of telling the practitioner what they have to do we are giving them the flexibility to do their job better.

6 Framework Benefits

The major benefit of this new framework is that it can be used for investigating any crime where the processing of digital evidence is required. It is equally important that this framework is able to be used anywhere in the world regardless of the underlying legal system. By focusing more on results instead of telling the practitioner what they have to do we are giving them the flexibility to do their job better.

7 Future Work

Additional research is needed into what concrete tasks are needed for the actual forensic work steps detailed above. This will be the natural continuation in the process for making it possible to eventually measure the capabilities of a person working in forensic computing. Each of these tasks will need to be assigned an appropriate skill level based on polls conducted of professionals in the field. Additional papers will be released as the data is examined and results become available.

8 Conclusion

This new framework takes into consideration the work done by others in this field before us. Separating the principles out from the actual work steps gives us a layer of abstraction that makes the framework extremely flexible and robust. Even major changes in an existing legal system will allow the framework to continue to be utilized. This framework gives us a fresh way of working that is more closely aligned to the working environment in a traditional computer forensic laboratory. Finally it gives us the ability to use this framework anywhere in the world no matter the underlying legal code. Taken together we feel this framework is a starting point for improving the performance of forensic computing as well as the perception of its value by others.

Acknowledgments

I am very grateful to my Doctorate advisor, Professor Felix Freiling, of the University of Mannheim. He provided invaluable assistance by reviewing my writing and helping me to learn how to better put my ideas to paper.

Bibliography

[BCI01] Beebe, Nicole Lang, Clark, Jan Gayness, “A hierarchical, objectives-based framework for the digital investigation process”, Digital Investigation, Volume 2, 2005

[CCE01] http://www.cce-bootcamp.com/

[CS01] Carrier, Brian, Spafford, Eugene H., “Getting Physical with the Digital Investigation Process”, International Journal of Digital Evidence, Fall 2003, Volume 2, Issue 2

[FS01] Freiling, Felix, Schwittay, Bastian, “A Common Process Model for Incident Response and Computer Forensics", IMF 2007

[GUI01] http://www.guidancesoftware.com/training/course_listing.aspx

[ICM01] International Centre for Missing and Exploited Children, “Child Pornography: Model Legislation & Global Review”, International Centre for Missing and Exploited Children, 2006

[PO01] Pollitt, Mark M., “An Ad Hoc Review of Digital Forensic Models”, Proceedings of the Second International Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE'07), 2007

[RCG01] Reith, Mark, Carr, Clint, Gunsch, Gregg, “An Examination of Digital Forensic Models”, International Journal of Digital Evidence, Fall 2002, Volume 1, Issue 3

[SH01] Sheldon, Andrew, “The future of forensic computing”, Digital Investigation, Volume 2, 2005

Using Observations of Invariant Behavior to Detect Malicious Agency in Distributed Environments

Thomas Richard McEvoy Stephen D. Wolthusen

Department of Mathematics
Royal Holloway, University of London
Egham, Surrey TW20 0EX
United Kingdom
{T.R.McEvoy , Stephen.Wolthusen}@rhul.ac.uk

Abstract: Detecting malicious software used for covert ends is problematical because skilled attackers invariably employ stealth mechanisms to conceal the injection and subsequent activity of such software. As a result, the evidence of such incursions frequently "disappears" once the attack has succeeded. In distributed environments, this difficulty is compounded because of the inherent difficulties in observing the global state of a computation.

We propose a novel approach to the detection of potentially malicious activity in distributed environments. We select key data elements, which are chosen on the basis that they are frequently subject to subversion during malicious attacks. We specify their behavior as a partial order of sequences in state, accounting not only for legal and illegal states, but also for less than normative behavior, whose occurrence may indicate the presence of anomalous conditions.

We show how we overcome the difficulties of observing state in distributed environments through employing a multiplicity of distinct and independent observer processes and by making use of well-known algorithms to synchronize and order our observations and we demonstrate that we are able to use the resulting data set to make inferences about the presence (or not) of malicious software based on comparisons of observed and expected behaviors.

1 Introduction

Malicious software activity used for covert purposes such as the exfiltration of data is difficult to detect because it generally employs stealth and anti-forensics techniques – including direct attacks on detection mechanisms. As a result, evidence of its activity may only be momentarily available either during incursion or on subsequent activation (Section 6). On distributed systems, this problem is compounded because the global state of the system may not be observed directly, due to the computational unfeasibility (Section 2). This problem is an inherent feature of distributed systems and also applies to software engineering issues, such as debugging and deadlock detection [MG01].

We identify – for observation – data elements, which are frequently targeted for subversion during an attack (Section 3). We show how the behavior of these data elements relates to

the behavior of the distributed system as a whole (Sections 4.1, 4.2). We describe the expected behavior of these data elements by modified state diagrams, using knowledge of invariant behaviors in relation to causality, or system security policies (Section 4.3, 4.4). We create from combining these graphs, a "fuzzy" partial order over the combined sequences of these states. From this, we may induce a resulting set of total orders and associated probability values.(Section 4.5).

We implement a multi-pronged observation of these elements using a set of independent and distinct observer processes and utilize well-known distributed algorithms to synchronize these readings, to provide a logical time frame, and – from time to time - record the global state of the observation as a whole (Section 4.6). We derive from this set of observations a second "fuzzy" partial order of observed states and induce a corresponding set of linearizations. Since the predicted and observed sets are equivalent, by comparing these sets, we may uncover anomalous behaviors. (Section 4.7).

The results are partial and probabilistic in nature, but the validity and integrity of the data set is underscored by the multiplicity, distinctness and independence of the observers (Section 4.8). The mechanism is also capable of self protection, since it is itself also a distributed system which possesses key data elements which may be likewise observed as part of the mechanism's own working (Section 4.9).

We list some early results using a multiprocessor system to simulate some of the problems of a distributed environment (Section 5). We discuss related work (Section 6) and conclude with a summary of the advantages of our approach. We intend to explore its application to several types of distributed and parallel processing environments and we list some early results (Section7).

2 Detection of Malicious Behavior in Distributed Systems

A distributed system consists of a set N of processes, which are physically, or logically, remote from one another. Each process executes independently. They do not share memory and they exchange information using messages via agreed communication channels. These messages may be subject to delay and arrive out of order. Processes do not share a clock e.g. for time stamping messages or events [Gar02].

From the point of view of an all-seeing observer with a physical clock, therefore, messages and states associated with a run of a distributed computation would happen in a number of different total orders (or linearizations) – which may or may not be valid. This environment therefore poses a difficulty in determining – for example, for purposes of debugging – the validity of a given computational run i.e. detecting whether a particular condition (referred to as a global predicate) such as an "illegal state", is true, or becomes true during a run of a system. In particular, for unstable global predicates (conditions which become true momentarily, but do not necessarily remain true, unlike a stable condition such as termination), this problem is known to be NP-complete [Gar02]. Malicious software detection falls into this category (Section 6).

We require to show that we can reduce this problem to a computationally feasible one

that allows us to make valid observations and inferences about the global state of the distributed computation. Given that a skillful and knowledgeable attacker will be aware of our detection mechanism, we should also show that our approach is resilient to direct attack upon it.

3 Concurrent Observation of Invariants

We focus our attention on a set X selected data elements, which are commonly attacked by malicious agencies [1]. In addition, we select the data elements of X for the characteristic of invariance - that is, under normal operating conditions, it is possible to specify their behavior for all runs of a distributed computation. For example, they may be values which enforce system security policies, such as privilege flags associated with a user login, they may be measurements of systems activity such as the number of network connections, or they may represent fundamental features of the systems operation such as kernel structures. The selection is such that the features are unlikely to vary in operation over the lifetime of the system, or their behavior is well-managed such that all variations are accounted for during the lifetime of the system. It may, however, be necessary to add data elements to the set where there is a significant change in attack behavior.

We represent this behavior of these data elements as a partially ordered set of transitions in state with associated probability values for successor preference (section 4.3). By modeling two or more elements $x \in X$, we may specify causal relations between the states of these elements. Where the behavior of an element $x \in X$ replicates, these linearizations may be represented as a bounded n-tuple of states. From this predictor set, we can induce a set of all possible linearizations of the states of X.

This set of predicted sequences provides the probability space for transitions in state for these elements. It is also isomorphic to a "fuzzy" set, in that each bounded tuple may be present (as members) in the run of a distributed computation to a greater or lesser degree. Moreover, it not only allows legal and illegal behavior to be distinguished, but also provides a finely grained distinction between legal, but unlikely states, which may be indicative of error conditions, and legal, and reasonably probable, states, indicative of normal operation. It follows that if we are able to observe the states of the selected data elements and place them in an an equivalent structure, we have a basis for determining the existence of anomalous behavior, which may be a sign of malicious software activity.

We propose to observe the behavior of our selected data elements by using a multiplicity of distinct observer processes assigned to each data element(section 4.6). Since our observers also form a distributed system, we face the same constraints on process scheduling and messaging as any such system. But by using well-known distributed computing techniques for synchronizing and ordering the observations and for taking snapshots of state [Gar02], we are able to induce a set of possible partial orders of the states of our elements, along with associated probabilities.

[1] based on a knowledge of evolution in attack behaviors

Using our knowledge of system behavior, we may infer from this a set of possible partial orders with varying degrees of membership with regard to the behavior of the system. We compare bounded sequences from the set of total orders, which can be derived from this, with the sequences of the predictor set, and report any mismatches, including low probability conditions, as indicative of possible malicious activity.

We also outline how this approach provides a basis for self defence of the mechanism by allowing it the ability to observe itself. In addition, the employment of numerous independent processes provides a source of resilience and also acts as a deterrent to attack, since even with full knowledge of our mechanism, the attacker finds himself in a position where he is unable to observe, still less control, the operation of the mechanism. Thus, we manipulate the disadvantages of the distributed environment to our advantage.

4 Model Description

4.1 Modeling a Distributed Computation

A distributed system consists of a set P of N processes, $P = \{P_1, P_2, \ldots, P_N\}$. Each process P_i , where $1 \leq i \leq N$ passes through a finite sequence m of local states $(s_1, s_2, \ldots, s_3)$ where $m \geq 1$.

Let S_i be the sequence of local states in P_i. Then S can be defined as

$$S = \bigcup_i S_i$$

A distributed computation trace can be modeled as a tuple $(S_1, S_2, \ldots, S_N, \rightsquigarrow)$, where $\rightsquigarrow$ indicates a state in S_i logically preceding a state in S_j $(1 \leq i \leq j \leq N)$. This structure forms a decomposed partially ordered set (deposet).

A deposet, or run of a distributed program, defines a partial order (the "happened before" relation $\rightarrow$) on the set of states. There exist many total orders (also known as linearizations or global sequences) of this partial order which are sequences of global states where a global state is a vector of local states [Gar02].

We use the *interleaving assumption* for modeling states and events. That is, we do not model simultaneous behavior, but assume for any two concurrent events, say e and f, that e may follow f or f may follow e.

4.2 Modeling Data States

Let X be a selected set of data elements in a distributed computation. For our purposes, the elements of X are selected on the basis that they behave consistently (i.e. are invariant) under normal operating conditions, but behave arbitrarily under abnormal conditions such

as the incursion of malicious software [2].

Each element $x \in X$ passes through a sequence of states S_x during a run of a distributed computation. Let

$$S_X = \bigcup_{x \in X} S_x$$

It follows that if we are able to induce a partial order over S_X as before, then S_X will form a deposet.

We may model the behavior of P in terms of the behavior of X. That is, since each data element $x \in X$ is associated with a process $P_i \in P$, it follows that there is a monomorphism between S_x and S_i, in that for each state in S_x there are one or more state transitions of S_i. There exists, therefore, a mapping between the global states of S_X and the global states S.

Clearly, the model of behavior derived will be less finely grained than if we had direct access to the actual states of P, but we have the advantage of having a smaller set of behaviors to consider. Moreover, the model proposed allows us to consider behavior more directly relevant to the purposes of intrusion detection.

Thus, we argue that if we record the states of S_X we may feasibly compare the resulting set of global sequences (or parts thereof) with an equivalent set of predicted sequences and use our findings to uncover the presence of malicious software.

4.3 Specifying the Behavior of a Single Data Element

We may specify the legal states and permitted transitions of an $x \in X$, denoted $x_i \rightarrow x_j$, using a modified state diagram, which we call an *expected behavior graph* [3]. Where a choice of possible transitions exists, we may also, based on a knowledge of the system, label the edges to show the probability that a given transition will occur [4]. For convenience, we may choose to represent a replicated state as a new node and indicate this using an asterisk. See Figure 1 for an example.

This graph is useful as it enables us not only to distinguish between legal and illegal behavior, but also to identify low probability behavior which may be indicative of anomalous conditions. In effect, the specification serves not only as a description of the probability space Ω of the behavior of x, but also as a "fuzzy" partial order for the relation "preferred successor state" which expresses what behaviors are more likely that others to belong to a set of normal behaviors for the computation.

Clearly, it is infeasible for the graph to express a complete set of global sequences (or even one), which might occur during an extended run of the computation. Rather we designate

[2] Clearly, the selection criteria may be altered for different applications of the method illustrated in this paper.

[3] We do not use loops as we assume a data element remains constant unless acted on and we make some additional changes which are explained in the text.

[4] Similar to the concept of "likely" and "unlikely" in Linux kernel programs used to deal with error conditions.

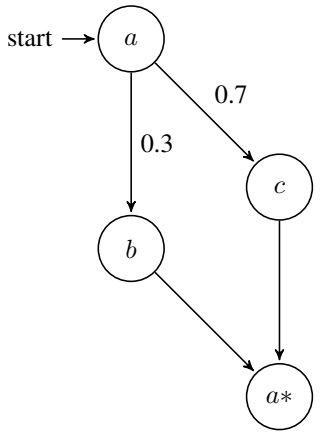


Figure 1: Expected Behavior Graph Showing Legal States and Transitions of x

the initial condition, the *maximum state* (or *maximal*, if it is repeated), any subsequent state which is replicated with its initial conditions, a *sub-maximal state*, and any terminal state as the *minimum state*, or *minimal states* if there is more than one. A set of global sequences may be modeled by chaining these bounded sequences together in various combinations. However, this is not necessary for our purposes.

4.4 Specifying the Behavior of Multiple Elements

We may extend this approach to specify the behavior of more than one data element concurrently, showing each data element as distinct components of the same expected behavior graph. This is meaningful where causal relations exist between the states of different elements. We formally define three such relations – *conditional dependency* , *strong causal dependency* and *weak causal dependency*.

These relations may be used to specify where the relation "potentially causes" ($\rightarrow_p$) must occur in our model, allowing us to induce a partial order over the predicted states of S_X, and hence deduce a set of corresponding global sequences, or rather parts thereof, which is consistent with the "happened before" relation [Gar02].

Where a data element $x \in X$ may only achieve a given state, say x_j, concurrent with a state of another element $y \in X$,say y_k, x_i is said to be conditionally dependent on y_k.

Definition(Conditional Dependency)

Let $x, y \in X$ be data elements. Let $x_i, x_j \in S_x$ be states of x and $y_k \in S_y$ be a state of y respectively. If x_i may not transition to x_j unless y is in y_k, we say that x_j is ***conditionally dependent*** *on y_k.*

We show this relation as a directed edge, which is dashed, on our expected behavior graph from y_k to x_j. y_k is called the *permitting node* of x_j. x_j is called the *dependent node*

of y_k. A joint dependency may also exist, shown by linking co-permitting nodes with an undirected dashed edge. This is called a *permitting component*. Clearly, a permitting node is also a permitting component (see Figure 2).

Two other forms of dependency may also exist where a transition in one component mandates a transition in the other. We graph this by showing a graph with both the "master" and "slave" components and directing an edge from the "master" component to the "slave" component (again, see Figure 2).

Definition(Strong Causal Dependency)

*Let $x_i, x_j \in S_x$ and $y_k \in S_y$ be states of $x, y \in X$. If the transition to y_k forces x_i to transition to x_j, and the transition to x_j only occurs due to y_k, we say that y_k strongly causes x_j, and we call this a **strong causal dependency**.*

The definition for *weak causal dependency* is similar, but removes the condition of uniqueness.

Definition(Weak Causal Dependency)

*Let $x_i, x_j \in S_x$ and $y_k \in S_y$ be states of $x, y \in X$. If the transition to y_k forces x_i to transition to x_j, and the transition to x_j may also occur as a result of other conditions, we say that y_k weakly causes x_j, and we call this a **weak causal dependency**.*

In some cases, a two-way dependency may exist. For example, if two processes are in communication, either process may end the conversation, forcing the other process to do the same.

4.5 Inducing a Partial Order Over Predicted States

Clearly, therefore, we are capable of inducing a partial order over a predicted set of states which when combined will allow us to model the possible global sequences of S_X. We denote this the *predictor set* R_X. This will consist of a set of tuples $(R_x, R_y, ..., \rightsquigarrow_p)$ – where $\rightsquigarrow_p$ indicates the relation (remotely) "potentially causes", which is consistent with the "happened before" relation [Gar02].

For example. Let $(\ldots)$ indicate an ordered, bounded n-tuple of states. Let $[\ldots]$ indicate a permutation of states. Let $\langle(\ldots), (\omega_i)\rangle$ be an ordered pair which indicates the probability of an ordered tuple. Let $\perp$ indicate process termination. Let $*$ indicate a replication of maximal states. Let $X = \{red, blue\}$ be two distinct data elements with causal relations between their respective states.

From Figure 2, the set of expected orders of red and $blue$ [5] are –

$$red = \{\langle(a, b, d, \perp), (0.03)\rangle,$$
$$\langle(a, b, e, a*), (0.27)\rangle,$$
$$\langle(a, c, e, a*), (0.28)\rangle,$$

[5] shown as white and gray respectively

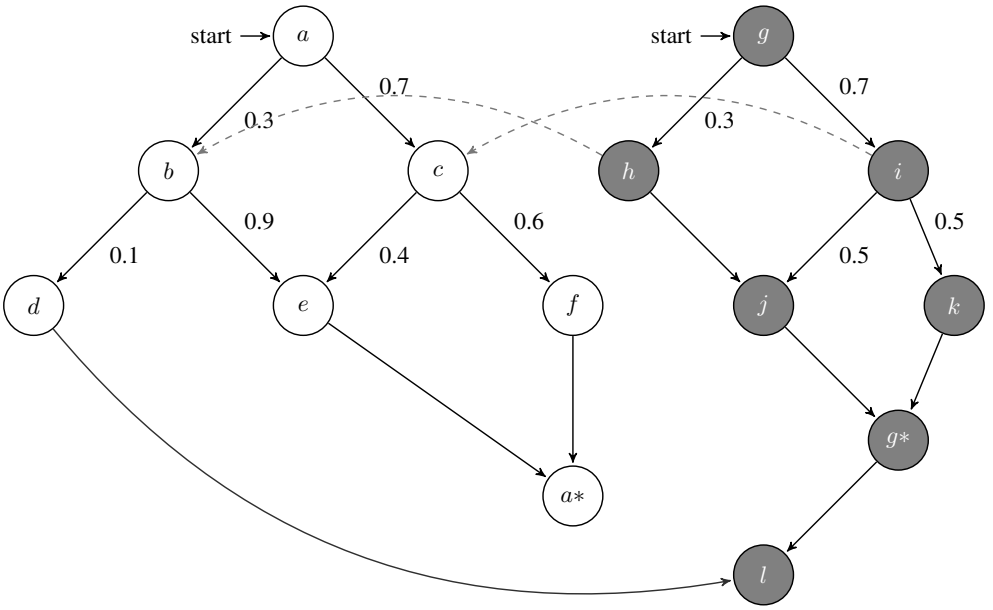


Figure 2: Expected Behavior Graph with Dependencies

$$\langle (a, c, f, a*), (0.42) \rangle \}$$

$$\begin{aligned} blue = \{ & \langle (g, h, j, g*), (0.3) \rangle, \\ & \langle (g, i, j, g*), (0.35) \rangle, \\ & \langle (g, i, k, g*), (0.35) \rangle \} \end{aligned}$$

while the set of partial orders for X, based on the causal relations which exist between these sets of sequences, consists of

$$\begin{aligned} R_X &= \{X_1, X_2, \ldots, X_6\} \\ &= \{ \langle ([ag], h, b, d, l, \perp), (0.03) \rangle, \\ & \langle ([ag], h, b, [ej], *), (0.27) \rangle, \\ & \langle ([ag], i, c, [ej], *), (0.14) \rangle, \\ & \langle ([ag], i, c, [ek], *), (0.14) \rangle, \\ & \langle ([ag], i, c, [fj], *), (0.21) \rangle, \end{aligned}$$

$$\langle([ag], i, c, [fk], *), (0.21)\rangle\}.$$

Taking the element of R_{X_2} in the order shown as an example, using the interleaving assumption, the set of linearized sequences would be

$$R_{X_1} = \{(a, g, h, b, e, j, *), (g, a, h, b, e, j, *), (a, g, h, b, j, e, *), (g, a, h, b, j, e, *)\}$$

,

each of which may occur with $p = 0.675$. The boundary states are the maxmimals a and g and the minimals d and l, respectively. The information provided is sufficient to model a global sequence, if required.

4.6 Observing Data Element Behavior

We show that we can observe the states of X in an order which closely approximates the logical order in which they occurred. To do so, we create a set of observer processes and assign to each data element a partition (or ensemble) of these processes, which takes periodic measurements of its state.

We weakly synchronize our observations by requiring that each observer in an ensemble exchange messages post-observation with every associated observer before proceeding to the next round of observations [Gar02]. This forces the condition that each set of observations must be temporally distinct from each previous round of observations since any process having completed its observation and messaging must wait until it has received from every other process a message concerning their observations before continuing. Hence all observations must have taken place in that round (as there cannot be a message about an observation which has not taken place) before the next round of observations commences.

Messages concerning observations enable us to uncover mismatches in state as the result of each observation is compared by every observer process with its own measurement. We assume that a mismatch indicates a change in state between two distinct observations. This means we avoid the need to log the values from every round of observations and need only log each mismatch in observations.

Observations are not continuous but by using a multiplicity of observers decrease the probability of missing a transition (section 4.8). The order in which transitions are observed and reported within a round for a single data element is not necessarily sequential. However, the order in which transitions are logged across rounds is necessarily sequential for the reasons already discussed in this section.

We use a vector clock [SC02] to logically time stamp messages in accordance with the "happened before" relation [TG98], so that communications regarding observations may be ordered. We use the "pulse" index of a synchronizer algorithm [Gar02] to allow us – if necessary – to identify messages arriving ahead of order to be buffered for future processing. From time to time, we also employ a snapshot algorithm [HS97] to synchronize

the vector clocks between ensembles as part of logically ordering interactions between the states of distinct data elements, using the "happened before" relation.

Hence, for each data element, the results of observation rounds are logged in order, while a partial order of events may be established between the measurement rounds of distinct ensembles, using snapshots in state in combination with vector clock values.

4.7 Inferring Data Element Behavior and the Presence of Malicious Software

The partially ordered set of observed values which we denote T_X is only an approximation of the set S_X which we are able to create from our predictor set R_X for several reasons –

1. Several transitions may take place during a single round of observations, for which we have no basis in observation for determining the order in which they occurred (Section 4.6)
2. There exists a probability we may miss one, or more, transitions in state (Section 4.8)
3. Data may also be missing, or inaccurate, due to the failure of channels or processes, possibly as the result of deliberate action by the attacker, or as the result of spoofing
4. The partial order induced by the "happened before" relation using the vector clock in conjunction with snapshots of state does not map directly to the "potentially causes" relation we identified for R_X

We, therefore, need an approach which deals systematically with these issues and allows us to use the sequences derived from R_X in conjunction with the observed values of T_X to uncover malicious activity.

We deal with reducing the probability of failure, malicious attack and spoofing in section 4.9. In this section, we concentrate on using our knowledge of predicted behavior to make inferences about observed behavior and give some simple examples of how this can be used to uncover evidence of malicious activity. We also demonstrate that our approach is computationally feasible.

To do so, we use both the boundaries we have defined for our sequences and the relations we have identified earlier in creating the predictor set R_X – the "preferred successor" relation, conditional dependency, weak and strong causal dependency – as a lens through which to view the data set T_X. We give examples of various forms of reasoning which may be applied as a result.

Using the maximal, sub-maximal and minimal states we have defined in R_X, we can identify bounded n-tuples of states for single elements. Using the "preferred successor" relation, we may examine the transitions of each element individually. Where a transition is missing, that is, there is no observation of any valid preferred successor, we may infer from R_X a set of probable transitions – which may be presented graphically as a tree

(see Figure 3) – and test these for consistency with subsequent observations, reporting any contradictions, or pruning invalid branches. Additionally, given that the probability of missing transitions will be low, if several appeared to be missing, we would report this as indicative of possible failure in the mechanism, or other anomalous condition.

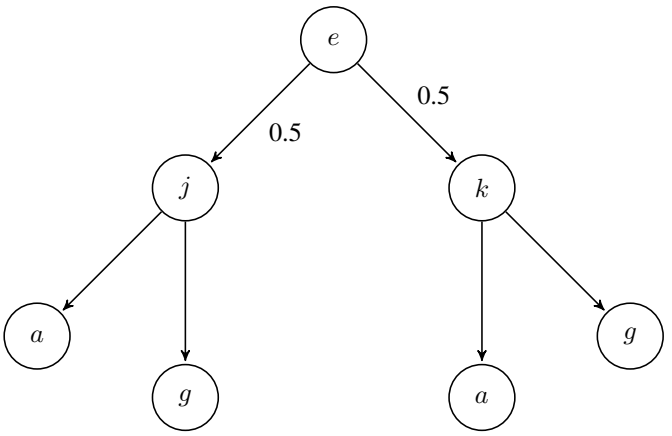


Figure 3: Part of a Hypothesis Tree (for missing observations of j and k) for red and $blue$

Similarly, where we observed several transitions during a single round, we may order these in accordance with the "preferred successor" relation, again presenting the results in a hypothesis tree, checking both that no two transitions lead to contradictory paths and, once again, against subsequent transitions in state for consistency. Once more, it may be possible to prune the number of combinations of transitions as a result of these consistency checks.

When we come to a joint consideration of causally related data elements, we are primarily checking for a consistency in the ordering of these causal relations, that is, that they are congruent with the "happened before" relation which is induced by the vector clock.

Taking strong causal dependency as an example (similar reasoning may be applied to the other relations), we assume that any "slave" state must be preceded by a "master" state . Hence for any two data elements – considering the states of each in order of observation – where we find an initial "slave", we look for an initial "master" state, which may be either observed, or, if missing from the set of observations, inferred – so long as its existence is consistent with the subsequent set of transitions in its associated data element. Similarly, where we find a "master" state, we may look for the "slave" state which is its consequence.

This analysis enables us to associate bounded n-tuples (i.e. sequences) of distinct data elements using "potential causality". We may proceed to check if this relation is consistent with the "happened before" relation over T_X by using vector clock values to link subsets of bounded sequences in a partial order. Where there is an apparent contradiction between the order induced by vector clock values and the order observed or inferred for our "potentially causes" relations, we log this as an anomaly.

In addition, we may also apply the "pigeon-hole" principle i.e. if there are, for example, more "slave" states than "master" states, this is also inconsistent with our predictor set.

It should be noted that, on occasions, establishing the partial order "potentially causes" will enable us to prune back on hypotheses concerning missing data, or multiple transitions in rounds, as the inference of a causal condition or of a permitted, or "slave", state may clarify what transitions have occurred. This is one of the advantages of our approach since it permits us to look at transitions in state from multiple points of view.

So from an initial set T_X of observations, making use of our knowledge of invariant behavior within the system in relation to transitions in state and causal relations between the values of data elements, we are able to infer a "fuzzy" partially ordered set of states of data elements, which we denote $T_X^{'}$, whose sequences and ordering may be checked for consistency against our predictor set R_X and for internal consistency. This gives us a basis for uncovering anomalies which are potentially the result of malicious activity.

As a final step, we may also give consideration to the overall state of the data set in the context of our analysis. We have already stated that low probability – that is, abnormal – conditions may result in an alert and that we might equally raise an alert where several sets of transitions appeared to be missing. In addition, we should also consider where various negative conditions – such as missing data, evidence of failed processes or low probability states - combine and establish a threshold of significance for the quality of the data set. Although this is an approximate concept, we hold that multiple caused failures in the data set may also be considered anomalous and used to trigger an alert.

The computational feasibility of our approach is demonstrated by the fact that we are using well-defined behaviors for our analysis; these behaviors may be considered in the context of bounded, replicating sequences and we can present the results graphically as a tree, therefore, making it possible to avail of well-known graph traversal algorithms to build linear sequences from our observations for analysis and comparison with the set of linear sequences derived from our predictor set R_X.

4.8 Observational Probabilities

There is a probability that one or more states of $x \in X$ may be unobserved. As we have seen, this adds to the complexity of drawing inferences from the data set. At the same time, the use of a multiplicity of observers significantly lowers the probability of this occurring.

Let L be an arbitrary length of time. We observe x using n observers where $n \geq 2$. Let there be a transition in x. We assume this transition δ takes zero time from the point of view of the observers [6] and may happen at any point during L. Let the probability of an observer in the round taking an observation of x before δ be p. Let the probability of taking an observation after δ be q. Let T be the event of observing a transition. If all observations in a round occur before δ or, equivalently, after it, no mismatch in observations occurs. Hence no mismatch in states is reported by the ensemble; this is equivalent to not observing

[6] i.e. either the observation is of state x, or of state x', as any interim state is undefined

it, and we see that $1 - P(T) = p^n + q^n$. Hence,

$$P(T) = \sum_{i=1}^{n-1} ({}^n_i) p^i q^{n-i}$$

This argument may be extended to the observation of further transitions of x during a single round, but it should be clear that where a multiplicity of observers exist, the probability of not observing a transition is much reduced. This reduces the number of possibilities we need to consider when making inferences about our observations.

4.9 Resilience and Self Defence

The multiplicity, independence and distinctness of the observers clearly provides a degree of resilience for the observer mechanism, although performance and messaging overheads need to be taken into account in setting the number of observers. In addition, if poor communications are an issue then some adjustments would need to be made to the synchroniser algorithm to take account of undue latency or communications failure, for example, a controlled time out on message waiting periods.

Considering the possibility of direct attack on the mechanism [Szo05], the major advantage of our approach is that the independence of the observer processes enables us to instantiate a subset of these processes to observe data elements associated with the observation mechanism itself. This enables periodic tests of the integrity of the mechanism. Combined with the possession of a multiplicity of independent observers, this acts as a deterrent to attack by requiring that the capability to simultaneously subvert a set of processes which may not even share the same logical or physical platform, or be functionally equivalent – even if observing the same data element.

A partial subversion of the mechanism is likely to fail because even if a subset of observers are subverted, or have their communications spoofed, this will appear to as a series of continual transitions in state whose permutations are unlikely to be equivalent to the expected behaviors of our selected data elements. If instead we consider an attack *en masse* where the attacker seeks to take advantage of a single vulnerability across an ensemble (or more than one) of observers, we find that this possibility is closed to them. Even within a single ensemble because the same data is observed from different points of view (representing either different APIs or different logical or physical platforms) requiring the use of distinct functionality, the likelihood that a single exploitable vulnerability will result in the mass compromise of the observer processes is low. The use of different platforms also implies that coding or systems operation details may be distinct, further adding to the attacker's difficulties.

Thus, the probabilistic nature of our mechanism acts as a deterrent to attackers since they may not easily predict where and when any actions of theirs may be observed, nor is it computationally feasible for them to seek to control multiple observers which work at different points in process time and space and which possess distinct functionality.

In other words, the constraints which we face in detecting malicious activity are redoubled for the attacker considering the subversion of a distributed computation which is guarded by our mechanism, even with full knowledge of the mechanism. Effectively, we have turned the attacker's apparent strength in being difficult to detect in a distributed environment "jiu jutsu"-like against him[7]

5 Experimental Justification

We summarize findings from an early "proof of concept" of our approach. We analyzed the action of several root kits on a Linux system and identified key invariants in the data structures which were altered by these examples of malicious software in order to subvert the system. A common example were pointers to system functions on the system call table, but more sophisticated attacks targeted other kernel structures such the VFS sub-system, or the Interrupt Descriptor Table (IDT). We used some of these example attacks to build a root kit simulator, which did not perform any malicious actions, but merely sought to conceal file-based information, such as might be disguised by malicious software.

We subsequently designed and implemented a kernel module application, based on a distributed computing architecture, called *KRAKEN* which made use of multiple observer processors to examine our selected data element from different points of view, that is, using different APIs for the purposes of measurement.

The application was based on a symmetric multiprocessing (SMP) system. Observer processes were assigned to different CPUs on instantiation, which not only ensured independence between observational viewpoints, but also introduced a non-deterministic element into process scheduling which we argue, similar to the model proposed here, would make evasion of the action of the anomaly detector problematical. The observations themselves were stateless, but each observer exchanged messages with its peers to compare results and reported any mismatches in state. The mechanism made use of the "synchronizer" algorithm, logical time-values and snapshots in state similar to the approach described in this paper (Section 4.6).

Several experiments were carried out using the root kit simulator to mimic different methods of concealment used by attackers. *KRAKEN* demonstrated particular success ($p \approx 1$) in dealing with root kits when observations were made at both high and low level APIs concurrently and the subversion method relied on concealing the true state of the data element at an interim level as the continuous contradiction between the observed values made detection inevitable. But even where the subversion method attacked a lower level API than any measured, there remained a significant probability of detecting the anomalies, with some dependence on the frequency of measurement (as high as $p \approx 0.6$ under realistic performance constraints).

Although there are some differences between the more tightly bounded environment of a SMP system and larger scale, more loosely structured distributed systems, this early work

[7] If the reader prefers Chinese martial arts, then he should consider the example of Choy Li Fut, where the student is trained to consider all objects in the local environment as weapons in the fight against his opponent.

demonstrated that the use of concurrent and distributed observations of data elements as an approach to ID justifies exploration.

6 Related Work

The difficulty of detecting an unstable global predicate in a distributed system is known to be NP-complete [Gar02]. Malicious activity may be so characterized since, on incursion, the attacker rapidly deploys stealth techniques, including the subversion of ID mechanisms [Szo05, HB05, Skl07, NW06], which conceal signs of its presence. For example, an email may be intercepted and malformed so as to inject malicious software onto a system and – as a first act of such software – the email will be restored to its original state[8].

Considerable research has gone into the observation of both stable and unstable global predicates on distributed systems, primarily motivated by software engineering requirements such as debugging [VD95] [MG01]. This research has led to the identification of various categories of observable predicates. It has established sophisticated and subtle means of identifying temporal, or causal, relations amongst them. This work has been accompanied by the creation of suitable algorithms, which are implemented from within the observed computation [MG95] [CK05] [Ksh96].

We distinguish our approach to observing a distributed environment by proposing a set of independent observers (processes) distinct from the observed system (Section 3) to take measurements of key parts of that system. Using well-known algorithms [HS97] [SC02] [Gar02] to determine the state of these observations and, from there – based on our knowledge of the system – to make inferences about the global state of the system.

Intrusion detection systems may be classified as host-based or network-based [FHL07]. They may use signature-based identification or heuristics [LPR07]. They may depend on statistical analysis for anomaly detection [KG03], or specify acceptable system behaviors, thus identifying unwanted behaviors [Ko00]. There has been an emphasis on switching analysis from examining usage anomalies to process anomaly detection - see [HFS98] for an early example. Our work carries similarities to this approach, but we distinguish it by focusing on the computational outcomes of processes rather than their operation and thus we abstract away from architectural considerations.

We also focus on the identification of causality between independent processes. Although previous work in this area appears to make implicit assumptions [KMLC05] which we do not believe are justified. The timestamps of events or its order are generally assumed to be correct, whereas we recognize that, even on small-scale networks and distributed systems, there are issues with understanding the timing and sequence of events and that too much may be made of the reliability of timestamps [GC06]. We use logical orderings of events to address these issues in line with recognized distributed computing paradigms [Gar02].

Moreover, much ID research focuses on attacks which are large-scale manifestations of malicious activity such as worm infections [KMLC05], rather than low-level covert infor-

[8] private communication

mation gathering – which over the lifetime of a system may be ultimately more damaging [LPKS05] [Emi06]. We believe our approach is uniquely suited to uncovering evidence of the latter, since even momentary misbehaviors in systems may become significant where "invariant" characteristics are violated and it is possible at a higher level of abstraction to link these breaches into a semantically meaningful pattern which may be used for attack identification, or forensics analysis and subsequent development of countermeasures.

7 Conclusion and Future Work

We have shown that by employing a multi-pronged observation mechanism and employing well-known algorithms from distributed computing, we may from a knowledge of the expected and observed states of key data elements, make inferences concerning the presence or absence of malicious software. Although the observation is inherently partial and the inferences may be considered probabilistic and approximate, the multiplicity of the observers underpins the validity of our approach. This last aspect also makes the mechanism difficult of subversion. The approach modeled also has the potential to detect direct attack upon it through self observation.

We consider the probabilistic nature of our mechanism (see Section 4.8) to be a deterrent since, while we assume the attacker has knowledge of our mechanism, such knowledge does not help him as he is not able to either predict or control the process of observation, and hence set himself to pass unobserved.

There are limitations on the currently proposed model. We currently require that the observers exchange a complete set of messages with each other during each round of observation. Clearly, where a large number of observers exist, or where communication is low bandwidth this could cause performance issues. This would need to be addressed through adjusting the synchroniser algorithm. Moreover, similar adjustments would be required to cope with poor communications (possibly due to malicious action) leading to messages being dropped. Nor have we formally addressed the observation of mobile processes, where new processes may be created, processes may terminate and channels similarly be created or destroyed.

The system would also require some mechanism for distinguishing genuine communications from spoofed messages. On high performance systems, this could be achieved using encryption mechanisms, but on low bandwidth or low powered systems this would not be feasible. Another approach is hinted at in section 4.9 where we show that the multiplicity of observers employed provides a basis for detecting contradictory messages and logging these as anomalous. As modern platforms migrate toward increased use of multiple processors on a single base, this justifies our approach.

We summarize some early results from a 'proof of concept' version in a multiprocessor system, which we are currently using to simulate some of the problems of working in parallel and distributed environments, and we intend to extend this work to similar larger-scale environments in future.

References

[CK05] Punit Chandra and Ajay D. Kshemkalyani. Causality-Based Predicate Detection across Space and Time. *IEEE Transactions on Computers*, 54(11):1438–1453, 2005.

[Emi06] Aaron Emigh. The Crimeware Landscape: Malware, Phishing, Identity Theft and Beyond. *Journal of Digital Forensic Practice*, 1:245 – 260, 2006.

[FHL07] Yingfang Fu, Jingsha He, and Guorui Li. A Distributed Intrusion Detection Scheme for Mobile Ad Hoc Networks. *COMPSAC '07: Proceedings of the 31st Annual International Computer Software and Applications Conference - Vol. 2- (COMPSAC 2007)*, 02:75–80, 2007.

[Gar02] Vijay K. Garg. *Elements of Distributed Computing*, chapter 1,2,3,4,10,11,12,20. John Wiley and Sons Inc, 2002.

[GC06] Ashish Gehani and Surendar Chandra. PAST: Probabilistic Authentication of Sensor Timestamps. In *ACSAC '06: Proceedings of the 22nd Annual Computer Security Applications Conference on Annual Computer Security Applications Conference*, pages 439–448, Washington, DC, USA, 2006. IEEE Computer Society.

[HB05] Greg Hoglund and James Butler. *Rootkits*. Addison-Wesley, 2005.

[HFS98] Steven A. Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security, 1998*, 6(3):151–180, 1998.

[HS97] Letian He and Yongqiang Sun. On Distributed Snapshot Algorithms. *APDC '97: Proceedings of the 1997 Advances in Parallel and Distributed Computing Conference (APDC '97)*, page 291, 1997.

[KG03] Oleg Kachirski and Ratan Guha. Effective Intrusion Detection Using Multiple Sensors in Wireless Ad Hoc Networks. In *HICSS '03: Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 2*, page 57.1, Washington, DC, USA, 2003. IEEE Computer Society.

[KMLC05] Samuel T. King, Z. Morley Mao, Dominic G. Lucchetti, and Peter M. Chen. Enriching Intrusion Alerts Through Multi-host Causality. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS)*, 2005.

[Ko00] Calvin Ko. Logic Induction of Valid Behavior Specifications for Intrusion Detection. *SP '00: Proceedings of the 2000 IEEE Symposium on Security and Privacy*, 00:0142, 2000.

[Ksh96] Ajay D. Kshemkalyani. Temporal interactions of intervals in distributed systems. *Journal of Computer and System Sciences*, 52(2):287–298, 1996.

[LPKS05] Michael E. Locasto, Janak J. Parekh, Angelos D. Keromytis, and Salvatore J. Stolfo. Towards Collaborative Security and P2P Intrusion Detection. In *Proceedings of the IEEE Information Assurance Workshop (IAW)*, June 2005.

[LPR07] Adrian P. Lauf, Richard A. Peters, and William H. Robinson. Embedded Intelligent Intrusion Detection: A Behavior-Based Approach. *AINAW '07: Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops*, 1:816–821, 2007.

[MG95] J.R. Mitchell and V.K. Garg. Deriving Distributed Algorithms from a General Predicate Detector. *Computer Software and Applications Conference, 1995. COMPSAC 95. Proceedings., Nineteenth Annual International*, 00:268, 1995.

[MG01] Neeraj Mittal and Vijay K. Garg. On Detecting Global Predicates in Distributed Computations. *21st IEEE International Conference on Distributed Computing Systems (ICDCS'01)*, 00:0003, 2001.

[NW06] Naoyuki Nagatou and Takuo Watanabe. Run-Time Detection of Covert Channels. In *ARES '06: Proceedings of the First International Conference on Availability, Reliability and Security*, pages 577–584, Washington, DC, USA, 2006. IEEE Computer Society.

[SC02] Chengzheng Sun and Wentong Cai. Capturing Causality by Compressed Vector Clock in Real-Time Group Editors. *IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium*, page 89, 2002.

[Skl07] Ivan Sklyarov. *Programming Linux Hacker Tools Uncovered*, chapter 21. A-LIST, LLC, 2007.

[Szo05] Peter Szor. *The Art of Computer Virus Research and Defense*. Addison-Wesley, 2005.

[TG98] Ashis Tarafdar and Vijay K. Garg. Happened Before is the Wrong Model for Potential Causality. Technical Report ECE-PDS-1998-006, Parallel and Distributed Systems Laboratory, ECE Dept. University of Texas at Austin, 1998.

[VD95] S. Venkatesan and Brahma Dathan. Testing and Debugging Distributed Programs Using Global Predicates. *IEEE Transactions on Software Engineering*, 21(2):163–177, 1995.

File Type Analysis Using Signal Processing Techniques and Machine Learning vs. `file` Unix Utility for Forensic Analysis

Serguei A. Mokhov and Mourad Debbabi
Concordia Institute for Information Systems Engineering
Faculty of Engineering and Computer Science
Concordia University, Montréal, Québec, Canada
{mokhov,debbabi@ciise.concordia.ca}

Abstract:

The Unix `file` utility determines file types of regular files by examining usually the first 512 bytes of the file that often contain some magic header information or typical header information for binary files or common text file fragments; otherwise, it defers to the OS-dependent `stat()` system call. It combines that heuristics with the common file extensions to give the final result of classification. While `file` is fast and small, and its magic database is "serviceable" by expert users, for it to recognize new file types, perhaps with much finer granularity it requires code and/or magic database updates and a patch release from the core developers to recognize new file types correctly. We propose an alternative `file`-like utility in determining file types with much greater flexibility that can learn new types on the user's side and be integrated into forensic toolkits as a plug-in that relies on the `file`-like utility and uses signal processing techniques to compute the "spectral signatures" of file types. We present the work-in-progress of the design and implementation of such a tool based on MARF's collection of algorithms and the selection of the best combination and the integration of the tool into a forensic toolkit to enhance the tool, called `fileType` with the automatic machine learning capabilities of the new file types. We compare the advantages and disadvantages of our approach with the `file` utility in terms of various metrics and apply the new tool to learn known stego files to attempt to classify potential unknown stego files and compare the results with `stegdetect`.

1 Introduction

We introduce the topics and tools that are central for this research (in their adequacy) along with the motivation for this work and considered techniques. The first version of a functioning proof-of-concept prototype is expected to be released as open-source at the IMF'08 conference in September 2008.

The main motivation for this work is to be able to determine automatically file types and being very flexible in learning new file types, perhaps with stego information and others, automatically, and inclusion of such a tool in Java-based forensic toolkits. For a large part the common Unix `file` utility does a very good job, but is inflexible in self-learning of the new file types requiring the developers or advanced users and the like to construct and contribute "magic" entries to its database. `file` in particular not very informative at its last stage of recognition saying simply "data" when nothing else matches – this is where we are trying to especially improve in our analysis using classical machine learning techniques. We perform our implementation in Java, to ease the integration as a plug-in to exiting toolkits and other Java-based technologies.

1.2 `file`

The `file` utility [DGC+08, DGC+07] uses algorithmic approach and ruleset wired into its code base to determine file types. It looks for magic numbers at certain offsets, commons structures, ASCII string patterns, etc. to determine file types. `file` tests each argument in an attempt to classify it. There are three sets of tests, performed in this order: file system tests, magic number tests, and language tests. By default, the first test that succeeds causes the file type to be printed [DGC+07]. It has a "magic" file database typically in `/usr/share/file/magic` that advanced users can customize to their needs and put into their home directories. The magic file contains patterns and offsets that map to the file type and description. Such entries can be in plain text or compiled versions and are usually contributed by people, such as developers and others in the know of the details of a particular file type. The official database is updated typically with `file` releases.

1.3 Digital Signatures vs. Spectral Signatures

Traditional digital signatures, such as MD5, SHA1, or any RSA-, DSA-, El-Gamel or any other PKI-based digital signatures are not applicable to our task due to their requirement being highly collision-resistant, so they cannot be used in a fuzzy-like imprecise matching approach where one can have a degree of similarity or difference – as a general rule if a single bit in the particular file changes, the entire signature should change, which is not very useful in our context so we put the cryptographic digital signatures to rest. Instead, we embark onto "spectral" signatures of signal-processing techniques, such as linear-predictive coding (LPC) and Fast Fourier Transform (FFT) combined with machine-learning techniques to classify our file types with some degree of likelihood of a given file type.

1.4 MARF

MARF Overview. MARF is an open-source collection of pattern recognition APIs and their implementation for (un)supervised machine learning and classification, including biometric forensic identification, written in Java [MCSN03, The08, Mok08b, Mok08a, Mok08c, Mok07]. As one of its roles, it serves as a testbed to verify common, found in literature, and novel algorithms for the sample loading, preprocessing, feature extraction, training and classification stages. In this role MARF provides researchers with a tool for the practical comparison of the algorithms in a homogeneous environment and allows for the dynamic module selection based on the wide array of configuration options supplied by applications. Within few years MARF accumulated a fair number of implementations for each of the pipeline stages allowing reasonably comprehensive comparative studies of algorithm combinations, studying their behavior and other properties when used for various pattern recognition tasks. MARF, its derivatives, and applications were also used beyond audio processing tasks, as in this work, due to the generality of the design and implementation in [Mok06, MHL07, Mok08d, Mok08f] and several other works.

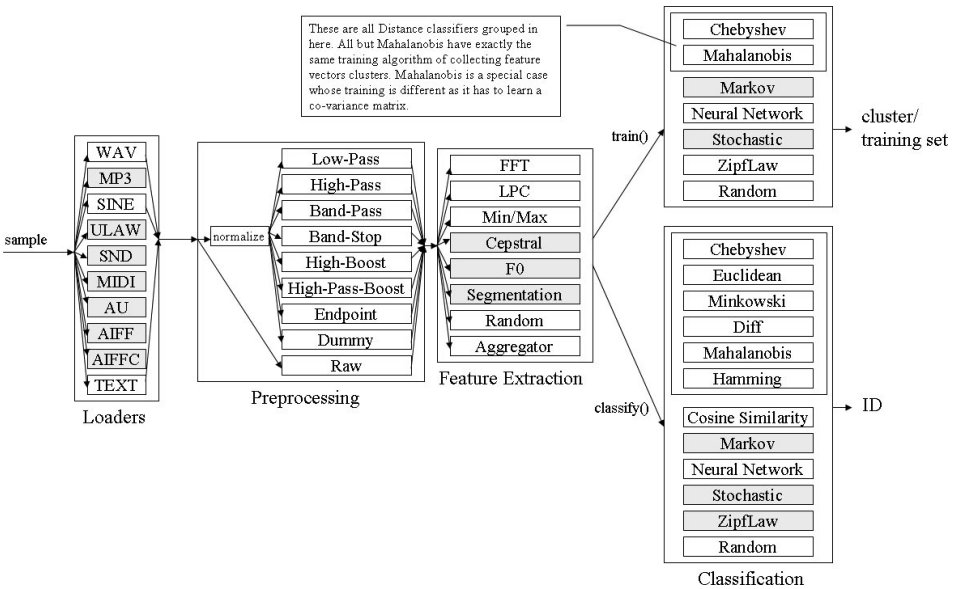


Figure 1: MARF's Pattern Recognition Pipeline

Pattern Recognition Pipeline. The conceptual pattern recognition pipeline design presented in Figure 1 depicts the core of the data flow and transformation between the stages in MARF [The08, MCSN03, Mok07, Mok08b]. The inner boxes represent most of the available concrete module implementations or stubs. The grayed-out boxes are either the stubs or partly implemented. The white boxes signify implemented algorithms. Generally, the classical pattern recognition process starts by loading a sample (e.g. an audio recording,

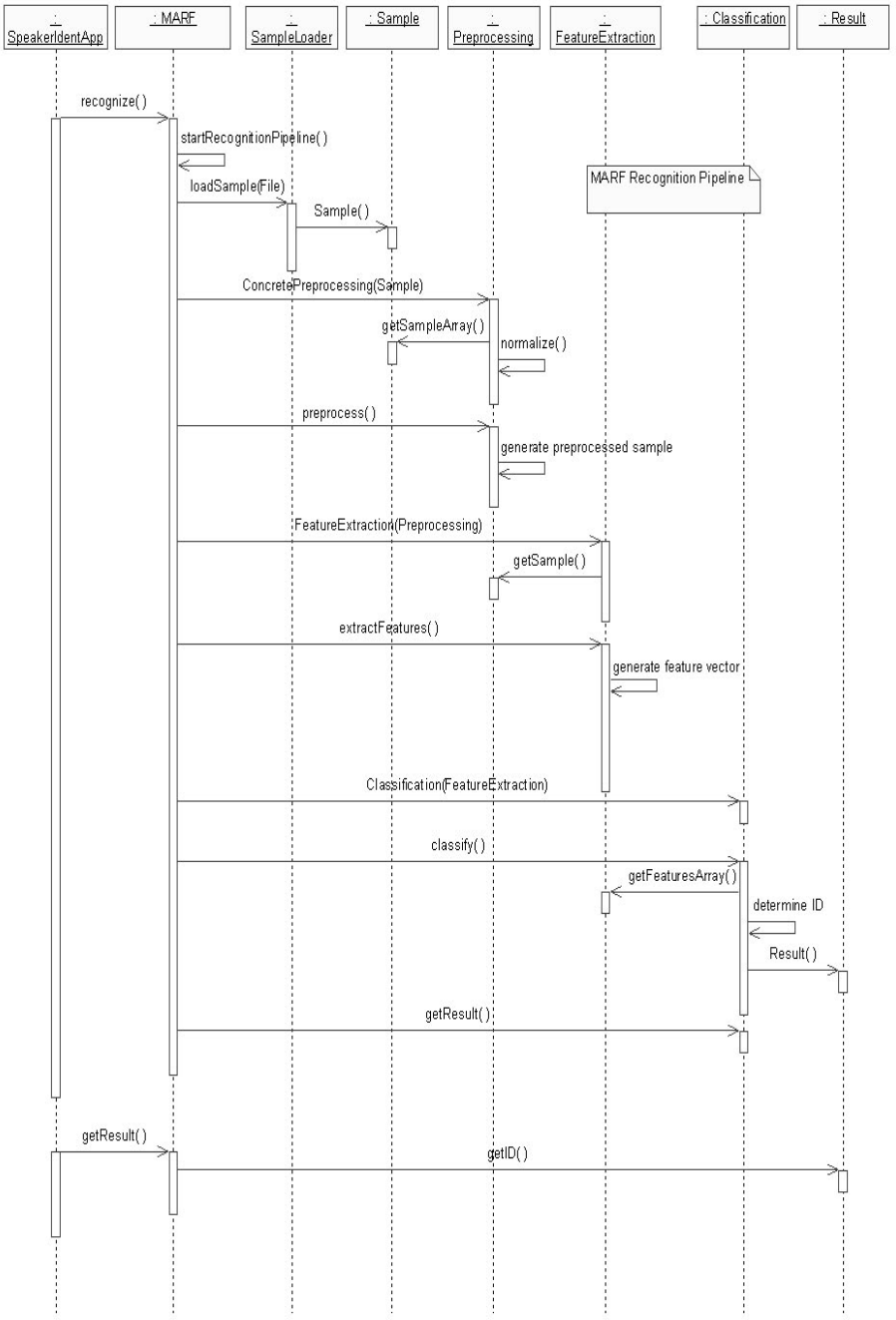


Figure 2: MARF's Pattern Recognition Pipeline Sequence Diagram

text, image file, or virtually any regular file), preprocessing it somehow (e.g. normalization and filtering out noisy and "silent" data), then extracting the most prominent features, and, finally either training the system such that it learns a new set of features of a given subject or actually classifies what/who the subject is. The outcome of training is either a collection of some form of feature vectors or their mean or median clusters [Mok08c], called *training sets*, which are stored per every learned subject. The outcome of classification is an instance of the `ResultSet` data structure, which is a sorted collection of IDs (`int`) and their corresponding outcome values (`double`); the sorting is done from most likely outcome to least likely. The most likely one is the ultimate outcome to be interpreted by the application. Some of the details of such processing of classification are illustrated on the actual sequence of events and method calls within the main `MARF` module is shown in Figure 2. MARF is designed to be a standalone `marf.jar` file required to be usable and has no dependencies on other libraries. Optionally, there is a dependency for debug versions of `marf.jar` when JUnit [GB04] is used for unit testing. MARF has an internal compiler support, which by itself is unrelated to the pattern recognition pipeline but helps with the natural language parsing and forensic biometric tasks [Mok08b, Mok08a, Mok08c]. In our work here we rely on the same pattern recognition pipeline of MARF presented in Figure 2.

1.5 Forensic Toolkits

There are several forensic toolkits in the open-source, academic, and commercial worlds. We project to have our work included into one or more of those either as a plug-in if the host environment is in Java like Forensic Toolkit as JPF Plug-ins [DAS+08, AD07, ADSS07], Ftklipse [LMBT08], or others [htt06], or a standalone tool by itself for inclusion into more general forensic toolsets for binary data analysis extracted in files for preliminary classification file data alongside `stegdetect` [Pro04] and many others. We are considering to make it work with the Linux Sleuth kit and commercial tools, such as FTK, Encase as a part of our future work.

2 Methodology

In this work, we focus on our methodology and techniques to determine file types. We present an approach using the MARF's collection of algorithms to determine file types in various ways and compare them using signal processing and NLP techniques, both supervised and unsupervised machine learning, and various file format loaders. MARF and its application `SpeakerIdentApp` [MSC+08] were shown to be used as a proof-of-concept for biometric forensic analysis of the phone-quality audio recordings to classify the identities of speakers irrespective of what speakers say on voice recordings, their gender, and spoken accent [Mok08b, Mok08a, Mok08c]. We adopt the MARF's pattern recognition pipeline, the `SpeakerIdentApp` application, and the magic database of `file` to be used together, in our new application called `FileTypesIdentApp`, which has a shorthand invocation of `fileType`.

MARF conveniently has a class, `ByteArrayFileReader` that can read a file from a file system or an URI (or any `Reader` or `InpupStream` for that matter). We employ this class to read our file data (either first 512 bytes or entire file as options) and the values of the array become "features" for classification ("spectral" or otherwise). We then may optionally do the regular signal pattern recognition techniques of preprocessing and feature extraction to remove all the unwanted "noise" and "silence" and extract more discriminating features.

The `Raw` preprocessing and `RawFeatureExtraction` modules are used to allow for "no-preprocessing" and feature extraction in a bypass mode (as a no-op) such that the whole testing can be done in a consistent pipelined manner. In the final application however, if this approach is preferred, it can be avoided altogether by going directly to the `Classification` stage if the run-time interactive performance is at issue (doubtly so with the modern hardware).

The `FileTypesIdentApp` application, a.k.a `fileType`, is capable of understanding some of the `file`'s options [DGC+07], and the work is under way to be able to experiment with `file`'s magic database. `fileType`, as its own database that it can learn throughout its lifetime automatically using machine learning techniques. As a more advanced experiment in the works, we train our system on known steganographic files in an attempt to classify the "unseen" stego files automatically. We compare the test runs of `file` and our applications on a set of known files and unknown files, and compile the accuracy statistics. The statistics of the algorithm combinations tried and their recognition accuracy performance along with the run-time are stored in a comma-separated values (CSV) file, per each major technique (see further).

2.1 Related Work

In part our methodology has some similarities in common with the related work on automatic classification of new, unknown malware and malware in general such as viruses, web malware, worms, spyware, and others where AI pattern recognition and expert system techniques are successfully used for automatic classification [RM08, BOA+07, PMM+07, SEZS01, SXC+04, GH99, CXZH07, SML07, CTS+07, HJ07, HRSS07, Sue07].

2.2 Training

We perform two-phase learning in our approach: initial and refining. The initial one is to collect the database of known types in the MARF's understood format using a Perl script, `collect-file-types-meta`, wrapped around the classical `file` utility: we traverse a file system from the root directory, search for regular files (i.e. in the first round we ignore special files, directories, named pipes, block devices, etc.), produce a hash by file type mapping to the the list of file names as "samples" for training and testing. We then take the training set and test it on another, unrelated file system. The filenames

collected only for the purpose of preventing to train on the same file more than once; this is an internal requirement of MARF [The08]. After having built the meta-database in the MARF-compatible format, we train the system regularly, which represents the standard training phase. The refining phase is when we add new file types not found in `file`'s magic database (that are usually classified as "data" by `file`) and we re-train our tool on the new types. The latter part is often manual.

2.2.1 n-grams

The n-gram models come from the natural language processing (NLP) and cryptanalysis of classical ciphers techniques, where languages (natural or programming or binary) can be identified by computing frequencies of n characters occurring one after another. These serve as "features" of a language. This creates a relatively short "dictionary" (a fixed-size matrix) of n-grams where a typical n is usually between 1–3. Since the statistical counts can produce sometimes sparse matrices (the dictionaries), they are smoothed using typical statistical smoothing likelihood estimate techniques. The default processing model is unigram – i.e. a sequence of single bytes are considered to form probabilities and weights for the learned feature vectors. The other two, slower and more demanding in space/time requirements, but more accurate, are bi-gram and tri-gram models that consider sequences of two and three bytes respectively. We use the n-gram models as feature learning and extracting tools and for refinement of the plain-text file types as well as binary sequences for comparison with the other techniques. n-gram modules of MARF add another degree or dimension to our analysis and as a side-effect on MARF required adjustments to be "first-class" citizens within the MARF's pipeline, which they weren't prior to this work, so we augment MARF's code based to make it so.

2.2.2 Meta-Data

At the present moment, the `FileTypeIdentApp` application (similarly to its predecessor applications such as `SpeakerIdentApp` [MSC+08], `LangIdentApp` [Mt08], and `WriterIdentApp` [Mok08f, Mok08e]) manages the training and testing metadata as a CSV file. Each entry in there is a tuple of four fields:

1. ID – a unique integer to represent the file type within the system.
2. Name – human description of the ID (the file type).
3. A set of filenames of training files.
4. A set of filenames of testing files.

The bulk composition of such a file we build via a script by traversing a host operating system in search for regular files, invoking `file` on them, capturing the output as a $Name$, and filling in the training and testing sets as absolute pathnames. The unique ID is later generated for each $Name$ in a sequence using the mentioned Perl script,

`collect-file-types-meta`. We then manually add known stego files (through a "scripted" command-line) for training and testing from various projects, such as honeynet [Hon03a, Hon02, Hon03b]. We also manually review some file types that were knowingly misclassified by `file`.

2.2.3 `strings`

As an interesting feature of reduction of data volume to test, we run the `strings` [Var06] utility to extract anything with string patterns from binary files prior running the MARF training and testing sequences – to determine if this has any performance and accuracy impact on the `fileType`'s learning and classification using the previously mentioned signal processing and NLP techniques. Something similar is being done in the mentioned automated malware detection work, except for us we are being more general [RM08, Sue07], to cover all file types, perhaps containing virii or stego information with them as well.

2.2.4 Magic Database

Another ongoing effort is being made is to determine if we can use `file`'s magic database patterns for training as a shorthand to speed up the training process and what type of accuracy we can hope to achieve for. This part of work is however in the very preliminary stages, and is not expected to work in the first release of `fileType`.

2.3 Testing

Baseline testing is performed on the the same file system we were trained on to make sure we do not deviate from the known types; in itself however, it is considered "cheating". To further validate the trained-on system we move on to a different system and run the classification there via a script using `fileType`'s and `file`'s output for comparison. For example, we move on from one Linux distribution to another doing such a file analysis (filtering out the files that do not exist on either system). Then we test on Windows native and Cygwin [Var08], an then the Mac OS X. We test multiple combinations of the algorithms of MARF and compile their statistics to select the better algorithm combination for the task. Then, we do misclassification testing where `file` was wrong and where we were wrong to compare the accuracy of the file analysis.

3 Limitations and Drawbacks

In the current implementation there are several drawbacks and limitations that we are to eliminate or reduce as a part of the future work. These are listed in part as follows:

- The presented here technique of machine learning and the tool are more effective to

use with the regular files only, and are not suited for special files and devices that are file-system specific. For those, we rely on the same approach as `file` does using the `stat()` system call [Var94], but this is no longer machine learning, so we effectively defer such tasks to `file` that does it better than us.

- "Poisoning" of the trained data with incorrect training information can deteriorate the recognition accuracy of the tool. The "poisoning" can be either accidental (local) or malicious (system-wide) by supplying a file of one type for training, but telling it is another. This is a general problem with any machine-learning tools and applications. A way of dealing with this partly is to validate each incoming training samples by classifying them first and comparing with the class specified for training and in the case of mismatch, report to the user of a potential problem; in the safe mode refuse to train in the case of mismatch. This will prevent the accidental training on the wrong data for misclassification. The latter only partly solves the problem, as the system can be cheated at the beginning when the new file type being inserted for the first time and is mistrained on.
- To be seriously considered in a real investigation toolset and environment, legally, the tool has to be proved to be correct in its design and implementation as well as components it relies on, such as MARF. We plan on using JML [Lea07, LC06, ea05] and Isabelle [PN07] for this task later in the project.

4 Conclusion

The utility presented here, `fileType` is to be released as open-source, just as `file` and MARF are. It will be made available for download and released at the conference with the full demonstration of the results and statistics for Linux, Windows, and MacOS X files and possibly others. We believe spectral signatures and machine learning techniques are very beneficial for file type analysis, especially when there is a need to bulk-preprocess a large collection of files for preliminary classification of "files of interest" on suspect's hard drive, etc. with the easier plug-in-like integration of the tool into Java-based plug-in frameworks, such as JPF, Eclipse, and others.

5 Future Work

The future work will focus on resolving of the known and unknown limitations and drawbacks as well as the few items below:

- Integration with Ftklipse [LMBT08].
- Integration with JPF [DAS+08, AD07, ADSS07].
- Export of Forensic Lucid expressions [MPD08, MP08] for case analysis.
- Integration with JDSF [MHLR07, MHLR08, Mok08d].

6 Acknowledgments

This research work was funded in part by the Faculty of Engineering and Computer Science of Concordia University, Montreal, Canada. We also acknowledge the reviewers who took time to constructively review this work.

References

[AD07] Ali Reza Arasteh and Mourad Debbabi. Forensic Memory Analysis: From Stack and Code to Execution History. *Digital Investigation Journal*, 4(1):114–125, September 2007.

[ADSS07] Ali Reza Arasteh, Mourad Debbabi, Assaad Sakha, and Mohamed Saleh. Analyzing Multiple Logs for Forensic Evidence. *Digital Investigation Journal*, 4(1):82–91, September 2007.

[BOA+07] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario. Automated classification and analysis of Internet malware. Technical report, University of Michigan, April 2007. `http://www.eecs.umich.edu/techreports/cse/2007/CSE-TR-530-07.pdf`.

[CTS+07] I. Cohen, Q. Tian, X. Sean, Z. Thomas, and S. Huang. Feature selection using principal feature analysis. In *Proceedings of the 15th International Conference on Multimedia*, pages 301–304, Augsburg, Germany, 2007. ACM Press.

[CXZH07] I. Cohen, Q. T. Xiang, S. Zhou, and T. S. Huang. Feature selection using principal feature analysis. `http://www.ifp.uiuc.edu/?qitian/epaper/icip02/icip02.pdf`, June 2007.

[DAS+08] Mourad Debbabi, Ali Reza Arasteh, Assaad Sakha, Mohamed Saleh, and Ann Fry. A Collection of JPF Forensic Plug-ins. Computer Security Laboratory, Concordia Institute for Information Systems Engineering, 2007–2008.

[DGC+07] Ian F. Darwin, John Gilmore, Geoff Collyer, Rob McMahon, Guy Harris, Christos Zoulas, Chris Lowth, Eric Fischer, and Various Contributors. `file` – determine file type, BSD General Commands Manual, file(1). BSD, January 1973–2007. man file(1).

[DGC+08] Ian F. Darwin, John Gilmore, Geoff Collyer, Rob McMahon, Guy Harris, Christos Zoulas, Chris Lowth, Eric Fischer, and Various Contributors. `file` – determine file type. astron.com, March 1973–2008. `ftp://ftp.astron.com/pub/file/`, last viewed April 2008.

[ea05] L. Burdy et al. An overview of JML tools and applications. *International Journal on Software Tools for Technology Transfer*, 7(3):212–232, 2005.

[GB04] Erich Gamma and Kent Beck. *JUnit*. Object Mentor, Inc., 2001-2004. `http://junit.org/`.

[GH99] S. Goodman and A. Hunter. Feature extraction algorithms for pattern classification. In *Proceedings of Ninth International Conference on Artificial Neural Networks*, volume 2, pages 738–742, 1999.

[HJ07] K. Hwang and D. Jung. Anti-malware expert system. In H. Martin, editor, *Proceedings of the 17th Virus Bulletin International Conference*, pages 9–17, Vienna, Austria: The Pentagon, Abingdon, OX143YP, England, September 2007.

[Hon02] Honeynet.org. Honeynet Scan 24, 2002. `http://honeynet.org/scans/scan24`.

[Hon03a] Honeynet.org. Honeynet Forensics Project Scans, 2002-2003. `http://honeynet.org/scans`.

[Hon03b] Honeynet.org. Honeynet Scan 26, 2003. `http://honeynet.org/scans/scan26`.

[HRSS07] N. Hnatiw, T. Robinson, C. Sheehan, and N. Suan. Pimp my PE: Parsing malicious and malformed executables. In H. Martin, editor, *Proceedings of the 17th Virus Bulletin International Conference*, pages 9–17, Vienna, Austria: The Pentagon, Abingdon, OX143YP, England, September 2007.

[htt06] http://www.dmares.com. *Software Links for Forensics Investigative Tasks*. 2006. `http://www.dmares.com/maresware/SITES/tasks.htm`.

[LC06] G. T. Leavens and Yoonsik Cheon. Design by Contract with JML. Technical report, Formal Systems Laboratory (FSL) at UIUC, 2006.

[Lea07] G. T. Leavens. The Java Modeling Language (JML). [online], 2007. `http://www.jmlspecs.org/`.

[LMBT08] Marc-André Laverdière, Serguei A. Mokhov, Djamel Bendredjem, and Suhasini Tsapa. Ftkplipse – Forensic Toolkits Eclipse Plug-ins. SourceForge.net, 2005-2008. `http://ciisesec.svn.sourceforge.net/viewvc/ciisesec/forensics`, last viewed April 2008.

[MCSN03] Serguei Mokhov, Ian Clement, Stephen Sinclair, and Dimitrios Nicolacopoulos. Modular Audio Recognition Framework. Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, 2002–2003. Project Report, `http://marf.sf.net`, last viewed April 2008.

[MHL07] Serguei A. Mokhov, Lee Wei Huynh, and Jian Li. Managing Distributed MARF with SNMP. Concordia Institute for Information Systems Engineering, Concordia University, Montreal, Canada, April 2007. Project Report. Hosted at `http://marf.sf.net`, last viewed April 2008.

[MHLR07] Serguei A. Mokhov, Lee Wei Huynh, Jian Li, and Farid Rassai. A Java Data Security Framework (JDSF) for MARF and HSQLDB. Concordia Institute for Information Systems Engineering, Concordia University, Montreal, Canada, April 2007. Project Report. Hosted at `http://marf.sf.net`, last viewed April 2008.

[MHLR08] Serguei A. Mokhov, Lee Wei Huynh, Jian Li, and Farid Rassai. A Privacy Framework within the Java Data Security Framework (JDSF): Design Refinement, Implementation, and Statistics. In Nagib Callaos, William Lesso, C. Dale Zinn, Jorge Baralt, Jaouad Boukachour, Christopher White, Thilidzi Marwala, and Fulufhelo V. Nelwamondo, editors, *Proceedings of the 12th World Multi-Conference on Systemics, Cybernetics and Informatics (WM-SCI'08)*, volume V, pages 131–136, Orlando, Florida, USA, June 2008. IIIS.

[Mok06] Serguei Mokhov. On Design and Implementation of Distributed Modular Audio Recognition Framework: Requirements and Specification Design Document. Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, August 2006. Project Report. A copy is found: `http://marf.sf.net`, last viewed April 2008.

[Mok07] Serguei A. Mokhov. Introducing MARF: a Modular Audio Recognition Framework and its Applications for Scientific and Software Engineering Research. In *Proceedings of the IEEE Engineering/Computing and Systems Research E-Conference (SCSS07/CISSE 2007)*, University of Bridgeport, U.S.A., December 2007. Springer. To appear, `http://cisse2007.org`.

[Mok08a] Serguei A. Mokhov. Choosing Best Algorithm Combinations for Speech Processing Tasks in Machine Learning Using MARF. In Sabine Bergler, editor, *Proceedings of the 21st Canadian AI'08*, pages 216–221, Windsor, Ontario, Canada, May 2008. Springer-Verlag, Berlin Heidelberg. LNAI 5032.

[Mok08b] Serguei A. Mokhov. Experimental Results and Statistics in the Implementation of the Modular Audio Recognition Framework's API for Text-Independent Speaker Identification. In C. Dale Zinn, Hsing-Wei Chu, Michael Savoie, Jose Ferrer, and Ante Munitic, editors, *Proceedings of the 6th International Conference on Computing, Communications and Control Technologies (CCCT'08)*, volume II, pages 267–272, Orlando, Florida, USA, June 2008. IIIS.

[Mok08c] Serguei A. Mokhov. Study of Best Algorithm Combinations for Speech Processing Tasks in Machine Learning Using Median vs. Mean Clusters in MARF. In Bipin C. Desai, editor, *Proceedings of C3S2E'08*, pages 29–43, Montreal, Quebec, Canada, May 2008. ACM and BytePress. ISBN 978-1-60558-101-9.

[Mok08d] Serguei A. Mokhov. Towards Security Hardening of Scientific Distributed Demand-Driven and Pipelined Computing Systems. In *Proceedings of the 7th International Symposium on Parallel and Distributed Computing (ISPDC'08)*, Krakow, Poland, July 2008. IEEE Computer Society Press. To appear, `http://ispdc2008.ipipan.waw.pl/`.

[Mok08e] Serguei A. Mokhov. Writer Identification Application. Unpublished, 2008.

[Mok08f] Serguei A. Mokhov. Writer Identification Using Inexpensive Signal Processing Techniques: Experimental Results. Unpublished, 2008.

[MP08] Serguei A. Mokhov and Joey Paquet. Formally Specifying and Proving Operational Aspects of Forensic Lucid in Isabelle. Technical report, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, August 2008. Accepted at TPHOLs2008 Emerging Trends Track.

[MPD08] Serguei A. Mokhov, Joey Paquet, and Mourad Debbabi. Formally Specifying Operational Semantics and Language Constructs of Forensic Lucid. In *Proceedings of IMF'08*, Mannheim, Germany, September 2008. To appear.

[MSC+08] Serguei A. Mokhov, Stephen Sinclair, Ian Clement, Dimitrios Nicolacopoulos, and the MARF Research & Development Group. Text-Independent Speaker Identification Application. Published electronically within the MARF project, `http://marf.sf.net`, 2002-2008. Last viewed April 2008.

[Mt08] Serguei A. Mokhov and the MARF Research & Development Group. Language Identification Application. Published electronically within the MARF project, `http://marf.sf.net`, 2003-2008. Last viewed April 2008.

[PMM+07] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadugu. The ghost in the browser analysis of web-based malware. `http://www.usenix.org/events/hotbots07/tech/fullpapers/provos/provos.pdf`, 2007.

[PN07] Lawrence C. Paulson and Tobias Nipkow. Isabelle: A Generic Proof Assistant. University of Cambridge and Technical University of Munich, 2007. `http://isabelle.in.tum.de/`, last viewed: December 2007.

[Pro04] Niels Provos. Steganography Detection with Stegdetect, 2004. `http://www.outguess.org/detection.php`.

[RM08] A. Newaz M. E. Rafiq and Yida Mao. A Novel Approach for Automatic Adjudification of New Malware. In Nagib Callaos, William Lesso, C. Dale Zinn, Jorge Baralt, Jaouad Boukachour, Christopher White, Thilidzi Marwala, and Fulufhelo V. Nelwamondo, editors, *Proceedings of the 12th World Multi-Conference on Systemics, Cybernetics and Informatics (WM-SCI'08)*, volume V, pages 137–142, Orlando, Florida, USA, June 2008. IIIS.

[SEZS01] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo. Data mining methods for detection of new malicious executables. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 38–49, Oakland, 2001.

[SML07] V. Sharma, J. G. Muers, and S. Lewis. Continual feature selection: a cost effective method to enhancing the capabilities. In H. Martin, editor, *Proceedings of the 17th Virus Bulletin International Conference*, pages 9–17, Vienna, Austria: The Pentagon, Abingdon, OX143YP, England, September 2007.

[Sue07] M. Suenaga. Virus linguistics – searching for ethnic words. In H. Martin, editor, *Proceedings of the 17th Virus Bulletin International Conference*, pages 9–17, Vienna, Austria: ThePentagon, Abingdon, OX143YP, England, September 2007.

[SXC+04] A. H. Sung, J. Xu, P. Chavez, , and S. Mukkamala. Static analyzer of vicious executables (SAVE). In *Proceedings of 20th Annual of Computer Security Applications Conference*, pages 326–334, December 2004.

[The08] The MARF Research and Development Group. The Modular Audio Recognition Framework and its Applications. SourceForge.net, 2002-2008. `http://marf.sf.net`, last viewed April 2008.

[Var94] Various Contributors. `fstat`, `fstat64`, `lstat`, `lstat64`, `stat`, `stat64` – get file status, BSD System Calls Manual, stat(2). BSD, April 1994. man stat(2).

[Var06] Various Contributors. `strings` – find the printable strings in a object, or other binary, file. [electronic], 2006. man strings(1).

[Var08] Various Contributors. Cygwin = Cygnus + Win32 – a Linux Emulation Layer in Windows. cygwin.com, 2008.

Attacking Test and Online Forensics in IPv6 Networks

LIU Wu, DUAN Hai-xin, LIN Tao, LI Xing, WU Jian-ping

Network Research Center of Tsinghua University
Beijing, P.R. China
liuwu@cernet.edu.cn

Abstract: Although IPv6 protocol has considered and implemented more security mechanisms compared with IPv4, there are still many security threatens in IPv6 Networks. Being one of the key protocols in IPv6, the Internet Control Message Protocol (ICMPv6) suffers from severe security risks. In this paper we construct an IPv6 attacking test system ATS_ICMP_6 exploiting the ICMPv6 Unreachable Message, which shows that the security of IPv6 protocol is still very weak. In the other hand, we has designed and implemented a network forensics prototype 6Foren in IPv6 environment based on the protocol analysis technology, its functions include packet capture, data reconstruct and messages replay etc. the 6Foren can be used as the online digital forensics which support the online forensic of HTTP, FTP, SMTP and POP3 protocols.

1 Introduction

As the rapid development of Internet applications, the Internet Protocol version 4 (IPv4) [10] addresses will soon be exhausted and there are more and more applications running in the next generation Internet based on the Internet Protocol version 6 (IPv6) which boosts the deployment of IPv6 networks [5].

IPv6 protocol (RFC2460) [1] inherits all the merits from IPv4 and adds some new characters, which will greatly improve transmission efficiency. For example, IPv6 changes some fields in the IPv4 main header and extension headers. In IPv4, all nodes can fragment the packets. In the mean while IPv6 also enhanced security in IPv6.

But, as the wide deployment of Internet Protocol version 6 (IPv6) [1] networks, its vulnerabilities have become visible. IPv6 deployment raises security issues for both those not yet managing IPv6 networks as well as those who are. Network attackers have successfully used IPv6 to evade the defenses erected against undesired network traffic [2][3][4].

Recently, [11] awareness has been raised about several threats against the TCP [6] protocol in IPv6. These attacks are based on sending forged TCP segments to any of the TCP endpoints, requiring the attacker to be able to guess the four-tuple that identifies the connection to be attacked.

While these attacks were known by the research community, they were considered to be unfeasible. However, increases in bandwidth availability, and the use of larger TCP windows [12] have made these attacks feasible. Several general solutions have been proposed to either eliminate or minimize the impact of these attacks [13][14][15]. For protecting BGP sessions, specifically, a counter-measure had already been documented in [16], which defines a new TCP option that allows a sending TCP to include a MD5 [17] signature in each transmitted segment.

All these counter-measures address attacks that require an attacker to send spoofed TCP segments to the attacked host. However, there is still a possibility for performing a number of attacks against the TCP protocol, by means of ICMPv6 [7].

With the Internet Control Message Protocol (ICMPv6) [8], one could extrapolate the concept of "hard errors" [8] to ICMPv6 Type 1 (Destination Unreachable) codes 1 (communication with destination administratively prohibited) and 4 (port unreachable). Thus, any of these messages could elicit a connection abort.

For example, ICMPv6 defines the "Packet Too Big" (type 2, code 0) error message, that is analogous to the ICMP "fragmentation needed and DF bit set" (type 3, code 4) error message in IPv4. For IPv6 networks, intermediate systems do not fragment IP packets. Thus, there's an implicit "don't fragment" bit set in every IPv6 datagram sent on a network. Therefore, hosts do not treat ICMPv6 "Packet Too Big" messages as hard errors, but use them to discover the MTU of the corresponding internet path, as part of the Path MTU Discovery mechanism for IPv6 [9].

The Host Requirements RFC [3] states that a TCP instance should be notified of ICMPv6 error messages received for its corresponding connection.

In order to allow ICMPv6 messages to be demultiplexed by the receiving host, part of the original packet that elicited the message is included in the payload of the ICMPv6 error message. Thus, the receiving host can use that information to match the ICMPv6 error to the instance of the transport protocol that elicited it.

Neither the Host Requirements RFC nor the original TCP specification [6] recommends any security checks on the received ICMPv6 messages. Thus, as long as the ICMPv6 payload contains the correct four-tuple that identifies the communication instance, an attacker could send a spoofed ICMPv6 message to the attacked host, and, as long as he is able to guess the four-tuple that identifies the communication instance to be attacked, he can use ICMPv6 to perform a variety of attacks, such as DoS, DDoS, Man-In-The-Middle, Smurf, Redirect attack etc.

This paper aims to raise awareness of the use of ICMPv6 to perform a number of attacks against IPv6 networks, to show that IPv6 protocol is also not secure, to pay attention to the security problems of IPv6.

The field of computer forensics has become a critical part of legal systems throughout the world used for detecting attacks in IP networks. As early as 2002 the FBI stated that “fifty percent of the cases the FBI now opens involve a computer. However, the accuracy of the methods -- and therefore the extent to which forensic data should be admissible -- is not yet well understood. Therefore, we are not yet able to make the kinds of claims about computer forensics that can be made about other kinds of forensic evidence that has been studied more completely.

Computer forensics can be divided into two types: host based forensics and network based forensics [18]. The host based forensics is also called Software Forensics, which is mainly used to trace code to its authors [25]. Some computer scientists focus largely on the examination of file system data [20], whereas others also include the collection of data [19] [21][22][23][24] [26].

While in the other hand, there is very few forensic tools for network based forensics, especially for IPv6 networks.

As IPv6 protocol (RFC2460) header structure has changed greatly with that of IPv4 protocol, current IPv4 forensic tools could not meet the IPv6 forensic needs. So we need to design and implement forensic tools for IPv6 network security management.

This paper is organized as follows. In section 2, we describe the basic idea of ICMPv6 attack. The details on implementation of ICMPv6 attacks are described in section 3. Section 4 describes the 6Foren: Online Forensics in IPv6 Network. In section 5, we show two cases of ICMPv6 attacks and display the 6Foren system. Finally we present our conclusions in section 6.

2 IPv6 Attacking Test and Online Forensics Environment

To verify and test our IPv6 Attacking tools and Online Forensic system 6Foren, it needs a pure IPv6 network environment.

As seen in Fig. 1, the left side is the topology of the pure IPv6 attacking test bed A6TB which is an open attacking test bed. In A6TB we deploy some IPv6 attacking host and some IPv6 victim host which is used for the IPv6 attacking test. Both the attacking hosts and victim hosts are all pure IPv6 machines connected via CERNET2 which is also pure IPv6 Backbone network connecting all the Chinese educational organizations to USA, Asia and Europe IPv6 networks.

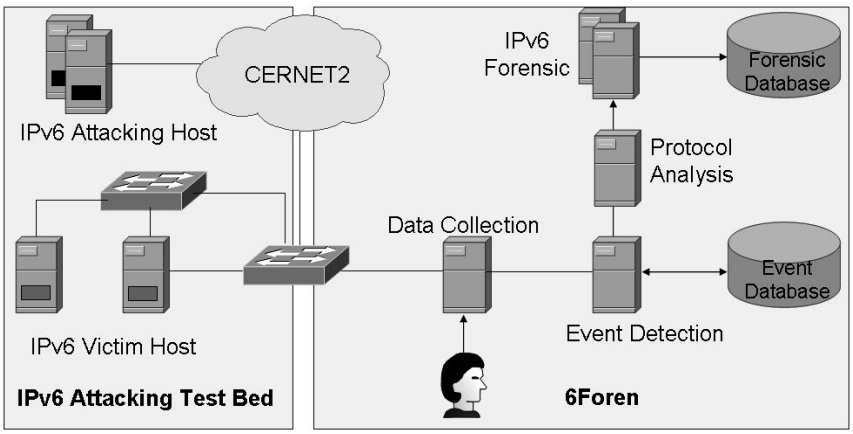


Fig. 1 The Topology of IPv6 Attacking Test Bed and 6Foren

As seen in the right of Fig.1, 6Foren is deployed near the key switch of the test bed A6TB to capture the attacking flow and then executing online forensic actions.

As seen in Fig. 2, in normal communication, when node V (Victim) want to visit node T (Target) which is out the V's LAN, packets from V to T is first delivered to the default router R1 (Router-1) in the LAN, and R1 will route V's packets to other routers and finally to T. in this case, packets from V to T are transmitted along the solid line shown in Fig. 2.

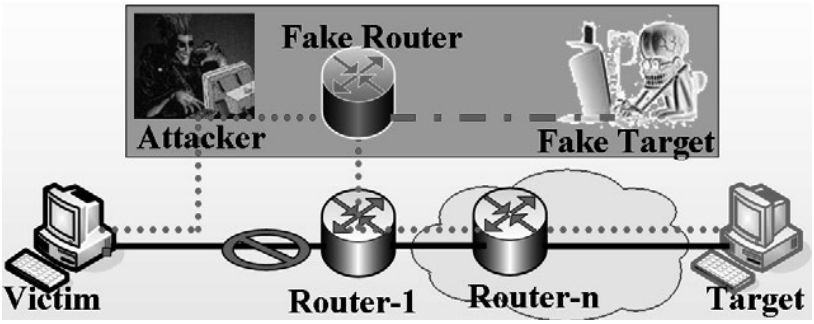


Fig.2 Basic Idea and Attacking Test Environment of ICMPv6 Attack

If A (Attacker) want to attack traffic from V to T, he will first detect the activity of V by sending an ICMPv6 Echo quest packet with bad IPv6 header parameter to T, if V is alive, A will then send an ICMPv6 Redirect Message to V which will redirect all V's packet from R1 to FR (Fake Router) which is illegally constructed by A. Finally, A can execute any attacking action to traffic from V.

In the following section, we will describe the attacking process in details.

3 IPv6 Attacking Test and Online Forensics Environment

In this section, we briefly summarize key techniques for IPv6 attacking test using ICMPv6 messages we combined for this research.

As seen in Fig. 3, the IPv6 Attacking Test System using ICMPv6 Messages (which is called ATS_ICMP_6) is mainly distributed in two logical hosts (Attacker and Fake Router), and consists of the following 6 functional modules:

Initialization: this module is mainly used to load some initial information used for ATS_ICMP_6, which include:

(1) Interface: network card used for the attacking system ATS_ICMP_6

(2) IPv6 Addresses of current legal default router, which includes both of the global IPv6 address and the link local IPv6 address

(3) MAC Address of current legal default router

(4) IPv6 Addresses of Attacker himself, which includes both of the global IPv6 address and the link local IPv6 address

(5) MAC Address of Attacker himself

(6) IPv6 Addresses of Fake Router, which includes both of the global IPv6 address and the link local IPv6 address

(7) MAC Address of Fake Router

Alive Detection: this module is used to detect whether the specific victim is now active in the network. In addition, it can also return the list of active hosts in the LAN.

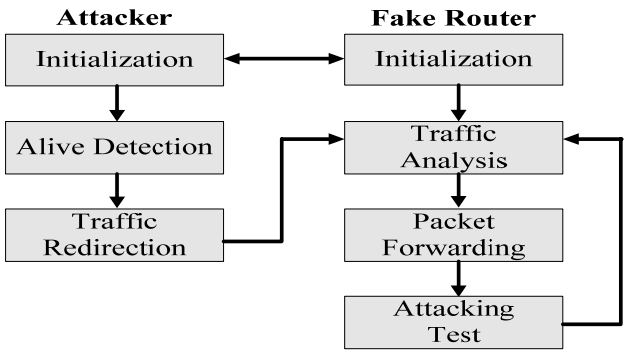


Fig.3 Structure and Functional Module of ATS_ICMP_6

Traffic Redirection: this module is used to change traffic from legal default router to the Fake Router further traffic analysis and attacking test. This module takes the key role in ATS_ICMP_6.

Traffic Analysis: When Fake Router starts working, this module is listening all traffic redirected from victims, used to change traffic from legal default router to the Fake Router further traffic analysis and attacking test. This module takes the key role in ATS_ICMP_6.

Packet Forwarding: This module is used to modify, drop or relay specific traffic according to the requirement of specific Attacking Test module.

Attacking Test: Based on the modules described above, this module is used to test some specific type of IPv6 attacking. Before test, this module will send the attacking policy such as attacking Type, Code and other parameters which is used for IPv6 attacking test.

Note: we must state that, in most cases the logical hosts Attacker and Fake Router can be deployed in one physical machine.

In the following sub sections, we will introduce some of the key modules used in this system.

3.1 Alive Host Detection

Before attacking, the attacker must first detect the activity of the victim host in the network.

To do this, the attacker can send an ICMPv6 message with bad parameter to the victim, when receiving this bad message, the victim will send back a Parameter Problem message to the attacker, according to which, the attacker knows that the victim is now "alive". Fig. 4 shows the Alive Host Detection algorithm.

Process Alive_Host_Detection

INPUT: V -- IPv6 Address of Victim, if V=NULL, then V="FF02::1", the multicast address

OUTPUT: The alive host list AHL: if V=NULL, AHL include all the active host in the LAN

BEGIN

Construct an ICMPv6 Ping Request message MS (Type=128, Code=0) with bad parameter

Send MS to V

Start the Packet Filter Process and set the filter="ip6 and dst V" to capture packets return from V with Parameter Problem Messages

If return TRUE, then V is alive

Else V is not alive

END

Fig.4 The algorithm of Alive Host Detection

3.2 Start the Packet Forwarding Function

Before attacking, especially the traffic redirection attacking, the attacker must start the packet forwarding function in the fake router to forward the redirected traffic to where the attacker wants it to be.

```
int Start_Packet_Forwarding(char* interface, char * policy)
{
        sprintf(cmd,"echo 1 > /proc/sys/net/ipv6/conf/%s/forwarding",interface);
        system(cmd);
        system("echo 1 > /proc/sys/net/ipv6/conf/default/forwarding");
        system("echo 1 > /proc/sys/net/ipv6/conf/all/forwarding");
        Forwarding_traffic(policy);
        return 0;
}
```

Fig.5 Packet Forwarding Function

Fig. 5 shows the packet forwarding function which can relay the traffic from victims to the “right” place after being redirected from default router to the fake router.

The first four lines in Start_Packet_Forwarding is used to open the packet forwarding function in the kernel of the Fake Router to forwarding all traffic flowing across it.

The function Forwarding_traffic will relay, modify or drop specific traffic according to the policies defined by the parameter “policy” which is transmitted by specific attacking test module.

3.3 Traffic Redirection

After the detection of alive host, now the attacker can use ICMPV6 Route Redirect Message to redirect the V’s traffic to FR which is the faked router constructed by the attack for further attacking actions.

The algorithm is shown in Fig. 6.

To redirection of the victim, the attacker must first construct some type of ICMPv6 Messages:

(1) ICMPv6 Neighbor Advertisement Message NAM (Type=136, Code=0)

(2) ICMPv6 Router Advertisement Message RAM (Type=134, Code=0)

(3) ICMPv6 Router Redirection Message RRM (Type=137, Code=0)

Second, these messages must be sent to the LAN in a proper period to notify the victim that the default router has been changed to the fake router.

To ensure the successful attack, the broadcast period is key factor.

Note: if V=NULL, the fake default route information will be planted in every hosts in the LAN, which means that all the Traffic in the LAN will be first redirected to the fake router.

```
Process Redirect_Traffic_Static_IPv6
INPUT: V -- IPv6 Address of Victim, if V=NULL, then V="FF02::1", the multicast address
OUTPUT: The alive host list AHL: if V=NULL, AHL include all the active host in the LAN
BEGIN
  Initial: routerip =get_default_router();// get current default router's IPv6 address
   victimmac =get_mac(V);// get V's MAC address
  Start_Packet_Forwarding();//start the packet forwarding function
  Construct an ICMPv6 Neighbor Advertisement Message NAM (Type=136, Code=0)
  Construct an ICMPv6 Router Advertisement Message RAM (Type=134, Code=0)
  Construct an ICMPv6 Router Redirection Message RRM (Type=137, Code=0)
  while (1) {
        Send NAM to V
        Send RAM to V
        Send RAM to V
        Sleep(period)
  }
END
```

Fig.6 Redirect Traffic for Host with Static IPv6 Address

We can see from Fig. 7 that, seconds after starting Redirect_Traffic_Static_IPv6, one more router information:

"2001:da8:200:9002:21d:9ff:fe1c:9bff dev eth0 lladdr 00:1d:09:1c:9b:ff router REACHABLE"

has been planted in the victim's router table, which shows that now the default router is "2001:da8:200:9002:21d:9ff:fe1c:9bff" that is the fake router's IPv6 address constructed by the attacker.

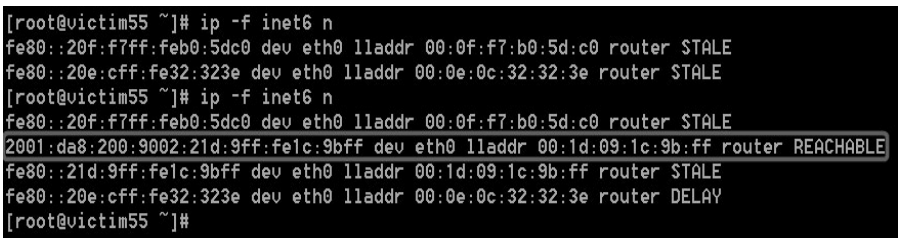


Fig. 7 Route Information Before Traffic Redirection v.s Route Information After Traffic Redirection in Victim

3.4 Attacking Test

Actually, this module includes two parts: the server side and client side.

The server side is deployed in the fake router, its main function is receiving and process attacking parameters and policy, then sends them to the Traffic Analysis module.

The client side is deployed in another machine, it is mainly used for attacker(s) to:

(1) fill in the attacking command and relative parameters

(2) control the attacking process

(3) adjust attacking parameters while attacking

(4) stop the attacking process

(5) display the attacking results

4 6Foren: Online Forensics in IPv6 Network Environment

Fig. 8 shows the basic structure and main modules for IPv6 network based forensics which we called it 6Foren. 6Foren mainly includes the following modules: Data Collector, TCP Stream Classification, TCP Message Reconstruction, Application Protocol Parser Manager and Online Evidence Display. We will describe these modules in detail.

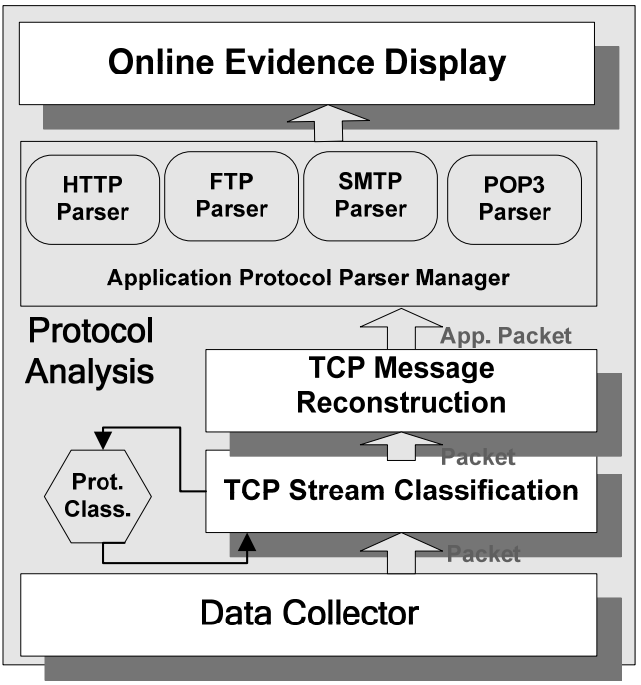


Fig.8 Structure and Main Module of 6Foren

4.1 Data Collector

To obtain accurate online forensic analysis result, we must first capture proper packets in which the attacking flow is transmitted.

In our 6Foren, we will take two different data collecting mechanisms to ensure the accurate data collection.

In normal condition, if we can control the entire network flow which means that the attacking flow can be "seen" by us, we can just mirror the network flow containing the attacking packets to a specific interface where we can easily collect the data.

But sometimes, we are not the administrator of the network and do not have the right get the network flow. In this case, we can redirect the network flow first to our data collector using the Traffic Redirection module described in § 3.3

4.2 TCP Stream Classification

When packet is captured, it must first be classified according to its TCP attribution, and the system needs to maintain a table to save the classification information of the TCP stream.

The TCP Stream Classification module must consider the following problems:

(1) The TCP stream table must save application layer information for this connection for further handling.

(2) The TCP Stream Classification should accurately recognize the begin and end of the TCP connection

(3) During the system running, the TCP stream table should be kept smaller

4.3 TCP Message Reconstruction

The main functions of the TCP Message Reconstruction module include:

(1) Reconstruct the IP packet with error arrival order

(2) Reproduce the session process of the communication sides

4.4 Application Protocol Parser Manager

To distinguish classify different kinds of application protocols, the Protocol Classification module of 6Foren inspects all captured packets and judge that whether it belongs to some protocol when transmitting. By adding and deleting elements of the TCP stream table, 6Foren dynamically obtain data in some TCP connection and then classify and reconstruct TCP packets to entirely save all the application protocol data. 6Foren will also parse file name, file type and contents etc. according to different protocol signatures. Fig. 6 shows the Flow Chart of Protocol Classification module in 6Foren.

(1) HTTP Protocol

In HTTP protocol, the client start to transmit session through the GET request packet in TCP connection, in the meanwhile the contents of the file will be sent to the client from the server side. 6Foren judge the start of the HTTP file transmission mainly by the GET signature in the traffic header (seen in Fig. 6), and then add the reverse TCP stream to the monitored TCP classification list, if the reverse TCP data has the “HTTP/*.* 200” header, then the TCP Packet Reconstruction module will save all the reverse TCP data.

(2) FTP Protocol

FTP protocol has two modes (seen in Fig. 6): PORT and PASV, which possess the signature PORT and PASV respectively in the starting of the FTP stream. As the FTP protocol negotiate data connection IP address and port number in the session, we must continuously listening the controlling connection for some times to get sufficient information, and also the FTP data can either be upload or download, so the implementation of FTP Forensic is very difficult.

In PORT mode, the data connection address and port can be directly gotten because they exist in the PORT command itself, while the RETR or STOR command exist in the next session of the same direction.

While in PASV mode, the data connection address and port exist in the 227 response packet, and RETR or STOR command is executed by client, so it needs multiple listening.

(3) SMTP and POP3 Protocol

POP3 packets and SMTP packets are similar. The POP3 packets begin with "Received", while the SMTP packets begin with "DATA" (in Outlook Express and Outlook) or "Data" (in Foxmail). The text and attachment of email will be transmitted in the same direction of the same TCP connection, so it can be easily captured.

4.5 Online Evidence Display

While the attacking event is detected, it will be recorded in the Attack Event Database for attack event replay in the law court.

In the mean while, the detected attacking event can be displayed in a new webpage for the online forensic.

5 Experimental Results

5.1 Start the IPv6 Attacking System ATS_ICMP_6

The IPv6 Attacking System ATS_ICMP_6 is implemented in Linux system with kernel version over 2.4. To simplify the operation of administrator and other tester, the server side has been registered as a Linux service, which will be started automatically when the operating system starting. When ATS_ICMP_6 starts, it will listen on port 8899 (which can be specified as other port).

The client side is implemented via web page. All the attacking test cases can be operated in browser anytime and anywhere.

In the following sub section, we will take two attacking cases as examples.

5.2 Case 1 ICMPv6 Error Message for DoS attack

As seen in Fig. 9, to start the Redirection Attack, the attacker execute the following steps:

(1) fills in the following items:

a) Redirection Source: the source that the packets will be illegally redirected

b) Redirection Target: the target that the packets will illegally redirected

c) Older Router: the current legal default router, which in normal case will forwarding traffic to legal target

d) New Router: the fake router which will execute the redirection attack

(2) click the “Start Attack” button

In this case, packets from victim54.vhost.edu.cn to www.cernet2.edu.cn via the default router router6.vhost.edu.cn will be first redirected to the fake router fakerouter6.vhost.edu.cn and then be leading to a fake target which is seen in Fig. 2.

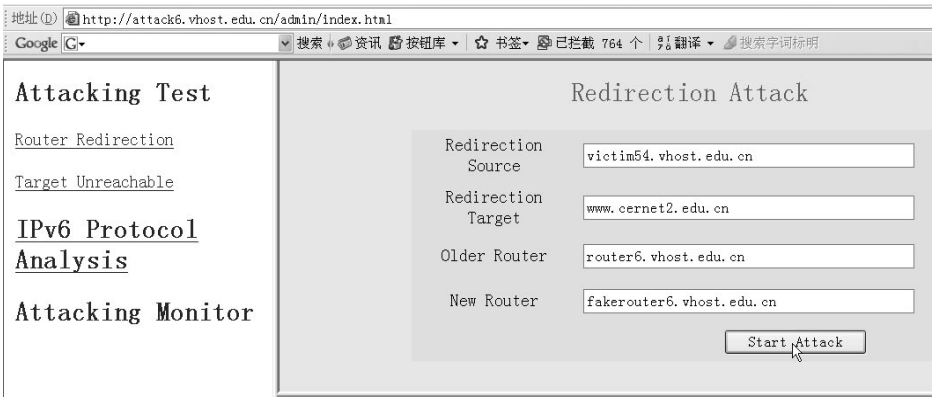


Fig. 9 The Redirection Attack

5.3 Case 2 ICMPv6 Destination Unreachable Attack

As seen in Fig. 10, to start the Destination Unreachable Attack, the attacker execute the following steps:

(1) fills in the following items:

a) Victim: the IPv6 address or DNS name of the attacked host

b) Target: the IPv6 address or DNS name that the victim will visit

c) Unreachable Type: an integer between 0 and 4 (seen in table 1) which will be used for the attack experiment

d) Unreachable Target Port: an integer which represents the target's port that the victim will connected to in the attack experiment

(2) click the "Start Attack" button

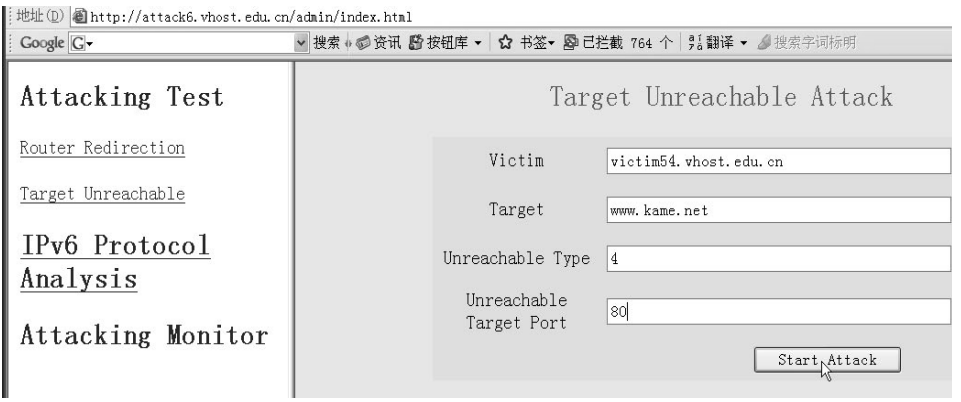


Fig. 10 The Target Unreachable Attack

In this case, before the attack, a user can visit any port of any host from victim54.vhost.edu.cn. But, after the attack, he will not visit the web service (port 80) in www.kame.net, but he can visit any other ports of this host and can visit any other hosts from victim54.vhost.edu.cn

In addition, either Victim or Target can be NULL if the Victim is NULL, it will attack all the hosts in the LAN, and if the Target is NULL, the Victim can not visit any other machine in the Internet.

5.4 Online Forensic for HTTP Protocol

As an example seen in Fig. 11, to take the online digital evidence by 6Foren, it should pass through the following 3 steps:

(3) We first send and IPv6 Route Deceive Attack by executing the following steps via HTTP protocol:

a) Redirection Source: the source that the packets will be illegally redirected

b) Redirection Target: the target that the packets will illegally redirected

c) Older Router: the current legal default router, which in normal case will forwarding traffic to legal target

d) New Router: the fake router which will execute the redirection attack

After clicking the "Start Attack" button, packets from victim54.vhost.edu.cn to www.cernet2.edu.cn via the default router router6.vhost.edu.cn will be first redirected to the fake router fakerouter6.vhost.edu.cn and then be leading to a fake target

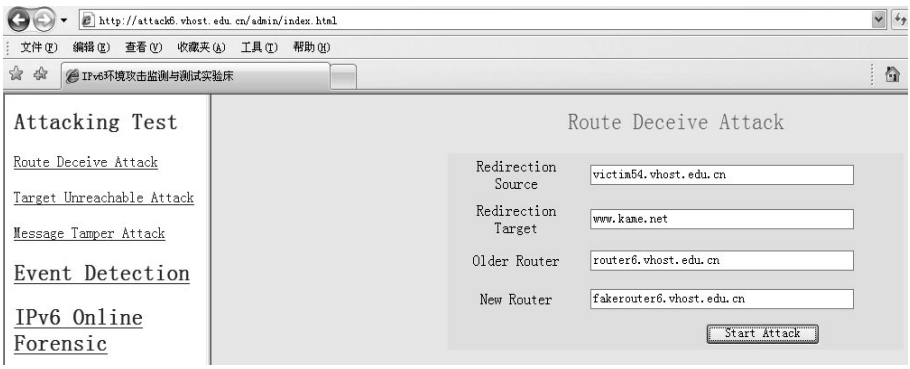


Fig. 11 Launch a Route Deceive Attack

(4) Attack Detection

Before the attacking is launched, the Event Detection daemon has been started to detect any attacks. As seen in Fig. 12, if click the "Event Detection" in the 6Foren system, it will link to the Event Detection webpage which displays the attacking event detected by 6Foren.

Fig.12 Route Deceive Attack Detection in 6Foren

(5) Online Evidence Display

When the the Event Detection module detect the attacking event, it will automatically open the Online Evidence Display webpage and open the original webpage that the attacker opened to launch the attacking, which can be seen in Fig. 13.

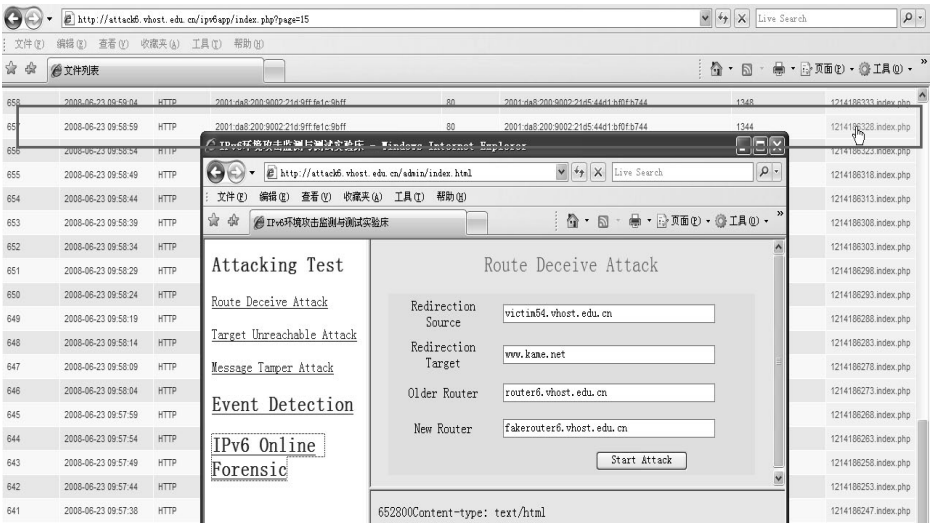


Fig.13 6Foren for HTTP protocol

As seen in Fig. 10, all the attacking events are logged into the IPv6 Forensic Database for Event Replay in courtroom, on the other hand (seen in the bottom webpage of Fig. 5), while the attack event is going on, the webpage will be displayed synchronized for Online Evidence Display (seen in the front webpage of Fig. 5).

5.5 Online Forensic for FTP Protocol

(1) Sending message with attacking signature via FTP protocol

As shown in Fig. 14, we send a PDF file named 6Foren_ftp.pdf with some attacking signature in the Attack Database

Fig. 14 FTP attacking

(2) Detect and Online Display the Attacking Evidence in Webpage

While the transmission of the attacking file, the Event Detection module successfully detected this malicious file and start the Online Forensic module to record and display the this attacking event, which can be seen in Fig. 15

Fig.15 6Foren for FTP protocol

5.6 Online Forensic for SMTP and POP3 Protocol

As seen in Fig. 16, we sent some malicious email via IPv6 SMTP email system Mail6 we specially developed for this project and then receive emails with Mail6.

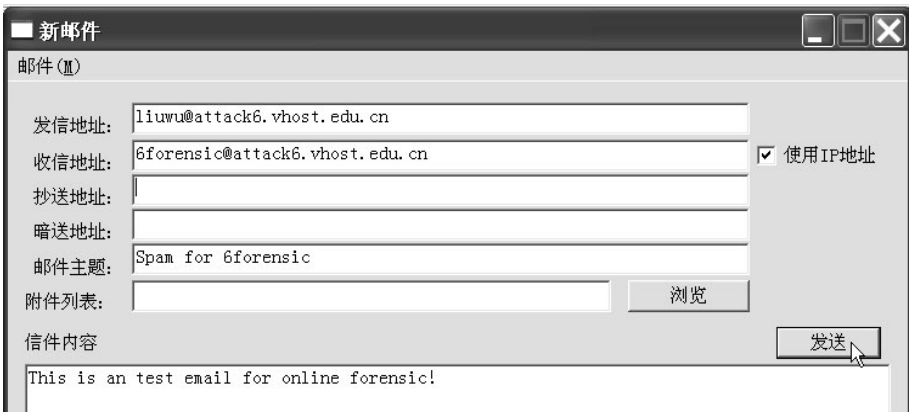


Fig. 16 Sending Malicious Email via IPv6 SMTP Email System

Seen from Fig. 17, while the attacker is sending and/or receiving emails, 6Foren will detect corresponding actions and start the Online Forensic module to record and display the original email that is being sent and received.

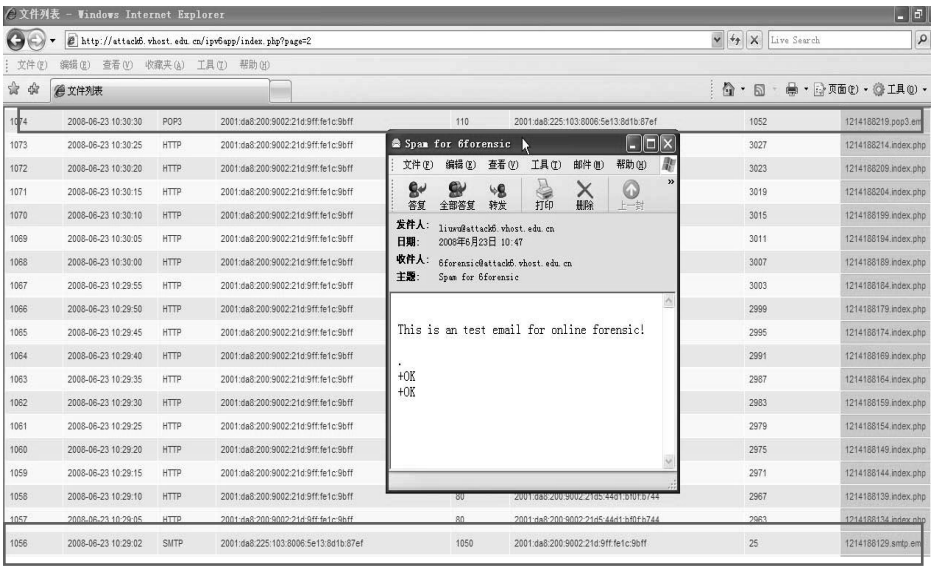


Fig.17 6Foren for SMTP and POP3 protocol

6 Conclusion

In this paper, we designed and implemented IPv6 attacking test system ATS_ICMP_6, which can execute IPv6 attacking experiments via ICMPv6 Messages. In the other hand, we designed and implemented a network forensics prototype 6Foren in IPv6 environment based on the protocol analysis technology, its functions include packet capture, data reconstruct and messages replay etc. the 6Foren can be used as the online digital forensics which support the online forensic of HTTP, FTP, SMTP and POP3 protocols.

Bibliography

[1] S. Deering and R. Hinden, Internet Protocol, Version 6 (IPv6) Specification, RFC2460, Internet Engineering Task Force, December 1998.

[2] T. Narten, E. Nordmark and W. Simpson, Neighbor Discovery for IP Version 6 (IPv6), RFC2641, IETF, December 1998.

[3] S. Thomson and T. Narten, IPv6 Stateless Address Autoconfiguration, RFC2462, Internet Engineering Task Force, December 1998.

[4] T. Narten, E. Nordmark and W. Simpson, Neighbor Discovery for IP Version 6 (IPv6), RFC2641, IETF, December 1998.

[5] Wu Liu, Study on Intrusion Detection Technology with Traceback and Isolation of Attacking Sources, PhD Thesis, 2004.

[6] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[7] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, September 1981.

[8] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[9] McCann, J., Deering, S. and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, August 1996.

[10] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[11] Watson, P., "Slipping in the Window: TCP Reset Attacks", 2004 CanSecWest Conference , 2004.

[12] Jacobson, V., Braden, B. and D. Borman, "TCP Extensions for High Performance", RFC 1323, May 1992.

[13] Stewart, R., "Transmission Control Protocol security considerations", draft-ietf-tcpm-tcpsecure-02 (work in progress), November 2004.

[14] Touch, J., "ANONsec: Anonymous IPsec to Defend Against Spoofing Attacks", draft-touch-anonsec-00 (work in progress), May 2004.

[15] Poon, K., "Use of TCP timestamp option to defend against blind spoofing attack", draft-poon-tcp-tstamp-mod-01 (work in progress), October 2004.

[16] Heffernan, A., "Protection of BGP Sessions via the TCP MD5 Signature Option", RFC 2385, August 1998.

[17] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[18] Wu Liu, Hai-xin Duan, and Jian-ping Wu, An Authorship Analysis Model MBSFAM in Software Forensics, J. of Computer Research and Application, 2005 Vol.11(5):121-128

[19] F. Buchholz. Pervasive Binding of Labels to System Processes. PhD thesis, Purdue University, 2005.

[20] B. Carrier. File System Forensic Analysis. Addison Wesley Professional, 2005.

[21] B. D. Carrier. A Hypothesis-Based Approach to Digital Forensic Investigations. PhD thesis, Purdue University, 2006.

[22] D. Farmer and W. Venema. Forensic Discovery. Addison Wesley Professional, 2004.

[23] B. A. Kuperman. A Categorization of Computer Security Monitoring Systems and the Impact on the Design of Audit Sources. PhD thesis, Purdue University, 2004.

[24] U. Lindqvist and P. A. Porras. eXpert-BSM: A Host-Based Intrusion Detection System for Sun Solaris. In Proceedings of the 17th Annual Computer Secrity Applications Conference (ACSAC), pages 240{251.IEEE Computer Society, December 10{14 2001.

[25] E. H. Spaord and S. A. Weeber. Software forensics: Can we track code to its authors? Technical Report CSD-TR 92-010, Department of Computer Science, Purdue University, 1992.

[26] T. Stallard and K. Levitt. Automated Analysis for Digital Forensic Science: Semantic Integrity Checking. In Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC), December 8-12 2003.

Live Forensic Acquisition as Alternative to Traditional Forensic Processes

Marthie Lessing[1], Basie von Solms[2]

[1]Council for Scientific and Industrial Research
Meiring Naudé Road, Scientia
Pretoria, South Africa
marthie.lessing@gmail.com

[2]Academy for Information Technology
University of Johannesburg
Auckland Park Kingsway Campus
Johannesburg, South Africa
basievs@uj.ac.za

Abstract: The development of live forensic acquisition in general presents a remedy for some of the problems introduced by traditional forensic acquisition. However, this live forensic acquisition introduces a variety of additional problems, unique to this discipline. This paper presents current research with regards to the forensic soundness of evidence retrieved through live forensic acquisition. The research is based on work done for a PhD Computer Science at the University of Johannesburg.

1 Introduction

Both computer and non-computer professionals use computers every day. Accordingly, it is safe to say that computers play a significant role in both business and personal life, and that the resultant technological advances are remarkable.

One of the bigger Information Technology advances was the creation of the Internet to connect people globally. Unfortunately, the Internet not only aided worldwide communication and commerce, but also sparked the growth of electronic crime. Criminals now make use of computers on a daily basis to assist with and to commit crimes.

To act against these electronic offenders, it is necessary to develop new processes and techniques to retrieve evidence from computers. Specialists commonly refer to this discipline as Cyber Forensics [BJ05].

1.1 Defining Cyber Forensics

According to Jones [Jo07], Cyber Forensics is *"… the process of copying data from a computer in a forensic manner"*. Another definition is *"… the discipline that combines elements of law and computer science to collect and analyse data from computer systems, networks, wireless communications and storage devices in a way that is admissible as evidence in a court of law"* [Us05].

There are a number of definitions available, varying with regards to the extent of the forensic process. However, it is generally accepted that forensic acquisition includes the transportation of the data from the crime scene to a safe location and safe storage, but excludes the interpretation of the acquired data.

The Federal Bureau of Investigation started to formally employ Cyber Forensics in 1984 as part of their criminal profiling initiative [Fe07]. In this regard, Cyber Forensics classifies as a retrospective profiling type, which is generally case specific and after the fact [NTV05]. Forensic investigators only enter the crime scene after the crime has been committed, and therefore need to follow a specific set of steps to avoid contamination or corruption of the evidence. Should the investigator contaminate the crime scene, it is likely that the evidence will not be usable in court.

Heated discussions exist in the world of Cyber Forensics. The two most prominent arguments are regarding pulling the plug (dead digital forensics), or exercising the analysis on a live, running system (live digital forensics). Whichever of the modes are applied, the basic Cyber Forensic methodology consists of three important principles:

- acquire the evidence without altering or damaging the original;
- authenticate that the recovered evidence is the same as the originally seized data; and
- analyse the data without modifying it [KH02].

The current forensic best practice is to unplug a machine to acquire an image of the hard drive. This technique can cause data corruption, system downtime and consequential revenue loss for businesses. Section 3 discusses this dead forensic acquisition in more detail. A newer technique, live forensic acquisition, emerged to counteract some of the problems caused by dead forensic acquisition. This technique refers to the acquisition of a forensically sound system image from a live machine, i.e. a machine that is still running. The system image can range from a mirror copy of a single USB drive, to a computer hard drive or a comprehensive image of the system network. Section 4 introduces live forensic acquisition in more detail.

1.2 Research Problem

The development of new criminal techniques leaves law enforcement techniques outdated. As a result, crime scene investigators cannot always apply dead digital forensics successfully to gather sufficient evidence to lead to a conviction. In response, cyber specialists developed live forensic acquisition techniques to specifically target these new criminal techniques. These new live forensic techniques, however, still need to be tested thoroughly before the law enforcement discipline will adopt them.

Unfortunately, despite the numerous benefits posed by live forensic acquisition, it brings about technical and legal difficulties as well: forensic investigators need to convince the court that their forensically acquired image is a true unaltered copy of the original. (This is necessary regardless whether the acquisition method is dead or live, but it is more complicated to prove with live forensic acquisition.) At present, there is no commonly accepted method of doing live acquisition and additional research is necessary. However, preliminary research presents a very viable attitude towards live forensic acquisition as alternative to traditional forensic acquisition.

2 Forensic Acquisition Process

Technically the forensic acquisition process should only incorporate the evidence collection stage. However, to ensure the forensic soundness of the acquired evidence, a comprehensive acquisition needs to start with the collection of evidence, and end with the successful transport and storage of the evidence. The next paragraphs will introduce the steps that form the forensic acquisition process. This applies to both dead digital forensics and live forensic acquisition.

2.1 Access the Acquired Device

Kruse and Heiser [KH02] list three methods to access the acquired device. Depending on the acquisition mode that the investigator chooses to follow, access to the acquired device might be different.

- The first method is to pull the power plug from the back of the computer (dead forensic acquisition - Section 3).
- The second method is to follow the normal administrative shut down procedure (dead forensic acquisition - Section 3).
- The third method is to keep the system running (live forensic acquisition - Section 4).

Regardless of which acquisition method is used, the very first step in any forensic investigation should be the isolation of both the system and the relevant data. The purpose of this isolation is two-fold: isolation can prevent the corruption of other systems, reducing the risk of a cascading failure throughout the organisation's IT infrastructure, and isolation freezes the state of the affected system, preserving an exact image to assist in the subsequent investigation [WP05].

Investigators should collect information by interviewing system administrators and other users who might have had contact with the affected system [WP05]. In the event of a covert investigation, interviews might not always be possible. However, investigators should still try to gather as much information about the system before starting the acquisition process. Occasionally it might be possible to retrieve passwords beforehand, saving valuable time and effort on the investigator's side.

Once the system is completely isolated, the investigator should collect all possible non-technical information, such as the suspect's office attendance in the days preceding the incident. This information assists in establishing a potential time line of events leading to the suspected cyber crime [WP05].

To develop this timeline further, the investigator needs to check the BIOS (Basic Input Output System). The BIOS provides a variety of critical information, such as the time and date. It is also possible to identify the HPA (Hardware Protected Areas) and the DCO (Device Configuration Overlays) on the computer by investigating the BIOS. One method is to compare the hard drive settings stored in the CMOS with the values on the drive's labels. Alternatively, the investigator can do a similar comparison with a series of ATA commands (READ_NATIVE_MAX_ADDRESS and IDENTIFY_DEVICE). Some forensic applications, such as TSK and Maresware, also allow for the detection of a HPA presence. The HPA and the DCO are reserved areas for data storage outside the normal operating system file system. Since these areas are normally used for specialised application data and configuration data, forensic investigators do not necessarily search these areas for additional hidden data. Accordingly, knowledgeable cyber criminals can store incriminating data in both the HPA and the DCO [Be05]. Should the two sets of compared values differ, the investigator knows that there exist a HPA and can make a more specialised effort of locating these files.

Once the investigator identified the existence of the HPA and DCO, s/he can make a full bitstream copy of the system to copy these hidden areas [St08]. When this process is complete, the forensic investigator can initiate the forensic acquisition with a write blocker to prevent accidental writing to the protected hard drive.

2.2 Initiate the Acquisition with a Write Blocker

During forensic acquisition and analysis, it is possible to write to the evidence drive accidentally. Since this will lead to the immediate dismissal of the evidence from court, the investigator should take care not to compromise the evidence. The easiest way to ensure this is to use a write blocker.

A write blocker allows a system to read data from an external drive at full speed. At the same time, it blocks any write commands to the external drive to prevent the unauthorised modification or formatting of the drive under examination [Pa07]. Normally a computer writes data to, or reads data from a storage device via specific commands, transmitting these commands from the computer's interface connection to the storage device's interface connection. By using a write blocker, the investigator prevents the computer from writing to the evidence hard drive's interface [Ni03a].

There are two types of write blockers: software write blockers and hardware write blockers. A *software write blocker* replaces a hard drive access interface on a computer with external hard drives. It blocks any commands that could modify a hard drive [Ni03b]. A *hardware write blocker* is a hardware device that physically attaches to a computer system. Its main purpose is to intercept and block any modifying commands from reaching the storage device [Ni03a].

It is of high importance to preserve the evidence and create copies of the evidence for analysis purposes [WP05]. After the physical acquisition, coupled with the write blocker, it is necessary to document the incident and all the actions taken by the investigators. The chain of custody, discussed in the next section, reflects this.

2.3 Chain of Custody

In any investigation, the acquired data and devices should be accounted for during the entire extent of the forensic acquisition process. Technically, this chain of custody should commence the moment the forensic investigator enters the crime scene, and continue until the court case completes. Although this step does not only belong to the acquisition phase, it forms a fundamental aspect of the case's validity and the forensic soundness of the evidence.

According to Ghelani [Gh06], chain of custody defines as *"… the gathering and preservation of the identity and the integrity of the evidential proof that is required to prosecute the suspect in court"*. Scalet [Sc05] provides another definition: *"… A chain of custody is the process of validating how any kind of evidence has been gathered, tracked and protected on its way to a court of law"*. In essence, it is the maintenance of the integrity of the evidence from seizure until the time the investigator produces it in court [Tr94]. It serves to make it difficult for a defence attorney to argue that the forensic investigator tampered with the evidence whilst in his/her custody [KH02].

The ability to prosecute any case rests on the validity of the evidence used in court. A court considers evidence valid if forensic investigators can prove that the evidence is in the same condition as during the original seizure. To do this, people who handled the evidence should testify as to the condition the evidence was in before and after it entered their possession. The proper use of a chain of custody can replace this tedious process of testifying [Tr94]. Complete and accurate chain of custody logging procedures help to ensure that the court will authenticate electronic data. It is therefore crucial to ensure that the chain of custody adheres to the prescribed standards [Le08].

2.4 Transport and Storage of Evidence

To complete the Cyber Forensic acquisition process, the evidence needs to be transported from the crime scene to the forensic laboratory. At the laboratory, the evidence will be stored securely, unless it is being analysed by the forensic investigators.

Magnetic media are in general very sensitive, and not originally created with transport in mind. Accordingly, it is necessary to take certain precautions to ensure that the evidence arrives without any damage. For example, both during transportation and storage, the evidence should be stored in static-free packaging. Additionally, to ensure that nobody tampers with the evidence during transportation or storage, the last investigator to handle the evidence at the crime scene should seal the package and sign the seal. If anybody attempts to open the package, the seal will be broken and the signature spoiled. Every time somebody needs to access the evidence, the old package should be put into a new package, and the new package be sealed and signed [KH02].

Some court cases are inevitably postponed several times, and can range a number of years. Accordingly, it is necessary to control *bit rot*. According to Church [Ch07], bit rot can be defined as *"… the degradation of magnetic media over time"*. It can be hugely problematic if your evidence has deteriorated beyond use when your court date comes up. It is therefore crucial to protect your evidence as best as possible, and use the original data as little as possible. This will not stop the deterioration, but at least slow it down [Au08].

3 Traditional Forensic Acquisition

The first of the two Cyber Forensic acquisition modes is dead acquisition analysis, often referred to as the traditional digital method. Dead acquisition involves examiners pulling the plug on a suspect system, avoiding any malicious process from running on the system and potentially deleting data from the system. It allows the examiner access to create a snapshot of the swap files and other system information as it was last running [St08].

A formal definition of dead acquisition analysis is *"… analysis done on a powered off computer"* [Jo07]. Usually there are four stages to traditional digital forensics:

- **Collection** entails the process on location: search and seizure and the acquisition of information in a forensically sound manner. The main actions are the forensic disk duplication and collection of random evidence, such as CDs, scraps of papers and personal interviews.
- **Examination** composes both a manual and an automatic investigation of the acquired data. This stage aims to identify and extract data relevant to the specific case, and includes file system parsing and extraction of mailboxes.
- **Analysis** is the process of using the identified data in a manner to prove that the actions performed on the computer was done by one or more individuals. This stage involves browsing, querying and correlating existing data [Al06].

- **Reporting** is the last stage in which the forensic examiner reports the information gathered in a written form [Jo07].

The forensic copying process is not straightforward, but with sufficient training and the correct forensic software packages, forensic investigators can copy the hard drive image, complete with unallocated sectors, slack space and file metadata. This is generally done by copying the seized hard drive bit by bit [Jo07].

Figure 1 illustrates the forensic process. The forensic investigator first approaches the computer and determines its power status. If the computer is powered on, s/he turns it off by either pulling out the power plug, or following the proper shut down procedure. Once the power is off, the forensic investigator physically removes the hard drive from the system, attaches it as an external drive to a forensic system and copies its content. The investigator takes the necessary precautions to ensure that no data modification takes place on the external drive. Depending on the specific situation, the investigator may either return the hard drive to the original system, or bag it as evidence.

For many years, traditional forensics has been the only means to perform forensic acquisitions. It is a simple procedure to follow, and straightforward steps have been tried and tested to perform these actions. However, this technique presents both advantages and disadvantages, discussed next.

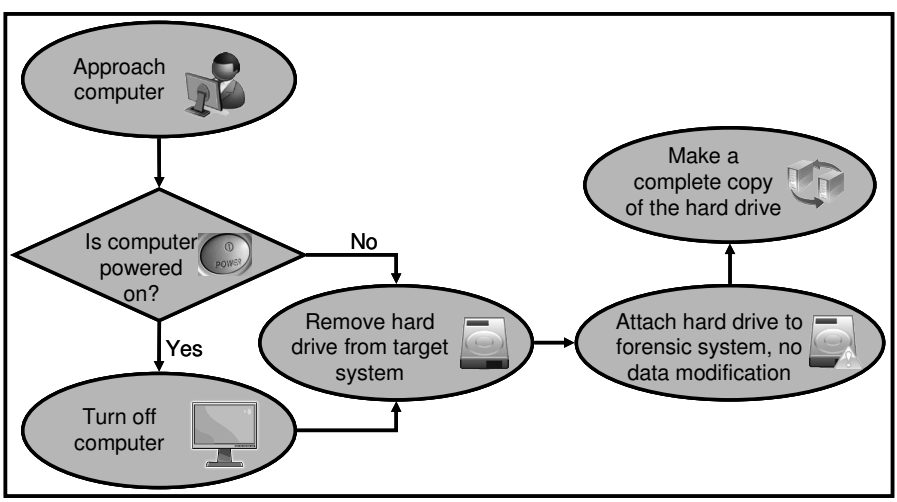


Figure 1: Traditional forensic analysis [Jo07]

3.1 Positive Aspects of Dead Acquisition Analysis

Under normal circumstances, the chance of forensic investigators accidentally overwriting or modifying evidentiary data is slim. Generally, sufficient precautions are in place to ensure that the computer allows *no modification during the copying process* to either the original or the copied image of the original hard disk [Jo07].

A distinguishing characteristic between dead and live forensics is that dead forensics cannot acquire live, volatile data. Once the computer is unplugged, the machine loses all the volatile memory in the RAM. However, a little known fact is that most modern *RAMs retain their contents for several seconds* after power is lost. The system does not immediately erase the volatile memory, but its content becomes less reliable when not refreshed regularly. A forensic investigator that is aware of this can therefore make use of this small window of opportunity to do a forensic acquisition [Ha08].

3.2 Limitations of Dead Acquisition Analysis

There are a number of limitations and problems associated with dead acquisition analysis. Some problems are more serious than other problems, but it is necessary to look at all instances.

- In response to the efficiency of dead acquisition analysis, criminals have resorted to the widespread use of *cryptography*. Now, even though forensic examiners have a complete bit for bit hard drive image of the suspect system, it is encrypted and of no practical value. In this scenario, the drive can only be decrypted with a unique password. Since investigators cannot always rely on a suspect's cooperation in supplying this password, the method of acquisition should be adjusted – if the same encrypted disk was acquired with live forensic acquisition, investigators would be able to access the disk. This whole-disk encryption is not only limited to criminals, but is now also a default feature of some operating systems.
- Another limitation of traditional forensic acquisition has surfaced in the light of *network data*. The need for acquiring network related data (such as currently available ports) grew dramatically. This type of information is volatile, and is lost in the event that the computer powers down - the foundation of traditional digital forensic analysis [Jo07].
- To comply with traditional forensic requirements, *all data must be gathered and examined* for evidence. However, modern computers consist of gigabytes, and even terabytes, of data to be analysed [LK04]. These complex technologies, coupled with cyber crimes becoming more advanced, lead to more complex and time-consuming digital investigations. It is increasingly difficult to use modern tools to locate vital evidence within the massive volumes of data. Log files also tend to increase in size and dimension, complicating a Cyber Forensic investigation even further [Fe07].
- A *lack of standardised procedures* leads to uncertainties about the effectiveness of current investigation techniques. In turn, this has led to the suboptimal use of resources. In some instances, investigators gather worthless

data that take unnecessary time. In addition, this data have to be stored and takes up valuable space [LK04].

- Many unique *practical and legal constraints* make the application of Cyber Forensics both interesting and defiantly complex. An example of a practical constraint would be if the suspect system were a public machine in an internet café with the owner claiming a possible loss of income for the duration of the forensic investigation. An example of a legal constraint is the restriction of the methods in which forensic investigators can obtain data. This is especially relevant when practising live forensic acquisition.
- If forensic investigators do not follow these restrictions exactly, data acquired in certain ways may be *inadmissible in court* and not allowed as intelligence [Jo07]. This negates the criminal investigation completely. For this reason, it is important that forensic practitioners are equipped with tools and mechanisms that can result in the acquisition of forensically sound system images. Only when this is possible, can data be seen as evidence and be admissible in a court of law.
- Already mentioned in Section 3.1, dead forensics is *not the optimal method to acquire live, volatile data*. Although modern RAMs allows a couple of seconds grace period in which the volatile data is not erased, this time is often too little to do a proper acquisition.

Due to the many limitations of traditional forensics, live forensic acquisition seems a likely alternative. This allows forensic practitioners to access a variety of invaluable information that would have been lost in traditional forensic analysis [Jo07]. Unfortunately, the practice of live acquisition brings about its own limitations, especially with regard to legal implications. The next section addresses this acquisition mode.

4 Live Forensic Acquisition

The method of live forensic acquisition is similar to the method of dead forensic acquisition. It developed in response to the shortcomings of the traditional forensic acquisition techniques, considering the retention of volatile data, and a countermeasure for encrypted files on a live system.

The acquisition philosophy is the same in that both methods need to ensure that the acquired image remains unchanged. The sequence of steps also applies to both dead and live acquisition (collection, examination, analysis, reporting). Investigators, however, should tailor the inner workings of the stages to allow for a forensically sound live acquisition [Jo07].

Figure 2 presents the forensic investigator's actions during live acquisition. The investigator first approaches the computer and determines its power status. If the computer is powered off, s/he continues with the dead forensic acquisition procedure discussed in Section 3.

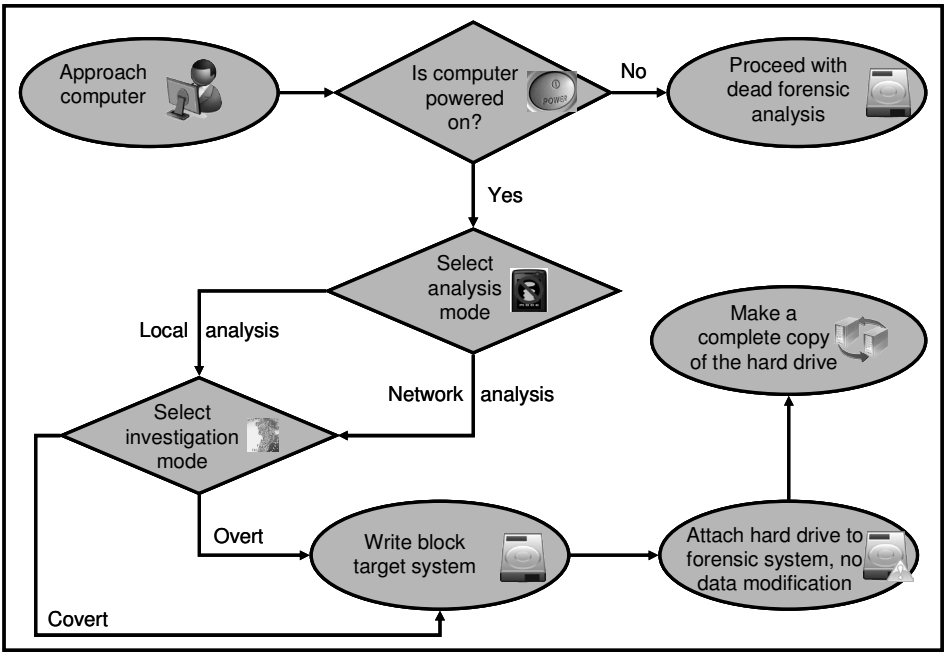


Figure 2: Live forensic analysis (Own compilation)

If the computer is switched on, the investigator first needs to select whether the data will be copied locally, or through the network. Additionally, s/he needs to decide whether the investigation will take place overtly or covertly. The investigator then attaches a write block system (either hardware based or software based) to both the suspect machine and as an external drive to a forensic system to copy the suspect machine's content. Of course, a live forensic investigation is a lot easier if the logged on account has administrative rights. However, should the current account not have administrative rights, the user functionality the investigator can work with depends on the rights assigned to the current account and the software already installed on the suspect system. The vast number of possible scenarios is beyond the scope of this paper.

During a live forensic investigation, it is necessary to determine whether the logged on account lies in a real or virtual environment. In essence, the different environments require the same investigation method. However, if the logged on account links to a virtual machine, the investigator needs to do further analysis to acquire both the real machine's system image, as well as other possible virtual machines located on the real machine. It may be difficult to detect whether the forensic investigator accessed a real computing environment, or a virtual machine. A number of techniques exist that can indicate whether a system is real or virtual. By far the easiest technique is to look for:

- copyright notes or vendor strings in various files;
- VMware specific hardware drivers;
- VMware specific BIOS;

- VMware specific MAC addresses;
- installed VMware tools; and
- hardware virtualisation (e.g. virtual sets of some registers).

However, these traces can be rather unreliable since it is easy to modify. A more reliable trace method is to do hardware fingerprinting, or checking for hardware that is always present in a virtual machine. In the case of Microsoft virtual machines, a clear indicator is if the motherboard is made by "Microsoft Corporation". DEVCON, a command line version of Device Manager, is also useful in detecting what hardware is in a system, identifying any virtual machine hardware. For example, REM Virtual Server and Virtual PC both have their IDE disks named "Virtual HD" [Ar05].

A more reliable technique is to install virtual machine detectors (or fingerprinting tools), but this software may have a negative impact on the forensic soundness of the evidence. Examples of these tools are the Red Pill, Jerry, ScoopyNG and VMware Virtual Machine Detector.

4.1. Positive Aspects of Live Acquisition Analysis

In response to the limitations of dead acquisition analysis, live acquisition analysis has surfaced as a remedy. This analysis allows forensic examiners to *retrieve volatile information* specific to the suspect system's network settings. In many instances, this information is invaluable to the prosecution of a cyber criminal. It is thus possible to view the development of live acquisition analysis as an improvement of current methods of both dead and live acquisition [Ni06].

In contrast with the procedural deficiency of traditional forensic acquisition, live forensic acquisition limits the amount of data gathered. Often investigators investigate large parts of the system, but only gather the relevant pieces of information [LK04]. Live acquisition *addresses this procedural deficiency*, but it introduces a number of other problems.

4.2 Limitations of Live Acquisition Analysis

Although live acquisition addresses most of the problems associated with dead forensic acquisition, it brings about additional problems:

- *Every computer installation is different*. Although there are many common components and aspects, computer users can compile their system to their own desire. For this reason, it is the forensic examiner's job to ensure that s/he has sufficient knowledge of a wide variety of hardware, software and operating systems. It is indeed possible to come across any combination of these components, and the examiner should be prepared to handle all of these. Due to the range of possibilities provided by live forensic analysis, forensic examiners only learn the principles of live acquisition and the effect that specific actions may have on the validity of the evidence. It is further up to the

interpretation of the examiner to analyse the situation, and apply the forensic principles in such a way that his/her actions can be justified in a court of law.

- *Data modification* during the acquisition process and the dependence of the forensic acquisition on the suspect system's operating system is two of the more prominent concerns regarding live forensic acquisition. If the acquisition process alters the data, courts will dismiss the data as forensically unsound. Part of the live acquisition forensic examination process is to execute code running on the CPU of the suspect system. This can potentially change data in the CPU registers, RAM or the hard drive itself. Even if the forensic system specifies no explicit write commands, the suspect system's operating system may decide to swap the program to hard disk. This inherent operating system feature may complicate the incentive for allowing the concerned evidence in a court case and the evidence may be ruled inadmissible. In addition, inappropriate action taken by forensic examiners may ruin evidence. In the event that a forensic examiner handles a situation incorrectly, a preventable amount of data may be changed. For example, running an application on the suspect hard drive may overwrite some of the associated properties, such as recent actions. If the specifics of this application were critical to the case, it will cause many issues in court [Jo07].
- Linked to the problem of data modification are *slurred images*. Similar to when you take a photo of a moving object, slurred images is the result of acquiring a file system while some program modifies it. The smallest modification may cause a problem, since the file system first reads the meta-data section of the hard disk. If the files or folders on the file system change after the file system have read the meta-data, but before the files system acquires the data, the meta-data and sectors do not correlate anymore [Jo07]. Similarly, volatile memory does not represent a single point in time, but rather a time sliding view. When acquiring volatile data, investigators cannot always use write blockers, nor is there always a MD5 comparison to the original data [Vi06].
- Another recurring problem concerning live forensic analysis, especially network evidence from untrusted networks, is *authenticity and reliability*. Anti-forensic toolkits are also widely available, and may obstruct the collection of evidence from live network sources [Ni06]. By applying anti-forensic measures, clued-up criminals may reduce the effectiveness of a potential forensic investigation. It is, for example, possible to write a program that destroys evidence when the operating system detects a forensic acquisition program [Jo07]. These types of programs is developed by individuals or organisations that want to thwart legit forensic investigations, and aims to delete all incriminating evidence on the victim computer and computer system. Some of these programs include Evidence Eliminator, The Defiler's Toolkit, Diskzapper, CryptoMite, Tracks Eraser Pro and Invisible Secrets [Co07]. Another type of anti-forensic software, developed by the Metasploit Project, targets specific functionalities of legitimate forensic investigation tools. These anti-forensic wares interfere with the forensic software's results during an investigation [Hi07]. Anti-forensic tools work on a variety of platforms, and perform a number of different functions.

- In some instances of live forensic acquisition, *limited amounts of information is gathered*. This may not always constitute a complete representation of the original affected system, and can be interpreted as possible data corruption [LK04]. The investigation into this aspect will contribute to a comprehensive model for live acquisition.

Considering the extensive lists of limitations for both traditional digital forensic and live forensics, it is necessary to compare these disciplines with regards to the forensic soundness principle. The next section looks at this.

5 Forensic Soundness

A vast number of technical issues can physically constrict Cyber Forensics, being a rather technical application of computer related knowledge. In addition to these limitations, numerous laws strictly bind forensic practitioners to the letter, the implementation of these sometimes rather disdain [Jo07].

Evidence can either make or break an investigation. Therefore, it is crucial to ensure that all evidence is admissible in court. Should the court reject any item of evidence, it can either hurt the case, or at the very least, portray the investigators as incompetent.

According to Bejtlich [Be06], a forensically sound copy of a hard drive is *"… created by a method that does not, in any way, alter any data on the drive being duplicated. A forensically sound duplicate must contain a copy of every bit, byte and sector of the source drive, including unallocated empty space and slack space, precisely as such data appears on the source drive relative to the other data on the drive. Finally, a forensically-sound duplicate will not contain any data… other than which was copied from the source drive."* Since this traditional definition of forensic soundness does not allow any leeway for live response and acquisitions, Mike Murr redefined forensic soundness by adding *"… The manner used to obtain the evidence must be documented, and should be justified to the extent applicable"* [Mu06].

The most important question in determining the admissibility of evidence concerns the authenticity of the data. If the data is changed in any way, counsel will have a very hard time convincing the court to include it as evidence [Br05]. Accordingly, this data needs to be considered as forensically sound before it can classify as evidence.

The growing number of attorneys and courts that rely on the results of digital examinations ignited a global debate on the exact constitution of sound forensics. All parties involved agree that forensic acquisitions should not alter the original evidence source in any way. However, forensic experts show that the act of preserving certain digital sources in many cases require the alteration of the original evidence item. For example, the common method of performing live forensic acquisition requires that the investigator load the acquisition tool into memory. This overwrites some of the system's volatile data. This is a distinct alteration of the original data evidence source [Ca07].

Another example concerns the use of remote forensic tools. These tools necessitate the investigator to establish a network connection, accordingly altering the original evidence source. Even the use of hardware write blockers when acquiring data from an IDE hard drive may temporarily reconfigure evidentiary data to access the HPA [Ca07].

5.1 Volatile nature of Cyber Forensics

Cyber Forensics as a discipline tends to be highly volatile. This is in part because the discipline relies on technology, which in itself can be very unstable, but also because the discipline largely constitutes incident response in reaction to unpredictable criminal behaviour.

This volatile nature can best be described by the Heisenberg uncertainty principle in quantum physics. The Heisenberg principle states that locating a particle in a small region of space makes the momentum of the particle uncertain. Additionally, measuring the momentum of a particle precisely makes the position uncertain [Wi08a]. This is very similar to the concept of live forensic acquisition. The mere action of collecting evidence figuratively makes the environment unstable. This translates as rendering evidence forensically unsound.

According to Heisenberg, it is possible to measure the position of an atom with a photon. The uncertainty principle states that, when the photon is introduced, it will change the momentum of the atom by an uncertain amount that is inversely proportional to the accuracy of the position measurement. The amount of uncertainty can never be reduced below the limit set by the principle, regardless of the experimental setup [Wi08a].

Similar to the uncertainty principle, is the *observer effect.* This principle refers to changes that the physical act of observing will make on the observed phenomenon. The same example applies to this principle. In order to see an electron, a photon must first interact with it. This interaction will indefinably change the path of the electron [Wi08b].

Although the modification of evidentiary data remains a huge problem, it is not a new concept. Methods in traditional biological forensic disciplines, such as DNA analysis, also alter the original evidence. However, courts still accept this evidence. When a traditional forensic examiner collects samples of biological material, he/she needs to scrape or smear the original evidence. In many instances, DNA tests are highly destructive. Although investigators can extract information from the original evidence, investigators cannot present the original blood sample or skin sample to the court as evidence. Despite the changes that occur during preservation and processing, courts consider these methods as forensically sound. In fact, investigators regularly admit DNA evidence as evidence [Ca07].

Similarly, digital forensic investigators need to acquire data from a suspect system and analyse, examine and alter its presentation to produce meaningful information to the courts. This meaningful information is rarely in the same format as the format in which the investigator acquired the original data (similar to a traditional forensic investigator acquiring a piece of fingernail, and running DNA analysis to present to court).

When considering the use of traditional digital forensic measures, courts should allow the minor alteration of original evidence, similar to the allowed minor alterations of original evidence in traditional biological forensics. However, investigators should still adhere to the basic Cyber Forensic principles and not alter evidence in such a way that the meaning thereof changes. The legal system therefore needs to be updated to accept digital forensic analysis in a court of law, as long as the data still adhere to the definition of forensic soundness presented in Section 5. Investigators should still focus on maintaining the reliability and authenticity of the evidence [Ca07].

It is not practical to set an absolute standard that dictates the preservation of everything and the modification of nothing. This excludes the viability and usability of both Cyber Forensic processes and traditional forensic methods. This would send the entire legal system in disarray [Ca07].

5.2 Ensuring forensically sound acquisition

The key to forensic soundness is documentation, or a proper chain of custody. The acquisition process should change the original evidence as little as possible. Investigators should document any changes whatsoever and assess it in the context of the final analytical results [Ca07].

In most cases, courts will accept the forensic soundness of a piece of evidence if the supporting documentation is sufficient. This documentation should report on the evidence's origin and the way investigators handled it since acquisition. The acquisition process needs to preserve a complete and accurate representation of the original data. Investigators should do this in such a way that courts can validate its authenticity and integrity [Ca07]. However, this open standard allows for exploits by cyber specialists that knows how to manipulate both the digital evidence and the chain of custody.

There is no specific test to determine whether digital evidence possesses the required scientific validity. However, Cyber Forensic evidence proposed for admission in court should at the very least satisfy two conditions. Firstly, the evidence should be relevant. Secondly, evidence must be *"… derived by the scientific method"* and *"… supported by appropriate validation"*. Cyber Forensics is very technical in nature and therefore grounded in science, including computer science, mathematics and physics [RS05].

In order to ensure that digital evidence is in fact forensically sound, counsel need to investigate the evidence and the investigator's reliability thoroughly. When witnesses are involved, counsel needs to investigate the testing and verification of their theories and practical techniques of Cyber Forensics. Additionally, counsel may investigate differences of opinion among Cyber Forensic experts regarding the validity and acceptance of specific tools and techniques [RS05].

6 Conclusion

Although intense research still needs to be done before live forensic acquisition can formally be introduced into law enforcement, the preliminary study shows that live forensic acquisition measures up to traditional digital forensics. When the volatile nature of forensics as a whole (including live forensics, traditional digital forensics and traditional biological forensics) is considered, it should be clear that forensic soundness is possible. However, similar to biological forensic practices, some minor (controlled) modifications should be allowed, without rendering the digital forensic evidence inadmissible from court.

Bibliography

[Al06] Alink, W., Bhoedjang, RAF., Boncz, PA. & De Vries, AP. 2006. XIRAF – XML-based indexing and querying for digital forensics. *Digital Investigation.* Volume 3, Supplement 1. Pp 50 - 58.

[Ar05] Armstrong, B. 2005. *Detecting Microsoft virtual machines.* Available from: http://blogs.msdn.com/virtual_pc_guy/archive/2005/10/27/484479.aspx (Accessed 4 August 2008).

[Au08] Australian Government. 2008. *How Do I Protect and Handle Magnetic Media?* Available from: http://www.naa.gov.au/records-management/secure-and-store/physical-preservation/faq/ magnetic-tape.aspx (Accessed 25 February 2008).

[Be05] Bedford, M. 2005. Methods of discovery and exploitation of Host Protected Areas on IDE storage devices that conform to ATAPI-4. *Digital Investigation.* Volume 2, Issue 4. Pp 268 - 275.

[Be06] Bejtlich, R. 2006. Forensically Sound Evidence. *TaoSecurity.* Available from: http://taosecurity.blogspot.com/2006/08/forensically-sound-evidence.html (Accessed 20 March 2008).

[BJ05] Brungs, A. & Jamieson, R. 2005. Identification of Legal Issues for Computer Forensics. *Information Systems Management,* Volume 22, Number 2. Pp 57 - 66.

[Br05] Brown, CLT. 2005. *Computer Evidence: Collection and Preservation*. Charles River Media. Available from: http://www.charlesriver.com/resrcs/chapters/1584504056_1stChap.pdf (Accessed 18 April 2008).

[Ca07] Casey, E. 2007. What does "forensically sound" really mean? *Digital Investigation.* Volume 4, Issue 2. Pp. 49- 50.

[Ch07] Church, CA. 2007. *Long Term Hard Drive Storage and Data Integrity*. Available from: http://photo.net/bboard/q-and-a-fetch-msg?msg_id=00NXpz (Accessed 25 February 2008).

[Co07] Computer Network Defence. 2007. *Anti-Forensic Tools*. Available from: http://www.networkintrusion.co.uk/foranti.htm (Accessed 20 June 2008).

[Fe07] Fei, BKL. 2007. *Data Visualisation in Digital Forensics*. University of Pretoria. Available from: http://upetd.up.ac.za/thesis/submitted/etd-03072007-153241/unrestricted/dissertation.pdf (Accessed 17 January 2008).

[Gh06] Ghelani, S. 2006. *Chain of Custody - A Suspect's Chargesheet*. Available from: http://images.google.co.za/imgres?imgurl=http://www.niiconsulting.com/checkmate/wp-admin/images/0206/cocfrm.jpg&imgrefurl=http://www.niiconsulting.com/checkmate/2006/02/chain-of-custody-a-suspects-chargesheet/&h=407&w=500&sz=25&hl=en&start=16&um=1&tbnid=vm5fqnvwwpkYM:&tbnh=106&tbnw=130&prev=/images%3Fq%3D%2522chain%2Bof%2Bcustody%2522%26um%3D1%26hl%3Den%26sa%3DN (Accessed 25 February 2008).

[Ha08] Halderman, JA., Schoen, SD., Heninger, Na., Clarkson, W., Paul, W., Calandrino, JA., Feldman, AJ., Appelbaum, J. & Felten, EW. 2008. Lest We Remember: Cold Boot Attacks on Encryption Keys. Proc. 2009 USENIX Security Symposium. Pp 1 - 16.

[Hi07] Hilley, S. 2007. Anti-forensics with a small army of exploits. *Digital Investigation.* Volume 4, Issue 1. Pp 13 - 15.

[Jo07] Jones, R. 2007. *Safer Live Forensic Acquisition*. University of Kent at Canterbury. Available from: http://www.cs.kent.ac.uk/pubs/ug/2007/co620-projects/forensic/report.pdf (Accessed 11 January 2008).

[KH02] Kruse II, WG. & Heiser, JG. 2002. *Computer Forensics: Incident Response Essentials*. Addison-Wesley: Boston.

[Le08] LexisNexis. 2008. *Preserving Chain of Custody in E-Discovery.* Available from: http://www.lexisnexis.com/applieddiscovery/clientResources/techTips9.asp (Accessed 25 February 2008).

[LK04] Leigland, R. & Krings, AW. 2004. A Formalisation of Digital Forensics. *International Journal of Digital Evidence*. Volume 3, Issue 2. Pp 1 – 32.

[Mu06] Murr, M. *Windows Incident Response – What is 'forensically sound"?* Available from: http://windowsir.blogspot.com/2006/08/what-is-forensically-sound.html (Accessed 4 August 2008).

[Ni03a] NIST. 2003a. *Hardware Write Blocker Device (HWB) Specification*. National Institute of Standards and Technology. Available from: http://www.cftt.nist.gov/HWB-posted.pdf (Accessed 26 February 2008).

[Ni03b] NIST. 2003b. *Software Write Block Tool Specification & Test Plan*. National Institute of Standards and Technology. Available from: http://www.cftt.nist.gov/documents/SWB-STP-V3_1a.pdf (Accessed 26 February 2008).

[Ni06] Nikkel, BJ. 2006. Improving evidence acquisition from live network sources. *Digital Investigation*. Volume 3, Issue 2. Pp 89 – 96.

[NTV05] Nykodym, N., Taylor, R. & Vilela, J. 2005. Criminal Profiling and Insider Cyber Crime. *Digital Investigation.* Volume 2, Issue 4. Pp 261 - 267.

[Pa07] PARALAN. 2007. *Computer Forensic Protection - SCSI Write Blocker - Models SR14A and SR15A SCSI Forensics*. Available from: http://www.paralan.com/sr14.html (Accessed 26 February 2008).

[RS05] Ryan, DJ. & Shpantzer, G. 2005. *Legal Aspects of Digital Forensics*. Available from: http://www.danjryan.com/Legal%20Issues.doc (Accessed 26 March 2008).

[Sc05] Scalet, SD. 2005. *How To Keep A Digital Chain Of Custody*. Available from: http://www.csoonline.com/read/120105/ht_custody.html (Accessed 25 February 2008).

[St08] Stimmel, CL. 2008. *Best Practices for Computer Forensics in the Field*. Available from: http://ezinearticles.com/?Best-Practices-for-Computer-Forensics-in-the-Field&id=124243 (Accessed 10 January 2008).

[Tr94] Trench, RL. 1994. Chain of Custody: Keeping Track of Property and Evidence. *International Association for Property and Evidence, Inc*. Evidence Log - 1994 Vol 94, No 4. Available from: http://images.google.co.za/imgres?imgurl=http://www.iape.org/EvidenceLog/1994-04/sample-chain-of-custody-tag.gif&imgrefurl=http://www.iape.org/EvidenceLog/1994-04/chain-of-custody-keeping-track.html&h=265&w=508&sz=16&hl=en&start=4&um=1&tbnid=thR-tyQLVby4FM:&tbnh=68&tbnw=131&prev=/images%3Fq%3D%2522chain%2Bof%2Bcustody%2522%26um%3D1%26hl%3Den%26sa%3DN (Accessed 25 February 2008).

[Us05] US-CERT. 2005. *Computer Forensics.* Available from: http://www.us-cert.gov/reading_room/forensics.pdf (Accessed 20 March 2008).

[Vi06] Vidas, T. 2006. *Forensic Analysis of Volatile Data Stores*. CERT Conference. Available from: http://www.certconf.org/presentations/2006/files/RB3.pdf (Accessed 27 March 2008).

[Wi08a] Wikipedia, the free encyclopaedia. 2008a. *Uncertainty Principle*. Available from: http://en. wikipedia.org/wiki/Uncertainty_principle (Accessed 25 March 2008).

[Wi08b] Wikipedia, the free encyclopaedia. 2008b. *Observer Effect*. Available from: http://en. wikipedia.org/wiki/Observer_effect (Accessed 4 April 2008).

[WP05] Weise, J. & Powell, B. 2005. *Using Computer Forensics When Investigating System Attacks.* Available from: http://www.sun.com/blueprints (Accessed 27 February 2008).

Reconstructing People's Lives: A Case Study in Teaching Forensic Computing

Felix C. Freiling Thorsten Holz Martin Mink
Laboratory for Dependable Distributed Systems
University of Mannheim, Germany
{freiling|holz|mink}@informatik.uni-mannheim.de

Abstract: In contrast to the USA and the UK, the academic field of forensic computing is still in its infancy in Germany. To foster the exchange of experiences, we report on lessons learnt in teaching two graduate level courses in forensic computing at a German university. The focus of the courses was to give a research-oriented introduction into the field. The first course, a regular lecture, was accompanied by two practical exercises: (1) a live-analysis of a compromised honeypot, and (2) a dead-analysis of a set of hard disks purchased on the web. The second course was a laboratory course with extensive experiments including forensic analysis of mobile phones. We give an overview over these courses and pay special attention to the reports resulting from the exercises which clearly document the ubiquity of data available to forensic analysis.

1 Introduction

Forensic science is the application of science to questions which are of interest to the legal system. In the standard definition [15], *forensic computing* is the gathering, interpretation and presentation of evidence found on computers. Obviously, methods in forensic computing are very dependent on the kind of legal system they work with, and since the legal systems are very diverse worldwide so are the methods and process models in forensic computing.

The field of forensic computing research has developed primarily in the anglo-american area. This observation is easily validated by looking at the academic forums that have the most reputation in the field [2, 6, 17, 18, 20]. We focus here on the German legal system in which handling of digital evidence has not reached the mainstream portfolio of investigators' skills. In academic education it is therefore difficult to give students firm guidance on how to process digital evidence in a way conforming to the legal system, to interact with the legal system and to testify as an expert witness in front of a court. It is therefore still a question of much dispute how to teach forensic computing in an academic environment like Germany.

In this paper, we report on lessons learnt in teaching graduate level courses in forensic computing at the University of Mannheim, a German institution of higher education with approximately 11,000 students enrolled in five schools. The courses were given within the School of Mathematics and Computer Science. In writing this report, we wish to motivate

other universities and instructors to enter a discussion and an exchange of experiences in this area.

1.1 Our Courses

To give an orientation, the courses were held in the spring terms of 2007 and 2008. The *first course* (given in 2007) was a lecture entirely devoted to forensic computing. It had a regular attendance of about 30 students. The students were mostly in their 4th year of studies and took the course as part of their Diploma degrees in computer science ("Informatik") and business informatics ("Wirtschaftsinformatik"). The final exam was taken by 21 students. The course was meant to give a research-oriented introduction into the field and explicitly did not want to train participants to immediately become investigators. For example, the common commercial tools like EnCase [4] or FTK [1] were only mentioned briefly and practical lab experiences were mostly unsupervised. This contrasts our course with those offered in many specific Bachelor degree programmes on forensic computing (often called *computer* or *digital forensics*) or the professional training offered by the IACIS organization [5].

Two practical exercises accompanied the course: (1) a live-analysis of a compromised honeypot, and (2) a dead-analysis of a set of hard disks purchased on the web. In this specific iteration of the course we focussed much resources on the design and evaluation of the second exercise. Since there exists (to our knowledge) no generally accepted standard of writing reports about the results of a digital investigation, we also place particular emphasis on this subject here in this paper.

The *second course* (given in 2008) was a laboratory course on computer security in which students undertook team projects in specific fields. The motto of the course was that teams simulate the actions of a CERT (Computer Emergency Response Team). Following this idea, the projects encompassed analysis of malicious websites, binary analysis of malware, tracking of botnets and investigation of phishing incidents. About one third of the course was also devoted to forensic computing. In this part of the course, students had to perform a forensic analysis of two floppy discs, two hard disks and at least one mobile phone. The results of analyzing the discs were similar to those of the first course, so we do not report them in detail. However, our experiences in the analysis of the mobile phones were interesting as they were our first experiences in this field. Since we are not aware of similar experiences from other German universities, we devote some attention to the lessons learnt here too.

Thirteen students participated in the second course. Like in the first course, they all were registered as students for a Diploma or Bachelor degree in computer science or business informatics.

The field of forensic computing has not yet reached the mainstream of academic teaching in Germany. This is evident from the small number of courses offered on this subject in Germany compared to the numbers in the anglo-american world. To our knowledge, the first course devoted entirely to forensic computing offered by a German university was given by Dornseif at RWTH Aachen University in the winter term 2004/2005 [16] (see also the discussion by Anderson et al. [9]). As some of us co-organized Dornseif's course, we drew strongly from this course when designing ours.

Starting in 2007, at several German universities instructors began lecturing in the field of forensic computing. Among these were Baumgartl at TU Chemnitz [10], Hahndel at FH Ingolstadt and Hammer at FH Offenburg [19]. While an increasing number of other universities is starting to discuss aspects of forensic computing within a general course on computer security, we are not aware of any other courses entirely devoted to the topic of forensic computing.

1.3 Paper Outline

The paper is structured as follows: We give more details about the motivation and the contents of the first course in Section 2. We then present the results of the dead analysis exercise from the first course in Section 3. After that we turn our attention to the second course and the results of the mobile phone investigation in Section 4. We summarize the lessons learnt in Section 5 and conclude in Section 6.

2 Outline of First Course

2.1 Definition of Forensic Computing

Our goal was to give a research-oriented introduction into the field of forensic computing to technically interested students. This meant to abstract mostly from concrete legal regulations and therefore to broaden the definition of forensic computing to a more computer science-like definition. We understand forensic computing not primarily as a tool for the legal system, but also as a tool for understanding security in general. Sound engineering principles dictate a thorough analysis of failures to learn the workings of a system and avoid subsequent failures of the same kind in the future. We define forensic computing therefore as *the discipline to reconstruct the events which lead to a security policy violation in an information system*. Thus forensic computing also includes the analysis of security incidents to learn the tools, tactics and techniques of the attackers and to gather facts needed to improve security in the future.

The course on computer forensics is taught with two lecture hours (in total 90 minutes) per week over a complete semester lasting 12 weeks. It was accompanied by three on-demand non-periodic meetings to hand out and explain the practical exercises and discuss problems that evolved during their pursuit. A final (half-day) meeting was used to present and discuss the results of the dead-analysis exercise that will be presented later in Section 3.

The goal of the course was to provide students with the necessary knowledge to understand digital evidence at a very deep level. A large part of the lecture consisted of deepening topics from operating systems and systems programming courses in areas of specific interest to forensics. For example, the course gave advanced background in computer networks, process management and filesystems with a strong bias towards the latter.

Based on the understanding of how relevant parts of information systems work, we aimed at teaching how to extract and interpret evidence from such systems and to evaluate the validity of the information gathered. The focus was specifically on a low-level, fundamental view on how the extraction of evidence from IT systems works, enabling the students to conduct forensic analysis without anything but the most basic tools. The idea was to follow the insight that investigators should not be restricted by their tools. Instead, investigators should be able to develop tools they need to support them in an optimal way. As mentioned above, commercial software like EnCase were only covered briefly in the lectures. We are convinced that the fundamental knowledge acquired during class enables students to quickly understand the commercial tools available on the market.

The twelve weeks roughly covered the following sequence of topics:

1. Course organization; overview: forensic science and digital evidence
2. Attack patterns and common computer crime; forensic mindset
3. Process models for forensic computing
4. Hard disk technology, imaging, integrity preservation
5. Disk volumes and disk partitions (DOS partition system)
6. File system analysis: Carrier's reference model for file system data [12]
7. File system analysis: FAT
8. File system analysis: NTFS
9. File system analysis: Ext2/3
10. Network and Internet forensics
11. Commercial tools and legal aspects
12. Theoretical forensic computing: Carrier's hypothesis-based approach [14]

Name	Affiliation	Topic of the talk
Steven Wood	Alste GmbH	Performing large digital investigations
Andreas Körner	PricewaterhouseCoopers	Forensics and white-collar crime
Andreas Schuster	Deutsche Telekom	Main memory analysis of Windows systems
Knut Eckstein	ESA	Advanced file system forensic analysis

Table 1: Talks accompanying the first course.

2.3 Invited Talks

The course was accompanied by a series of four invited talks by forensic computing practitioners from industry and law enforcement. The presenters and talk topics can be seen in Table 1. A fifth talk by a law enforcement practitioner from the German Federal Police (BKA) had to be cancelled due to health reasons.

2.4 Exercises

The exercises accompanying the lecture aimed at giving the students opportunity to gather experience with different forensic techniques themselves. We prepared two exercises:

1. Live-analysis of a compromised computer system.
2. Dead-analysis of a real hard disk.

2.4.1 Exercise 1: Live Analysis

The material for the first exercise consisted of a paused VMware image of a compromised machine. The image was taken from material prepared by the Honeynet Project as part of their forensic challenges [21]. Briefly spoken, honeypots are electronic bait, e.g., computers deployed to be probed, attacked, and compromised. Honeypots run special rootkit-like software which permanently collects data about the system and greatly aids in post-incident forensic analysis. Honeypots therefore not only provide a way to gather a large number of individually compromised machines over time, they also deliver a "true" story of how the system was compromised through their monitoring functionality. This is fortunate to instructors who have a better chance to grade investigation results of students.

More specifically, the image consisted of a Red Hat Linux 7.2 system compromised in August 2003 [22]. The skill level required for this challenge was estimated by the Honeynet Project to be "intermediate to advanced". The system contained many traces of compromise, beginning with the network interface being in promiscuous mode and ending with an installed rootkit plus a multitude of deleted files, some of them in Romanian. We considered this to be an ideal image for our course since the technical skills of the students were very diverse. So while some of the students only found little or unconvincing evidence that

the system had been compromised, other students were able to identify the attack vectors to an extent that superseded the "official" solution [13].

2.4.2 Exercise 2: Dead Analysis

To make the second exercise as realistic as the first one, we had planned to have students image and analyze *real* hard disks. Since the focus of this exercise was not primarily on incident response but rather on data recovery analysis, we did not choose Honeypot hard disk images. Instead, we acquired a large amount (about 50) of pre-used hard disks through online auction platforms and asked the students to image and analyze these. Roughly half of the hard disks has IDE and the other half SCSI interfaces. The task for the students was to find out as much as possible about the former owner of the disk.

Because the acquisition process of the disks had taken place several years ago, many of the disks turned out to be broken. Only about 20 disks were readable and could be analyzed. Most of these disks had a capacity below 500 MB which made them slightly easier to handle than modern large volume disks. The disks also contained many different file systems, ranging from FAT16 over FAT32 and NTFS to Ext3. We asked the students to prepare a copy of the disk image on CD or DVD so that they can be re-used in the future even if the hard disk itself would stop functioning.

We set up one of the server rooms in our building which have strict access control checks to play the role of the room where a court or police keep court exhibits (see left side of Figure 1, the set of disks is shown on the right). To access a hard disk, people had to register formally with a lab member and the "evidence" was handed out. This was meant as a starting point for the documentation of the chain of custody. We prepared an investigation workstation in our lab which had many analysis tools pre-installed, but students could also perform the investigation at home.

As a result of this exercise, students had to write an investigation report. Following best-practices we recommended students to use the following structure when writing the report:

1. Formalities: name of investigator, date, reference/file number.
2. List of evidence to be investigated (serial number), documentation of chain of custody.
3. Task description ("find out as much as possible about former owner").
4. Overview over evidence found.
5. Details of acquisition process of evidence.
6. Summary of used tools.
7. Summary of the implications of the evidence found.
8. Appendix: log files, screenshots, hand-written notes, etc.

Most students followed these suggestions. Details of these reports will be discussed in the following section.

Figure 1: Room keeping “court evidence” (left) and hard disks serving as “evidence” in second exercise (right).

3 Dead Analysis Results

In this section we describe in more detail the results of the second exercise in our course on forensic computing: the dead-analysis of hard disks purchased through a large auction platform on the web. Since we placed much resources on the design and evaluation of this exercise, we devote an entire section to it in this paper.

Table 2 gives an overview over all analyzed disks and the written reports. For sake of anonymization and brevity we assigned each disk a unique letter from the alphabet. In total, 14 disks were analyzed (letters A to N). The table shows the disk manufacturer and the size in MB. It also lists the number and page sizes of the reports written by students. Reports are identified by the letter of the disk plus a consecutive number. For example, report G2 is the second report written about evidence disk G. It can be seen that reports usually have a size of at most 20 pages but extreme cases with around 200 pages also exist (I1 and M1). The core part of these reports however was also at most 30 pages long. The remainder of the document consisted of log files, screenshots and recovered user files. Of all the reports submitted a selected number is presented here.

#	Manufacturer	Size (MB)	Reports and their size (in pages)
A	Western Digital	170	A1 (9), A2 (10), A3 (16), A4 (56), A5 (7)
B	Seagate	545	B1 (52)
C	Conner	412	C1 (13)
D	IBM	4330	D1 (19)
E	IBM	30700	E1 (14), E2 (13)
F	Conner	210	F1 (39), F2 (18)
G	Conner	420	G1 (65), G2 (48)
H	Seagate	545	H1 (14)
I	Western Digital	325	I1 (186)
J	Seagate	546	J1 (29)
K	Seagate	8400	K1 (15)
L	Fujitsu	1700	L1 (17)
M	Quantum	170	M1 (211)
N	Conner	406	N1 (13)

Table 2: Overview over analyzed hard disks.

3.1 Disk A, Report A1

Disk A contained a single FAT16 partition which appeared to be empty. Investigating the partition table with a disk editor, the student found traces of a prior Windows98 installation. Using the file carving tool foremost [3] the student tried to reconstruct data but the reconstruction failed. So the student reverted to the low-level Unix tool `strings` to identify readable characters on the raw hard disks. This resulted in recovery of a large number of text files containing invoices for special steel constructions manufactured by a well-known German steel processing company.

Particularly interesting in this report was the fact that the student always used two tools to recover and cross-check evidence found, whenever possible. For example, recovery and analysis of the partition table was performed using both `mmls` and `foremost`.

3.2 Disk A, Report A4

Report A4 also documents that the disk had been used by the same company as reported by A1. However, A4 used a demo-version of FTK [1] to investigate removed or overformatted files. FTK was able to recover many bitmap files from the Windows System folder as well as three Microsoft Excel files with calculations of the total work hours of 10 employees of the company in the year 1995. All evidence is documented with exact offsets on the disk where it was found.

Particularly interesting in report A4 was that the author did a partial web-search on the company and the names of the employees. The company itself had declared bankruptcy recently and now is owned by a Japanese technology company.

Disk B was also empty, but turned out to have been reformatted before being sold. The student was able to recover a FAT16 partition and recover half a dozen MS Word and Excel files. Interestingly, they belonged to the same steel processing company which seems to have been the source of disk A. We assume that both disks were bought at the same time (as part of a larger "batch" of disks).

Report B1 is a typical example of a very technical approach. It contains a lot of numbers and hardcopy technical details from log files but hardly any explanation *why* the investigation was performed in the way it was done. However, the report contains a well-readable executive summary of the found evidence and the implications.

3.4 Disk F, Report F2

The author of report F2 used a demo-version of FTK to analyze the disk. It also contained a FAT16 partition, but this time it had not been formatted before the disk had been sold. The installation appeared to be a Windows for Workgroups 3.11 based on MS-DOS 6.20. FTK was able to recover over 1600 files on the disk.

It turned out that this disk had belonged to a notary from Eastern Germany. The system installation information also gave hints on the remaining network infrastructure (network drives, attached printers, etc.) of the prior user.

The report was the only one that clearly indicated on the title page that the report contained confidential personal data. F2 also was unique in its attempt to guide the investigation through hypotheses. For every step in the investigation, F2 listed a couple of hypotheses that seemed possible based on the previously collected evidence. These hypotheses were then scrutinized in a systematic fashion, leading to new hypotheses etc.

3.5 Disk M, Report M1

Report M1 was by far the longest and largest report submitted for this exercise. This seems only partly to be the result of the skill and determination of the student, because disk M contained an overwhelming wealth of information. The disk had been used in a computer of a small company selling and building kitchen interiors. The main user was a carpenter who also used the computer for private purposes. The student was able to recover more than 5000 files that seem to have been created and used between the years 1992 and 2000.

From these files it was possible to reconstruct an extensive personal profile of the former owner and his family. Figure 2, taken from report M1, for example shows the complete family tree of the former owner. The disk also contained (among other data) telephone numbers, social security and insurance numbers, bank account numbers, registration numbers of cars, and details about medical treatment for almost all (living) members of the

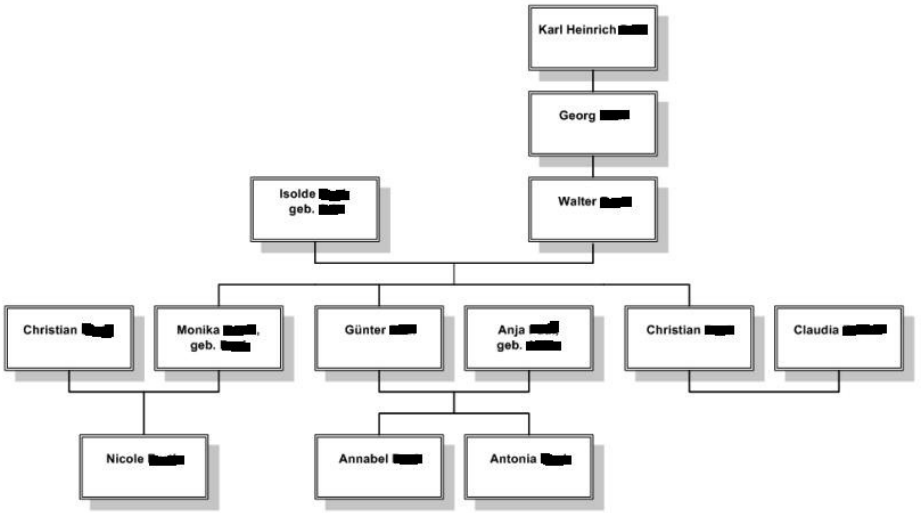


Figure 2: Reconstructed family tree taken from report M1.

family. In summary, it was possible to reconstruct the life of many individuals from this report.

3.6 Disk G, Report G1

Students sometimes make mistakes in data acquisition and analysis. For example, analysis of disk G showed traces of data from two totally different sources: relatively old data from a user called Mike D. from Bochum, and relatively new data from a student from the university of the authors. It seems that this disk has been acquired some time ago and was used in the meantime by a student for writing a project report for another professor at our university.

4 Second Course

We now report on our experiences with the second course, a laboratory course offered in the spring term of 2008.

4.1 Course Outline

The laboratory course was organized as a simulated CERT (Computer Emergency Response Team). A CERT is an organizational entity responsible for handling computer security incidents in the organization. Thirteen students formed 4 CERT teams (3 teams with 3 participants and one with 4). All had to investigate the same set of incidents and write up a final report.

The areas touched in the course were malware, botnets, phishing and forensic analysis. Students had to "apply for a position" within the CERT and were selected after a short interview. The course ran for a full semester (13 weeks) and consisted of a sequence of projects which were disguised as "real incidents". While all projects had to be handled with a forensic mindset, only the final couple of projects was really concerned with forensic analysis. These projects consisted of a forensic analysis of two floppy disks, two hard disks and at least one mobile phone. The floppy disks were meant as an initial exercise and had been prepared from scratch by copying onto and deleting some files on them. The hard disks had been acquired from an administrative unit of University of Mannheim which regularly collects old and broken hardware and makes it available for other use within the university. The mobile phones were second hand phones and had been bought through a forum website on the Internet.

Since the results of the floppies and hard disks were similar to those described in Sect. 3, we focus on the results of the mobile phone analysis here.

4.2 Results of Mobile Phone Analysis

As confirmed by the practitioners who gave talks in the first course, mobile phones are one of the prime sources of digital evidence today. They contain (increasingly) large portions of flash memory that remain in the phone even if the SIM card is removed. The challenge with mobile phone analysis is not only handling the digital evidence from memory but also acquiring the evidence from within the phone.

To enable the acquisition of memory images, we bought a *twister box*, a commercial device usually used to install new firmware on the phone. It can also be used in read-only mode to access the memory chip in the system. A twister box usually comes with an impressive set of cables and adapters since every manufacturer has his own way of connecting peripherals to his phone.

In total, we had purchased 10 phones for a price of around 130 Euros; 7 of them were analyzed in the course. Since we wanted to be sure that common analysis software as well as our cables were compatible with the acquired phones, we restricted our purchase to popular models. The result was that all purchased phones were manufactured by Nokia: 6 model 3510i (see Fig. 3), two 6510, one 6030 and one 6800. Students prepared memory images using the twister box and then started data analysis on the image.

Since mobile phones usually use standard file systems on their flash memory, the challenge

Figure 3: Nokia 3510i analyzed by Team 1.

in data analysis lies rather in the proprietary file formats used by different manufacturers. Fortunately, the analyzed phones belonged to a popular and already rather old series of phones and therefore the data formats for short messages and other types of data was documented. Most students started data analysis using the same low-level tools used for dead-analysis of hard disks and then turned to the use of file carvers. File carvers allowed to reconstruct several graphics files, most of which were screen savers or logos of the mobile operators. Only one picture could be reproduced that seemed to stem from a mobile phone camera. Since none of the investigated phones had a built-in camera, it is unclear where this photo came from.

Since the open source tools did not help in analysing proprietary data, students turned to the use of commercial tools. Since we had no budget for these tools, they were restricted to evaluation versions. Independent research by all teams lead them to use one particular tool called *Cell Phone Analyzer* [11]. Using this tool it was very easy to recover the full phone book, the full call history as well as all active and all deleted short messages (SMS). Using this information it was relatively easy to identify the previous owner of the phone.

The evaluation version of Cell Phone Analyzer obfuscates the output of recovered SMS with dots instead of letters in random places of the SMS. The students quickly analyzed this behavior, verified that dots were placed in truly random places, and quickly wrote short scripts that merged the output from different invocations of the program. By this technique they were able to construct an (almost) perfectly readable transcript of SMS traffic.

A particularly interesting discovery was made by Team 2, that recovered a long SMS conversation (in German) discussing the pleasures of bodily union in some detail (see Fig. 4). Team 2 concluded that the former owner of the phone would not be amused seeing this information being made public. The team also discovered a series of SMS on that same phone, partly in a foreign language and dated older than the ones already mentioned. So it seems this phone had (at least) two former owners.

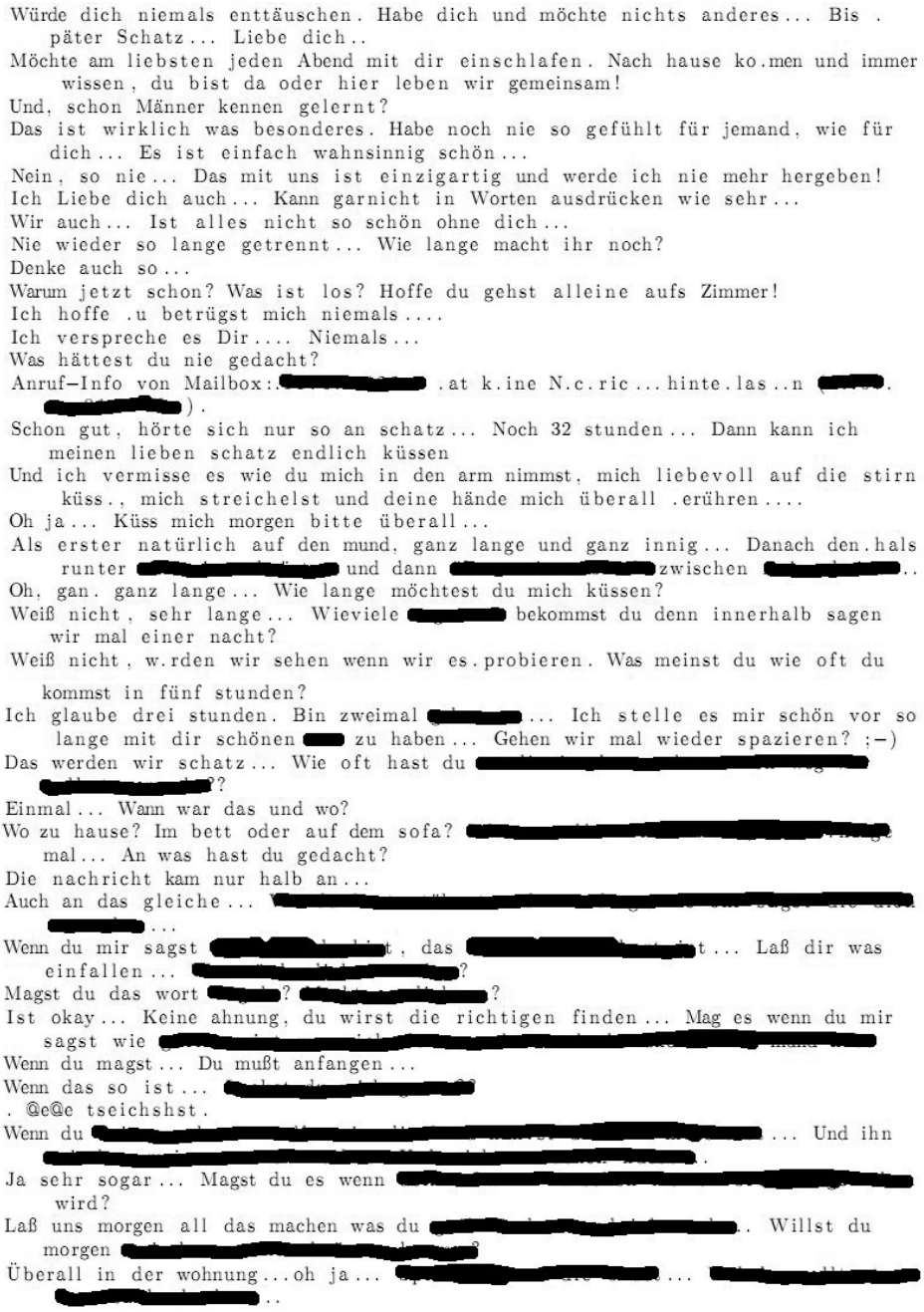


Figure 4: Excerpts of short messages recovered by Team 2, personal details and obscene language removed.

5 Lessons Learnt

We now discuss several lessons we learnt from our courses.

5.1 Tool Support

Following the bias towards low-level open source tools in both courses, most students started their analysis by using open source tools like the Sleuthkit [7] and `foremost` [3] in the beginning. In the first course, roughly 6 students used evaluation copies of the commercial software FTK [1] and X-Ways Forensics [8]. One student was able to use a licensed version of EnCase [4]. Regarding those disks for which multiple reports are available we did not see any fundamental difference in the depth and breadth of results achieved between open source and commercial tools. The main difference seems to have been that the analysis results were obtained *faster* using commercial tools, but no real data exists to measure the effort.

In the second course, no real open source tools exist to analyze memory images of mobile phones. After a first scan of the images using hex editors or command line tools like `strings` students reverted to evaluation copies of commercial tools. An interesting point here is that, although we did not give advice towards specific tools, all teams ended up using the same tool [11] because it was the only commercial tool offering a free evaluation version. Noteworthy is also that they were quickly able to circumvent the restrictions built into the evaluation copy by writing scripts to merge multiple analysis results and thereby mask out the random blinds. This is a clear indication to us that programming experience is a powerful skill to overcome restrictions of analysis tools (be they artificial or not).

5.2 Experiences with Documentation

In the first course, the recommended structure of the reports turned out to always lead to good results in documentation. Most students forgot to document the chain of custody in a rigorous fashion. Also, only half of the students documented sufficiently their analysis environment and argued that it was trustworthy.

Participants in the second course had partly followed the first course and so the quality of the reports in the second course was on average higher than in the first course.

From the level of detail of the personal data found, it also makes sense to add a general initial notice to the structure of the document that the document may contain sensitive personal data and that the report is not meant for public distribution. At the moment, we store all reports only in our lab's internal network. They are not accessible from outside.

Reports need to follow a quality control process too, i.e., they should contain a version number and a change log. The names of the authors responsible for individual parts of the document must also be clearly marked. Furthermore we will demand a short executive

summary “for the attorney” at the beginning.

In general we found that we need to motivate students more to focus on the prospective readers (CEOs, attorneys, forensic examiners). For example, the technical parts of the report must be written for an expert reader who is trying to validate the documented results. This helps to follow the report and allows to quickly reproduce the results from the same initial evidence. For this it is better, for example, to give exact disk sector numbers as reference for a particular piece of evidence on the disk rather than giving a screenshot of the evidence “on screen” during the investigation. Reproducibility is also improved by clear language. For example, instead of writing “the evidence disk is in a bad state”, an examiner should rather write: “the tool `dd` produced read errors in the following disk sectors”.

Finally, the report should follow standard academic practices, i.e., it should have a clear structure, clear statement and a set of bibliographic references if necessary. This is what students should learn anyway.

6 Conclusions and Open Questions

The first course was evaluated at the end of the semester. Of roughly 30 students who regularly attended, 15 gave feedback via a questionnaire. The results were very encouraging: On a scale between 1 (best) and 6 (worst), the course scored a total of 1.27 (standard deviation 0.44). Regarding the specific question whether the course has improved their skills as computer scientists, the students unilaterally voted 1.0. So from this feedback it seems that students liked the course.

There are a couple of open research questions coming out of this work. First of all, there is the legal question whether we are allowed to perform forensic analysis on hard disks which have been intentionally erased. Who is the owner of the data found? Who are we allowed to pass it to? What steps should students take when they find personally identifiable information during analysis? To investigate this question, a bachelor thesis currently is conducted at our Lab.

Connected to this question is another one: How can we create a large set of hard disk images for analysis automatically without using actual personal data? This is a rather technical problem which we wish to tackle in the future.

Acknowledgments

We wish to thank Maximillian Dornseif for helpful discussions.

References

[1] AccessData Corporation. `http://www.accessdata.com`.

[2] Digital Forensics Research Workshop. Internet: http://www.dfrws.org, 2007.

[3] foremost. http://foremost.sourceforge.net/.

[4] Guidance Software, Inc. http://www.guidancesoftware.com.

[5] IACIS Homepage. Internet: http://www.cops.org, 2007.

[6] International Journal of Digital Evidence. Internet: http://www.ijde.org, 2007.

[7] The Sleuthkit. http://www.sleuthkit.org.

[8] X-Ways Software Technology AG. http://www.x-ways.net.

[9] Philip Anderson, Maximillian Dornseif, Felix C. Freiling, Thorsten Holz, Alastair Irons, Christopher Laing, and Martin Mink. A comparative study of teaching forensics at a university degree level. In Oliver Göbel, Dirk Schadt, Sandra Frings, Hardo Hase, Detlef Günther, and Jens Nedron, editors, *IT-Incidents Management & IT-Forensics - IMF 2006, Conference Proceedings, October, 18th-19th, 2006, Stuttgart*, volume 97 of *LNI*, pages 116–127. GI, 2006. ISBN 978-3-88579-191-1.

[10] Robert Baumgartl. Vorlesung Computerforensik. Internet: http://rtg.informatik.tu-chemnitz.de/lehre/cf/cf.php, 2007.

[11] BKForensics. Cell Phone Analyzer by BKForensics. Internet: http://www.bkforensics.com/CPA.html, 2008.

[12] Brian Carrier. *File System Forensic Analysis*. Addison-Wesley, 2005.

[13] Brian Carrier. Honeynet project - scan of the month #29. Internet: http://www.honeynet.org/scans/scan29/sol/carrier/index.html, October 2003. Solution to Honeypot Challenge 29 [22].

[14] Brian D. Carrier. *A Hypothesis-Based Approach to Digital Forensic Investigations*. PhD thesis, Purdue University, 2006. Available as CERIAS Technical Report 2006-06.

[15] Eoghan Casey. *Digital Evidence and Computer Crime - 2nd Edition*. Academic Press, 2004.

[16] Maximillian Dornseif. Lecture on computer and network forensics (course material). Internet: http://md.hudora.de/presentations/forensics/, 2005.

[17] Elsevier. Digital Investigation. Internet: http://www.elsevier.com/locate/diin, 2007.

[18] GI e.V., Fachgruppe SIDAR. IMF 2007 – Conference on IT Incident Management and IT Forensics. Internet: http://www.gi-ev.de/fachbereiche/sicherheit/fg/sidar/imf/imf2007/, 2007.

[19] Daniel Hammer. Computer Forensik. Internet: http://security.fh-offenburg.de/forensics.php, 2008. Course homepage at FH Offenburg, in German.

[20] IEEE. IEEE Transactions on Information Forensics and Security. Internet: http://www.ieee.org/organizations/society/sp/tifs.html, 2007.

[21] The Honeynet Project. Honeynet challenges. Internet: http://www.honeynet.org/misc/chall.html, 2008.

[22] The Honeynet Project. Scan of the month: Scan 29. Internet: http://www.honeynet.org/scans/scan29/, September 2003.

Network Flow Security Baselining

Tsvetomir Tsvetanov[1], Assoc.Prof.Dr. Stanislav Simeonov[2]

[1]Sofia University
James Boucher Blvd 5
Sofia, Bulgaria
[2]Burgas Free University
San Stefano Blvd 62
Burgas, Bulgaria
ttsvetanov@gmail.com
stan@bfu.bg

Abstract: Networks are a critical factor in the performance of a modern company. Managing a network is as important as managing any other aspect of a company's performance and security. There are many tools and appliances for monitoring the traffic and analyzing the security aspects of the network flows. They are using different approaches and they rely on different characteristics of the network flows. The network researchers are still working on a common approach for security baselining that might enable early alerts. This paper is focusing on the security baselining based on a simple flow analysis utilizing the flows measurements and the theory of the Markov models.

1 Introduction

Two major approaches are utilized in the theory of computer network modeling: the queuing theory [Da05] and the hidden Markov models [Ra89]. The queuing theory addresses basically the models for the ubiquitous service. The general issue of the contemporary online services is how we can assure that every customer would be satisfied in fairly short timeline. This issue is the main objective of the ubiquitous service. Apparently, the queuing theory is providing an approach for analysis of the quality of service (QoS) issues. QoS was addressed in the early days of the computer networking and it is still ongoing issue for the modern telecommunications. The one aspect of the QoS problem is the quality measurement. The quality assessment might be done in different ways: measuring the delay of the system access, the time frame for the data access, taking an account the quantity of the lost information, the voice signal distinction or the data purity. Usually the quantity measurements are random variables and the results are the average rates of the variable distributions. The major issue regarding the QoS delivery is the price for the quality that the

end user would agree to pay. The price generally depends on the resources needed for providing the service quality level. The QoS analysts are focused on that problem, building up the optimal share between the quality rate and the price of the service.

For more than three decades there was a trend against ubiquitous service in the packet-switching networks. The main reason for that was the price decrease with simply increasing the capacity of the shared resources. On the other hand, there was no mechanism in place that might provide certain service quality level on a reasonable price. Then the queuing theory has been adopted as the main approach for solving that problem for building the optimal rate between the shared resources value and the quality of the service that the end user achieve. The queuing theory also aided the analysts for discovering those internal network processes that allocate the significant part of the network resources. Once knowing the processes, the analysts might be able to decide on the policy of the resources allocation in order to ensure the target level of the service quality.

The seconds approach utilizes the hidden Markov models (HMM). In fact, the queuing theory also uses the Markov chains and models but we are addressing the network analysis based on the Markov models only. The hidden Markov models are stochastic systems that have two parts: internal hidden part and a visible part that is manifesting the hidden behavior to the observer. The internal states of the model are hidden for the observer. We call them also *hidden states*. The system moves from state to state within the discrete time intervals. In the context of the computer networks we can think about the hidden states as internal processes or events occurring in the network environment. Besides the hidden part, the hidden Markov model has a "visible" part. The observer can "see" the hidden state manifestation in means of *observed symbols* emitted by the internal processes. The set of the observed symbols is the symbols *alphabet*. When we are talking about hidden Markov model application for network analysis, we can also call that alphabet of the observed symbols network alphabet. Should we pick up the hidden Markov model for network analysis application, we will need to define the set of the hidden states and the network alphabet, matching them to particular semantics in the context of the network analysis application.

Unlike the queuing theory approaches, the network analysis based on the Markov models are not addressing the quality of service issues; it rather solves the problem for pattern recognition onto the network flows. Using the HMM the analysts are building different kind of network profiles for the different network environments. Those profiles aid the analysts in abnormal network behavior detection as well as for finding specific trends in the network environment utilization. The profiles also might be applied in network security management. When building a set of profiles within certain timeline, the analysts are able to determine the differences and then detecting potential threads.

In this paper we will focus on the network flow modeling for network behavior pattern recognition. In order to address this problem we are choosing the HMM approach. Once having built the model patterns we would be able to recognize if our network is handling a

legitimate traffic or the traffic flow is offensive and malicious. The reason for choosing the HMM approach is also based on the following facts:

There are certain internal processes running in the network and they are hidden for the end-users and even for the monitoring tools. Those processes are affecting the network resources and are changing the infrastructure behaviour. Due to the internal processes the network is moving to another state within certain discrete time interval. Every internal state transition emits external phenomena in means of network traffic attributes like specific header values, payload content, combination of both, or specific sequence of network packets on the wire. All those manifestations might be different for the particular infrastructures, environments and also might vary between the network segments within an autonomous system.

The comprehensive network devices are able to provide some predefined characteristics of the network flow to the network monitors. That information, of course, cannot be unlimited by quantity and by property, obviously because this is additional function for the network devices and it produces additional overhead on the device's resources. The device vendors addressed the network monitoring requirement by aggregation and export of the protocol headers in predefined data structures. Using that information the network monitors are able to collect traffic data, i.e. the network symbols emitted by the internal processes.

2 The Network Alphabet

Using a network tool for collecting the traffic tokens we can build up a network profile for certain discrete time intervals. The basic idea of this task is to receive two groups of observed *network words*:

- Words that were read in a time of normal network behavior, when there was no anomaly in the traffic flow or the anomalies were too weak to affect the network assets and performance.
- Words observed during long running intensive network flow anomaly when the there might be a Denial-of-Services or similar attack executed against the network resources.

The definition of the network alphabet that we are going to use is based on the relation between the communication peers from the previous traffic entry and the current one. The comparison of those two traffic entries depends on the network and transport protocol header values. As we are focusing on IP network stack analysis, the following values are taking in account:

- Source IP address
- Source port
- Destination IP address

- Destination port
- Protocol code (e.g. ICMP=1, TCP=6, etc.)

The following table contains the network alphabet semantics.

A	The communication peers are absolutely the same. The previous traffic flow entry and the current entry are absolutely the same, i.e. those are the same connection flows.
B	The previous and the current traffic flow entry have one different port, either source or destination, for example: **TCP / 10.10.167.154**:2311 – **10.10.10.5:80** **TCP / 10.10.167.154**:2314 – **10.10.10.5:80**
C	The traffic flow entries have one different IP address, either source or destination, for example: **UDP / 10.10.10.8:53** – 10.10.16.4:**53** **UDP / 10.10.10.8:53** – 10.10.16.5:**53**
D	The traffic flow entries vary by the ports and the IP addresses are still the same, for instance: **TCP / 10.10.10.3**:162 – **10.10.10.8**:53 **TCP / 10.10.10.8**:543 – **10.10.10.3**:53
E	The traffic flow entries holds the same IP address and port pair for one of the peer, either source or destination, but the other peer IP address and port does not match. The most common reason for this kind of consequence could be that the matching peer is actually a service that was requested from different client as you can see in the flow entries example: **TCP** / 10.10.167.154:2311 – **10.10.10.5:80** **TCP** / **10.10.10.5:80** - 10.8.130.35:45319
F	The traffic flow entries, the previous and the current, hold the same IP address, either source or destination, but all other peer properties are different: **TCP** / 10.10.167.154:2311 – **10.10.10.5**:443 **TCP** / **10.10.10.5**:80 - 10.8.130.35:45319
G	The current and the previous traffic flows match only by one port, either source or destination: **UDP** / 10.11.101.230:21439 – 10.10.10.5:**161** **UDP** / 10.16.116.159:19012 - 10.10.10.4:**161**
H	The traffic flow entries have absolutely nothing to deal each other.

Table 2.1: The network alphabet semantics

It is important to note that all the symbols except the symbol ‘H’ require the same transport protocol code. When the parameters source IP address, destination IP address, source port,

and destination port, do not match between the traffic flows, the emitted symbol will be ‘H’ no matter the protocol is the same or not.

The obvious purpose of this network alphabet semantics is to find out the following events and processes occurring in the network environment:

A long sequence of ‘A’ symbols would match to a communication between two network nodes over the same communication channel. The common scenario is when the communicating pairs are connected over TCP channel and they are exchanging data sporadically. In this case the network device would aggregate the traffic flows in individual entries as the flow cache at the network device memory would expire before the TCP channel is closed.

A long sequence of ‘B’-s then would be long running communication between a client and a server similar to that described for the symbol ‘A’ but with the difference that the client is using several communication channels to the server. As every channel might be able to finish its lifecycle within the traffic flow cache expiration at the network device, the communication data might be fully aggregated and exported as a single traffic flow entry.

If we find out a sequence of many ‘C’-s it would mean a service that is used by many different clients. However, the service itself requires the clients to use a specific port number rather than an arbitrary number. Examples for this kind of services are Domain Name Service (DNS) and Hot Stand-by Routing Protocol (HSRP).

Too many occurrences of ‘D’-s might be caused by an excessive communications between two nodes over different kind of protocols and services. The common case is when the peers are connecting each other on ad-hoc application ports.

Similarly to the symbol ‘C’, the occurrences of ‘E’ match a service used by different clients. However, unlike ‘C’, the service does not require the clients to connect from specific port and they can use arbitrary values. This is the most common scenario of service utilization.

If we observe many ‘F’ symbols, it might mean that different clients are accessing different services from the same server host.

Similarly to ‘F’ observations, if we see many ‘G’-s then it might be the case when different clients are accessing same kind of service running on a different host, for example when the service is provided in a high-availability mode by a server farm.

The occurrences of ‘H’ symbols match fully heterogeneous traffic flow. For the normal network behavior this is expected t be the most common symbol in the observed network words.

3 Collecting the Network Symbols

In order to collect the network symbols from the traffic flow data, we will need a software system that interfaces the data and translates the flow entries into network words. The tool implements the following functions:

- Collects the traffic flow data from the network devices utilizing the standard NetFlow data format [Cl04].
- Translates the traffic flow data into network symbols according to the network alphabet semantics. The results (the network words) then are exported into a file or set of files onto the file system.

Due to performance considerations and in order to make a flexible design we separated those two tasks in two different processes communicating via shared memory segment. The first process collects the data from the devices via UDP port. Then the collected NetFlow traffic data is preprocessed and put into an IPC message queue for handling by the other process. The second process takes the data from the message queue and compares the traffic entries. Depending on the relation between the neighbor traffic flow entries, the process decides on what network symbol is emitted. The decision to use a shared segment and two processes rather than a single process that reads and handles the data is based on the consideration that the single process might fail to read all data when it is busy with processing it. For example, if too many devices are forwarding the NetFlow data to the tool, the buffer could be quickly filled up as the process is spending more time in emitting the network symbols. In this situation the next traffic flow data might be missed and the output would not be relevant. Using a shared memory we are able to overwhelm and process promptly the input data. The second process can also continue emitting the symbols even if the first one was stopped if there are still data in the message queue.

The tool was developed and deployed into a real network environment for making the field tests. The adjacent devices were configured for traffic data export to the tool. In order to avoid the data duplicates the data was captured by the ingress routers only. The tool collected and translated the data into network symbols replacing the symbol ‘H’ with dot (‘.’) just for easy-to-read purpose as we expected to see much more ‘H’-s for the normal traffic behavior. The network symbols were collected in fairly long period of time making sure different network states were captured. The results are shown in the following figures.

Figure 3.1 shows mostly dots that in fact are ‘H’ symbols as we replaced them with dots. According to the network alphabet definition, as many ‘H’-s occur in the network words as heterogeneous the traffic is. The heterogeneous traffic normally stands for normal network behavior. Some researchers are focusing exactly on the homogeneity flow measurements in order to detect network threads. According to the conclusions done, the homogeneous traffic is most likely produced by machine rather than by normal human interactions. That’s why

the homogeneous (machine-generated) traffic might be an alert for an offensive traffic flow running on the wire.

Figure 3.1: Networks words captured during a normal network behavior

The next figure 3.2 comprises of parts with fairly heterogeneous symbols, however, there are some parts with mostly ‘C’-s in a long sequences. According to the alphabet semantics, there adjacent traffic flow data contain the same peer host and port, either source or destination, while the peer is different between the flow entries. This might be the common case of using the same service from different clients. The service does not require specific client port number. Depending on the length of the ‘C’ sequences and depending on the real time interval (note that the symbols are matching a discrete time interval), this could be either legitimate traffic flow (legitimate clients are excessively using the service) or it could be a flooding targeted to the service provider host (a denial-of-service attack). We consider the example on figure 3.2 as quasi-homogeneous flow.

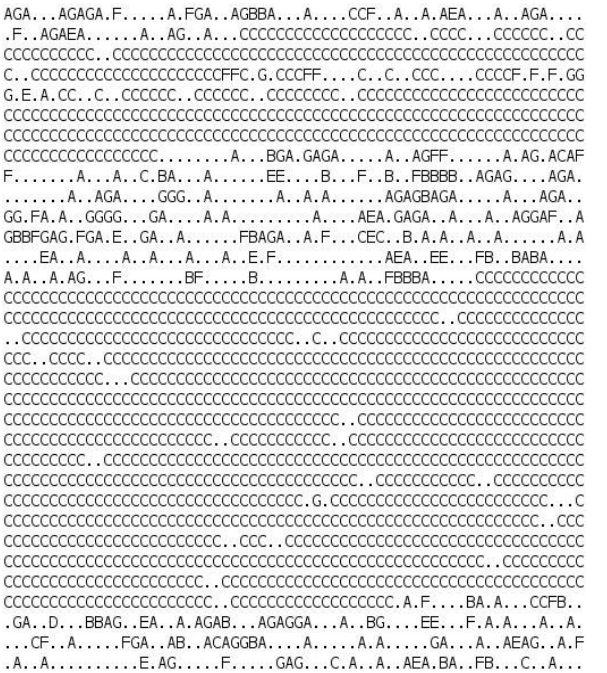


Figure 3.2: Network words emitted during suspicious traffic flows

The example on figure 3.2 refers to a suspicious traffic flows to some specific service. This is the case when the ‘C’ symbol occurs in long sequences. The same traffic model would be observed when the server ports are scanned or when the service is excessively flooded with useless messages. Both cases are denial-of-services attacks with different goals: scanning and flooding. However, we have to take in consideration that the excessive packet flows to a particular service does not necessarily means the traffic flows are malicious. It is absolutely true that the offensive port scanners and flooders are generating those kinds of network words. However, the same words would be emitted even by the legitimate clients if in that particular moment the clients are connecting to the service. Usually this might be the typical case when in the morning business hours the employees are authenticating onto the Intranet and starting to access their common services. The other situation would be when a scheduled task runs and it produces a big bulk of packet flow onto the wire. Another example that comes in mind is the FTP protocol uploads and downloads. When huge files are transferred, the connection stays open a long time. Then the network monitors would observe long ‘A’-s and ‘B’-s, although it is absolutely legitimate network usage. Due to those reasons, when talking about a malicious traffic actually we are talking about a network behavior that might be exposed when a malicious traffic is going on. In order to be more precise we will call those traffic flows mostly suspicious rather then malicious. The suspicious traffic is a general

definition that includes the real malicious and offensive traffic as well as the legitimate traffic that behaves the same way as the malicious does.

What would be then if too many long sequences of 'A'-s and 'B'-s were observed in the network words? As we already clarified, the occurrences of A, B, and C, stands for suspicious flows. The difference would be the target. In case of long A-sequences the target could be either source or destination but as we said it might be excessive legitimate traffic. The B-sequences are likely to target a service flooded from the same source host. Also, the same sequence matches to the case when an attacker is scanning the destination ports in a long sequential order. Then the source side is using same port for packets generation. The C-sequence is excessive using of a service by multiple clients (legitimate case) or a denial-of-service flooding (malicious case).

4 Result Analysis

After the network words are collected by the tool and stored into files, we are going to make a post-mortem analysis on the observed flows. Before proceeding to the task, we need to make few definitions used later in the paper.

Let the observed symbols set be $V = \{V_1, V_2 ..., V_M\}$. The set V is the network alphabet as previously defined in table 2.1. As per the definitions the network alphabet size is 8, i.e. M=8. There are two major characteristics that we would be interested during the pre-modeling analysis. The first question to answer is how long the longest sequence of same symbols is, no matter the symbol itself. The next problem is what is the symbol share overall the whole observation discrete period of time. For those purposes we are introducing two definitions over the network alphabet set V.

Definition 1: The longest sequence $\rho_i(T)$ of same symbols V_i in the observed word O for the discrete period of time T, is called *density of the symbol V_i for the observed period T*. The set of all densities $\rho_i(T)$ for the corresponding symbols V_i, $\rho(T) = \{\rho_1(T), ..., \rho_M(T)\}$, is called density of the network alphabet V for the observation period T.

Definition 2: The relation $\phi_i(T)$ of the number of the occurrences of symbol V_i for the discrete period of time T and the length of the observed word O for the same period T is called *frequency of the symbol V_i for the observed period T*.

According to the second definition, the following relation immediately implies $\Sigma \phi_i(T) = 1$, $i \in [0; M]$

Once collected, the network words are calculated against the density and the frequency values. If the words are observed during the normal behavior then we would get ρ and ϕ parameters for the normal traffic model. Using a test program we simulated flowing attack

and collected the network symbol. The overall field tests were made over 24-hours period of time. The results were collected in series of files and then we proceeded with the flow analysis.

The next figures 4.1, 4.2, 4.3, and 4.4 summarize the analysis result. We have processed the network words as follow. The overall period of time was separated in series of time intervals. As the tests ran for 24 hours, we divided the results in 24 output words, one for every day hour. Then we picked up the words and for each of them we calculated the density and frequency for every of the network symbols. Finally, we have had 24 values of the arrays $\rho_i(t)$ и $\phi_i(t)$ for each time interval $1 \leq t \leq T$. The output results were generated as double arrays $\rho(T)$ and $\phi(T)$ and the values were translated in a graphical format.

The four figures are presenting the values of $\rho(T)$ and $\phi(T)$ during the normal flows and during the suspicious flows.

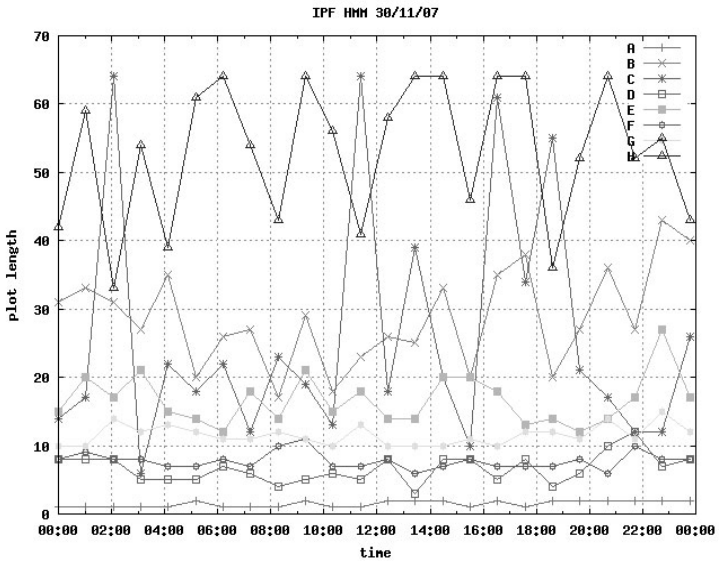


Figure 4.1: Density of the network symbols during the normal network behavior.

Based on the results on figure 4.1 we can make the following conclusions:

- During the normal behavior the highest density belongs to the ‘H’ symbol. This means fairly high flow heterogeneity.
- The densities of ‘B’ and ‘C’ might also be relatively high compare to the other symbols’ densities. As we can see ‘C’ was burst in some of the discrete time intervals, while the ‘B’ was keeping a plateau rates. As we discussed earlier, the bursts on ‘B’ and ‘C’ are

acceptable during the normal traffic flows due to an excessive usage of a specific service by one and more clients.

The graphic on figure 4.2 shows the values of ϕ_i (T) captured for 24h period. The values are percentage of the symbol occurrences over the word length. The 'H' symbol again has the highest share rate. It is over 50% which means it has higher share even than the sum of the other symbols shares. The next share rates are those of the A-s and B-s. The occurrences of A-s might be a long running connection between two particular nodes. Similarly, 'B' symbols mean short term usage of a connection and then opening a new one by the same client but using different client port. That's the way some of the application protocols work. For example, the old HTTP/1.0 does not support keep-alive TCP connections. The client opens a channel, sends the request, retrieves the reply, and finally closes the channel. For each individual request-reply transaction, the protocol requires different TCP channel. Another example could be the DNS queries. Every individual query is provided in a different UDP datagram. Then an individual host might produce too many 'A'-s in a short time if the host is resolving too many names. The common case is the network monitoring software. It scans the network on regular basis in order to build up-to-date topology.

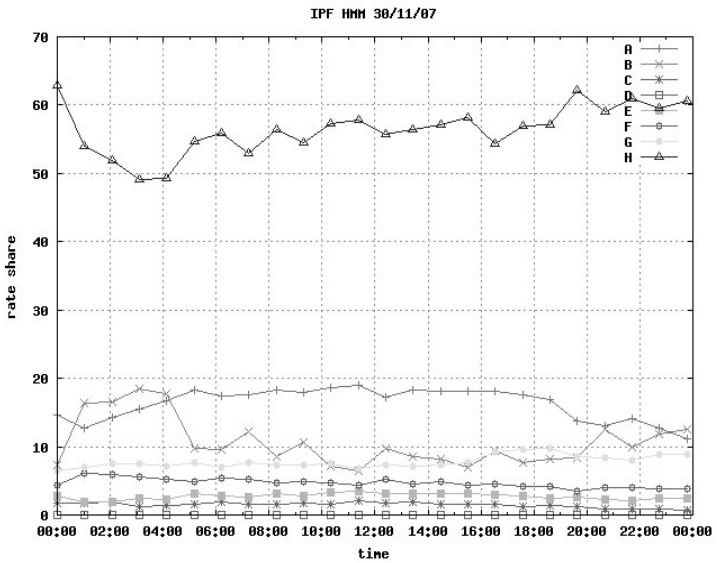


Figure 4.2: Frequency of the network symbols during the normal traffic flow.

During the network symbols collection we simulated an offensive traffic flow. The figures 4.3 and 4.4 show how that traffic emerges in the network words parameters. The figures present graphically the results within the observed time interval 12:00 – 14:00. They are respectively addressing the density and the frequency. Note that the suspicious traffic flows

are running for relatively short time period and usually with fluctuating frequency. That's why in order to get the clear picture of the network symbols emission we need to narrow down the observed period within few hours. The results then are made in a shorter time slots aggregation. In this example on the figures 4.3 and 4.4 the observed period of 120 minutes is broken down in 12 timeslots each of 10 minutes. Thus, for the interval 12:30-12:40 we calculated density index 192 for the symbol 'B', which means there was a sequence of 192 'B'-s in the network words for that particular 10 minutes interval.

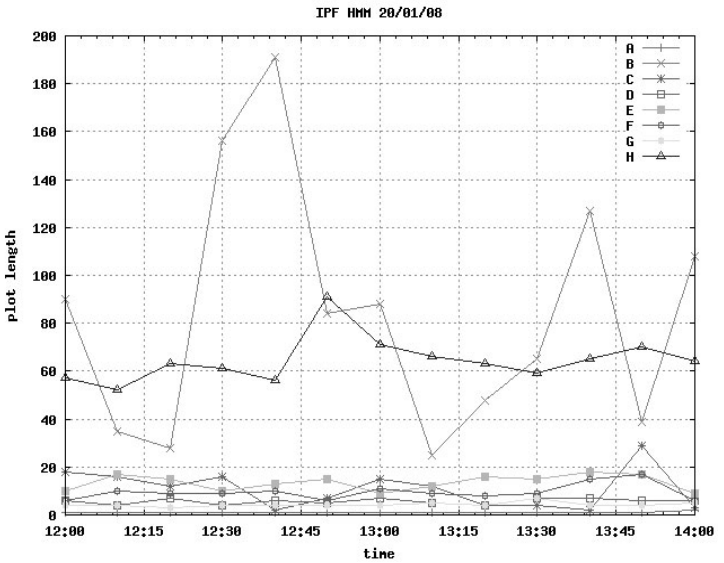


Figure 4.3: Network symbol density during a suspicious flows.

The network alphabet was defined in way to semantically represent the homogeneity grade of the network flows. As many top alphabet symbols (A, B, C) we observe, as more homogeneous the traffic flows are. The opposite is also true. As many symbols from the end of the alphabet we see, as more heterogeneous the traffic is. As we already emphasized, the homogeneous traffic is likely to be machine-generated. Hence, the first alphabet symbols are tokens for suspicious traffic, while the last alphabet symbols are tokens for normal network behavior.

The fairly high density values for the symbol 'B' at figure 4.3 are warnings for suspicious traffic running on the wire. Comparing the values against those captured during the normal traffic flows at figure 4.1 we can see that the densities are fluctuating in some of the time intervals for the symbol 'B'. However, the symbol 'H' again has also high density values, even though the traffic flows were fairly homogeneous. The frequency graphic on figure 4.4 addresses the symbol share rates during the suspicious traffic. The interesting fact is that the

'B'-s went beyond 20%; however, again the 'H' is still the top symbol with frequency rates over 70%. Again, comparing to the corresponding graphic for the normal behavior at figure 4.2 we can easily notice that all the frequencies are below the 20% threshold expect the one of 'H'. Nevertheless in some of the discrete time intervals the frequencies of 'A' and 'B' are fluctuating most probably due to the application level protocol specifics, we could be quite convenient that the critical frequency threshold between the machine generated and the regular traffic flow is around 20% share rate. Saying more simple, if we observe some of the first symbols in the network alphabet (say, 'A', 'B', or 'C') having 20% frequency, the traffic is likely to be machine-generated.

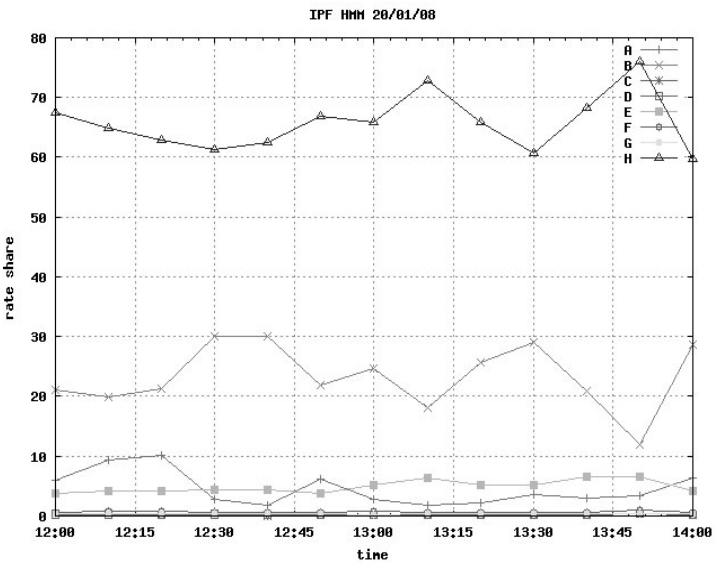


Figure 4.4: Network symbol frequency during a suspicious flows.

To what extend might be helpful the values of the density ρ and the frequency ϕ for the purposes of the traffic analysis? First, those parameters are presenting the entropy grade in a measurable way. The average values for the density of 100 counts for the symbols 'A', 'B', and 'C' together with the frequency rates over 20% will be a token for machine-generated flows and correspondingly for potential malicious network traffic. Table 4.5 summarizes the critical threshold values for the density and the frequency. Those values might be considered when deciding on the traffic flow characteristics.

Symbol density	ρ	100
Symbol frequency	ϕ	20%

Table 4.5: Critical values for ρ and ϕ of the symbols A, B, and C

Once we build and utilize the traffic flow collector together with the analytic engine, we can make network security probes on short time interval basis. Using those probes the tool would be able to generate early alerts for the ongoing suspicious flows. Even though we already defined the critical threshold values for the symbol density and the symbol frequency, it would be even better if the values are tuned up according to the specific network environment.

5 Conclusion

The contemporary network appliance is providing network flow export to the network monitoring systems. The vendors have already implemented the standards for the exported data. Besides the basic network measurements and flow baselines we can utilize that data in a way to build also security analysis. In this paper we have discovered a simple approach how to "extract" the security characteristics of the network flows. We used a Markov model definition for creating a flow model. The model was based on the relation between the previous and the current flow entry exported by the network device. Once the observed symbol definition was made we developed a tool for collecting the data and emitting the network symbols according to the alphabet definitions. The collected network words were analyzed against two characteristics: the symbol density and the symbol frequency previously defined. The results we receive clearly emerged the thresholds between the normal flows and the machine generated flows.

There are two major benefits of that simple flow analysis. First, the network security analyst might be able to easily apply early alerts on the network segments. After the analysis tool is deployed in the critical network segments it can be used for notifying the administrators or the network monitor stations for a suspicious network behavior. The second benefit is the security profiling of the corporate network. The network words could be collected over longer time period and stored for further profile analysis. If we span the observation time over a week, we will be able to profile the flows against the business hours. Respectively, running the observation over a month, quarter, or even a year would build not only the flow profile but also will establish trends measurements for a longer period.

Bibliography

[Cl04] Claise, B.: Cisco Systems NetFlow Services Export Version 9: RFC 3954. Oct 2004.

[Da05] Daigle, J.: Queuing Theory with Applications to Packet Telecommunication. Springer, 2005.

[Ra89] Rabiner, L.: A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. Proceedings of the IEEE, 77 (2), 1989; p. 257–286

Network Forensics of Partial SSL/TLS Encrypted Traffic Classification Using Clustering Algorithms

Meng-Da Wu Stephen D. Wolthusen

Department of Mathematics
Royal Holloway, University of London
Egham, Surrey TW20 0EX
United Kingdom
{M.D.Wu , Stephen.Wolthusen}@rhul.ac.uk

August 9, 2008

Abstract

Machine learning tools have long been used in network traffic analysis, but their application to the network forensics domain and its specific issues has been limited thus far. We investigate the applicability of several common machine learning techniques to identify and classify partial encrypted traffic as may be encountered by forensic investigators confronted only with partial post-hoc traces. It is highly desirable to identify the types of applications and endpoints using such tunnels to facilitate further forensic investigation. In this paper, we therefore examine several clustering algorithms, namely DBSCAN (Density-Based Spatial Clustering of Application with Noise), K-means, and EM (Expectation-Maximization) with regard to their ability to classify encrypted partial traffic using inter-arrival time and TCP length information chosen for its predictive significance. Our experiments demonstrate promising classification results.

1 Introduction

Network traffic analysis has been the basis for a number of application areas, including network design, management, and particularly network security where it is crucial for intrusion detection and prevention. The latter area has therefore seen considerable research devoted to both capturing and subsequent processing and analysis steps.

One particular challenge faced in this is the use of encrypted connections and tunnels in a number of application areas. While it is highly desirable to secure legitimate network traffic in this way, it is not an unmitigated benefit as this generally implies that monitoring will be considerably less effective unless special precautions are taken and e.g. encryption gateways or host-based traffic capturing are used. Even when such mechanisms are available, however,

significant performance issues remain since the capturing and filtering mechanism must provide the aggregate decryption bandwidth of all monitored traffic.

However, the conditions under which network forensics must operate differ in serveral important ways from those found in intrusion detection and prevention. As investigations will typically be post-hoc, the infrastructure of encryption gateways or even host-based monitoring mechanisms cannot be assumed. As a result, any traffic being encrypted by endpoints with key material that is not readily accessible to investigators will be available in encrypted form only. Given the limitations of using pre-shared keys, however, key material will generally be negotiated dynamically in key agreement protocols and hence not be available for subsequent investigation unless suitable precautions have been taken.

Forensic investigations will therefore frequently be confronted with encrypted traffic and must deduce as much as possible from this limited information. Since an increasing amount of legitimate network traffic is also encrypted, this requires discerning between legitimate and illegitimate traffic, both of which may be encrypted using the same protocol such as SSL/TLS or, in some cases, IPSec. Naïve address and port number mapping to identify protocols is insufficient as particularly port numbers are routinely changed, and SSL/TLS tunneling is typically effected through ports assigned to HTTP(S).

The remainder of this paper is organized as follows: Section 2 provides a brief overview of the specimen tunneling protocols (SSL/TLS) analyzed together with a discussion of the machine learning algorithms used for classifying traffic followed by a review of the related work in section 3

Section 4 then briefly describes methodological issues and the design of the experiment subsequently discussed in section 5. The mechanisms used to identify individual protocols in encrypted tunnels are then described in section 6 before providing a discussion of results obtained in section 7 and an outlook on ongoing and future work in section 8.

2 Background

The following section briefly reviews key features relevant for the analysis of tunneled protocols in the SSL/TLS protocol suites in section 2.1 and also provides an overview of the different machine learning algorithms selected to perform the classification of application tunnel behavior.

2.1 The SSL / TLS Protocols

The SSL (Secure Sockets Layer) / TLS (Transport Layer Security) protocol suite is situated above the transport (TCP) layer and provides a number of security services, including traffic encryption, client-side and server-side authentication, and message integrity, which can be combined in a flexible manner to support differing application requirements [Res00]. While it was originally developed to secure connections between web servers and browsers – which is still the most popular application area for the protocol suite – it is now increasingly used to provide security services for other applications and also to

provide general secure network connectivity and not restricted to a single protocol such as HTTP (Hypertext Transfer Protocol). Any upper-layer protocol or application relying on TCP for connection-oriented transport can integrate security services provided by SSL/TLS (e.g. FTP, POP, IMAP).

SSL/TLS is a multilayer protocol consisting of four separate components. The *SSL Record Protocol* formats and encapsulates messages from upper protocols before passing them to TCP layer. The *SSL Handshake Protocol* allows clients and servers to agree on a protocol version, to authenticate both sides, and to negotiate a cipher suite. The *SSL Change Cipher Spec Protocol* message is sent by either the client or the server to notify the receiving party that subsequent content will be protected by the newly negotiated CipherSpec and Keys. Finally, *SSL Alert Protocol* messages are generated when either party to the communication session encounters an error or intends to close a session.

The protocol suite is flexible in its configuration and choice of parameters, and it can be implemented by different requirement, such as server-side only authentication, mutual authentication, or (historically) even using the FORTEZZA handshake.

2.2 Machine Learning Algorithms

The problem of classifying data can be addressed in a number of ways; machine learning provides an attractive approach if training data is available and variations on a given pattern are to be recognized later without requiring manual intervention. Regardless of the concrete algorithms, such solutions usually consist of two processes. One is the classifier construction (learning) phase , and the another one is the actual classification or testing phase where the data sets for each application must be disjoint.

A further distinction can be drawn between supervised and unsupervised techniques. In the latter, clustering algorithms group patterns into different clusters according to feature similarities. This approach does not require prelabeled data sets, and the goal of the algorithm is to extract a number of hidden clusters from an un-labeled data set. In the case of supervised classification, functions are defined when mapping from a previously labeled set of input data (feature) vectors onto a finite set of discrete class labels.

While no plaintext is available for ready classification in case of encrypted tunnels, we posit that sufficient and statistical features are available for inferring clusters and use unsupervised clustering algorithms. To this end, three unsupervised clustering algorithms were chosen to analyze the data sets which are not only encrypted but also incomplete (only capturing one portion of the whole traffic set) typical for forensic analysis. Other research paper regarding to these algorithms can be referred to [BTA+06, BTS06, MLC07, EAM06, BSAS05, EMAW07, MHLB04, EMA06, ZNA05]. The following provides a brief summary of the attributes of each chosen algorithm.

2.2.1 DBSCAN

The Density-Based Spatial Clustering of Application with Noise (DBSCAN) [EKSX96, Han05] algorithm was first proposed to cluster spatial databases by using density-oriented criteria. It is designed to discover clusters of arbitrary shape. It regards clusters as dense areas of objects which are separated by

cluster is considered as noise data. Density-reachability and density-connectivity are the two key concepts to form clusters. For implementing these two concepts, two global parameters are defined: Eps, the maximum radius of the neighborhood; and $MinPts$, giving minmum number if points in an Eps-neighborhood of that point.

Further definitions in the algorithm:

Density Reachability A point p is density reachable from point q if the following two conditions are satisfied:

1. the points are close enough to each other: distance$(p, q) < Eps$
2. there are a sufficient number of points in a q-neighborhood

Density Connectivity A point p is density connected to a point q if there is a point o such that both, p and q, are density reachable from o.

Core Point A point with at least $MinPts$ objects within a Eps-neighborhood.

Border Point A point on the border of a cluster

The DBSCAN algorithm can be sketched as follows:

1. Select a fixed but arbitrary point p
2. Retrieve all points density-reachable from p with respect to Eps and $MinPts$.
3. If p is a core point, a cluster is formed.
4. If p is a border point, no points are density-reachable from p and DBSCAN visits the next point of the database.
5. Continue the process until all of the points have been processed.

2.2.2 K-Means

The k-means algorithm [HW79, Han05] is a partitioning algorithm. This kind of algorithm organizes all objects into k partitions where each partition represents a cluster. The k-means algorithm is the simplest one to perform clustering testing.

The algorithm takes the input parameter k and partitions a set of n objects into k clusters. Cluster similarity is measured with regard to the mean value of the objects in a cluster, which can be viewed as the cluster's center. For each cluster, the clustering algorithm maximizes the homogeneity within the cluster by minimizing the square-error, i.e. the process iterates until the criterion function converges. Typically, the square-error criterion is used, defined as

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} |p - m_i|^2$$

where E is the sum of the square error for all objects in the data set; p is the point in space representing a given object; and m_i is the mean of cluster C_i. For

each object of each cluster, the distance from the object to its cluster is squared, and the distances are summed. The k-means procedures are summarized as the following:

Input k, the number of clusters; D, a data set containing n objects

Output A set of k clusters

Procedures 1. Arbitrarily choose k objects from D as the initial cluster centers

2. Iterate the following until the clusters are stable:

   $\Rightarrow$ (re)assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster

   $\Rightarrow$ update the cluster means, i.e. calculate the mean value of the objects for each cluster;

2.2.3 EM

The final algorithm considered here is the EM (Expectation-Maximization) algorithm [Han05, DLR77]. It is typically used to perform maximum likelihood estimation of parameters in probabilistic models. By using the EM clustering technique, each cluster can be presented as a parametric probability distribution.

It assumes that the entire data set can be treated as a mixture of underlying probability distributions, which is typically referred to as a component distribution. This technique can cluster data sets by using a finite mixture density model of k probability distributions, where each distribution represents a cluster. It is aiming to find the best fit of the entire data by estimating the parameters of probability distributions. EM is a iterative refinement algorithm to estimate parameter by assigning each object to a cluster according to a weight representing the probability of membership. There are no strict boundaries between clusters, and new means are computed based on the specification of weighted measures. The algorithm is described as the following steps:

1. Provide an initial estimation of the parameter vector

2. Iteratively refine the parameters by using the following two steps:

   $\Rightarrow$ Expectation Step: Assign each object x_i to cluster C_k with the probability

   $$P(x_i \in C_k) = p(C_k \mid x_i) = \frac{p(C_k)p(x_i \mid C_k)}{p(x_i)},$$

   where $p(xi \mid C_k) = N(m_k, Ek(x_i))$ follows the normal distribution around the mean, m_k with expectation E_k. This step calculates the probability of cluster membership of object x_i for each of the clusters. These probability are the expected cluster memberships for object x_i.

   $\Rightarrow$ Maximization Step: Use the probability estimates from above to re-estimate the model parameters, i.e.

$$m_k = \frac{1}{n} \sum_{i=1}^{n} \frac{x_i P(x_i \in C_k)}{\Sigma_j P(x_i \in C_j)}$$

3 Related Work

In this paper, we examined three unsupervised clustering algorithms of machine learning techniques: DBSCAN(Density-Based Spatial Clustering of Application with Noise), K-means, and EM(Expectation-Maximization). These techniques have been applied to network traffic analysis domain in many literatures. K-means is the most straightforward clustering method in traffic classification[BTA+06, BTS06, MLC07, EAM06, BSAS05, EMAW07]. DBSCAN clustering technique was used to cluster network traffic for accurately identify applications[EAM06]. The EM applications in network traffic were examined and discussed in [MHLB04, EAM06, EMA06, ZNA05]. Further reference of clustering technique survey can refer to [XW05]. Other machine learning techniques also demonstrated promising results in network traffic analysis, such as Hidden Markov Model[WMM06, BTS06, BSAS05], Baysian classifier[ZNA05], and K-Nearest Neighbor[WMM06]. On the specific analysis application of SSL/TLS tunnel traffic, [BLJL06, SSW+02] used statistics techniques to analyze HTTPS traffic. [WW07] exploited the information leakage of SSL/TLS sequential headers for classifying software implementations.

In the network forensics domain, research literatures are good reference when performing practical analysis. [RJ05] modeled network forensics mechanism, and [EPF06] proposed network forensic readiness development life cycle. Both of them help to provide ont only a network forensic sound methodology but also a comprehensive overview. [WD05] proposed to build evidence graphs for enhancing network forensics analysis. The network uncertainty could lead to forensic investigation errors and should be concerned. This uncertainty of network evidence were discussed in [Cas02], such as data corruption, loss, tampering, or errors. More practical concerns and comparisons of choosing network forensics tools and functions can be found in [Cas04]. Improvements of network evidence collection are discussed in [Nik06].

4 Methodology of applying clustering algorithms

We employed standard classification mechanisms to classify partial traffic, obtaining n cluster descriptions per algorithm, which are described as clustering centers in the following discussion. Following the learning phase, each algorithm processed a specific clustering composition.

In the testing phase, each individual partial traffic set is first clustered by the clustering algorithms, yielding a candidate clustering composition. This candidate composition is then classified (compared) to all other pre-defined clustering compositions trained in the learning phase.

Our classifier is based on a simple Euclidean distance metric for candidate clustering centers X_i and pre-defined clustering centers Y_i as

Software Category	Server Side(software version)	Client Side(software version)
(A) File Transfer Protocol	TitanFTP (5.34)	CuteFTP (6.0)
(B) Real Time Streaming Protocol	VLC Media player (0.8.6c)	VLC Media player (0.8.6c)
(C) Remote Framebuffer Protocol	UltraVNC server (1.0.2)	UltraVNC viewer(1.0.2)

Table 1: Software Speciment using SSL/TLS

$$Dist(X,Y) = \sqrt{\sum_{i=1}^{n}(X_i - Y_i)^2}$$

where Y_i is given by one of the algorithms described above. The $Dist$ is computed by n clustering centers produced by different clustering algorithms and parameters.

5 Application classification using partial encrypted traffic

In the following section, the experimental methodology is briefly described, followed by the results of the experiments.

5.1 Experiment Scenario

Our mechanism employs *partial traffic* to classify network protocols which is encrypted by SSL/TLS. This is owing to the expectation that in a forensic application, it will be unlikely to have full traces available. We therefore chose 1000 continuous IP packets between two specific ports on a pair of machines as our defined *partial traffic* set. In the feature space, only the *inter-arrival time* feature is extracted and tested in this section.

There are 2 hosts in our scenario, and we extracted all parameters of their operation systems for the examining purpose. These parameters are highly critical because slightly changing OS parameters, such as TCP/IP stack, could lead to different traffic patterns. In such scenario, it is our task of network forensic investigations to classify network protocols by using partial traffic only.

5.2 Data Set Preparation

As this is a preliminary experiment, each protocol's behavior presented here is extracted from one specific software implementation. Three protocols are chosen for testing our mechanism. The overall data variance of *inter-arrival time* feature in our lab is: *minimum*, 0.000001; *maximum*, 1.023213; *mean*, 0.003838. The data variance is based on such experiment scenario that the traffic we produced gets through a intermediary hub device where we collected all our traffic. The permutations of software package specimens used in the classification experiment are detailed in table 1.

Table 1 is the overall software[1] information used for traffic generation. The operation system of both server and client side is *Windows XP Pro SP2 with default TCP/IP parameters*, and the SSL/TLS tunnel software is Barracuda HTTPS [2].

There are two data sets prepared for performing our mechanism: learning data set and testing data set. Learning data set is used to build the cluster composition of each specific software. The testing data set is for evaluating the accuracy of our classification mechanism. In real forensic cases, investigators can produce the learning data sets as many as necessary to guarantee sufficient data for forming reliable cluster compositions. In our experiments, 30 individual traffic sets as learning data are collected and promising classification results come out in later stage.

Both of the learning and testing data sets include 30 individual traffic sets for testing our mechanism. Parameters of the operation systems are the same during the data collection period, and they can be extracted from target machines by using computer forensics software, such as *EnCase* [3]. Depending on these known parameters, we can design our mechanism without building a table referring to different pattern descriptions based on different parameter settings. Such table can be set up in a more complicated case investigation.

5.3 Learning Phase

According to our methodology, the core concept of learning phase is to compute the clustering composition of each software. Each clustering composition includes n clustering centers(clustering descriptions). In this phase, firstly we used unsupervised clustering algorithms to generate n clustering centers. We then computed the means of clustering centers from 30 training data sets. Afterward, these means are used to form clustering compositions. Such clustering compositions are used as classifier information in later stage of traffic classification.

5.4 Testing Phase and Results

Every partial traffic set (1000 continuous IP packets) in the testing phase is clustered and processed to form a candidate clustering composition. Each candidate composition is then classified (compared) to all other predefined clustering compositions trained in the learning phase. All experimental results are tested by *R software* [4], including DBSCAN, K-means, and Mclust[FR06] packages. Classification results are presented in the following sections:

[1] Further information of our testing software:
[TitanFTP]`http://www.webdrive.com/`
[CuteFTP]`http://www.cuteftp.com/`
[VLCMediaPlayer]`http://www.videolan.org/`
[UltraVNC]`http://www.uvnc.com/`

[2] See`http://barracudaserver.com/products/BarracudaDrive/HttpsTunnel.lsp` for details.

[3] See `http://www.guidancesoftware.com` for details.

[4] See `http://www.r-project.org/` for details.

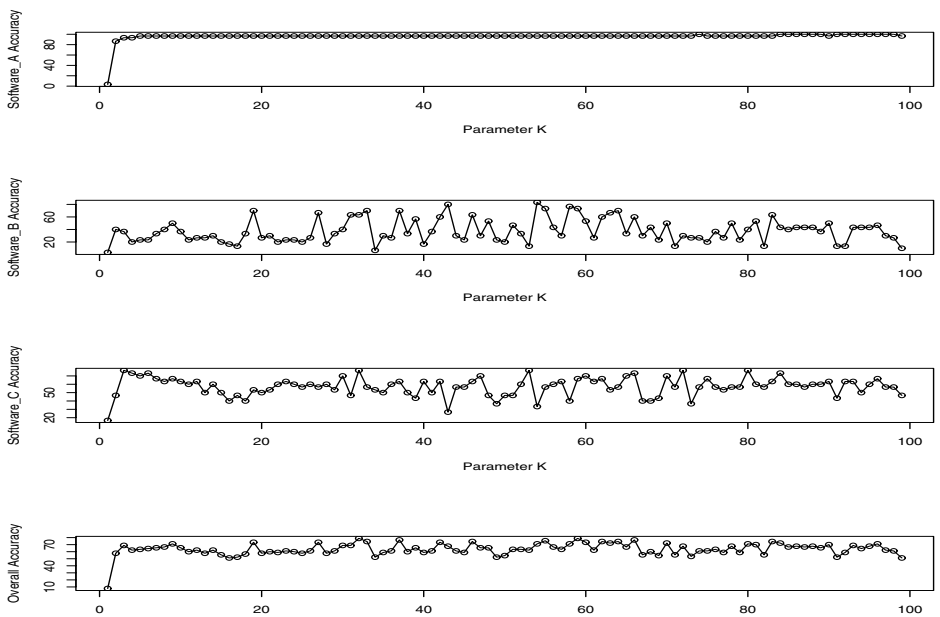


Figure 1: K-Means Accuracy

5.4.1 DBSCAN

For performing this clustering algorithm, two global parameters are required: $MinPts$ and Eps. These two parameters can vary in a great scale, but we demonstrated DBSCAN's strong capability in solving our defined problem by only choosing few parameter combinations. We chose the following parameter combinations of $MinPts/Eps$: 50/0.00005, 80/0.00008, and 110/0.00011. All of them have the same overall 99% accuracy in traffic classification.

The experimental results are generated by randomly choosing parameter combinations for DBSCAN. However, we did not really accept any parameter combinations, and we filtered them by an important factor: *we accept parameters only when they can generate stable clustering numbers.* In our observations, some parameter combinations generate unstable clustering numbers that could lead to difficulties in accurately describing data patterns. Unstable clustering numbers mean unreliable clustering separations. DBSCAN with unstable clustering numbers can not properly perform clustering description jobs, and we have to dismiss this kind of parameter combinations.

5.4.2 K-means

The K-means algorithm has only one input parameter: K. This input parameter presents how many clusters will be computed by its mathematical model. We tested the parameter from 2 to 100, and these experimental results are pre-

sented in Figure 1.

In this test,the overall average accuracy is 63%, and the best accuracy among all parameters is 79%. In the initial parameter 2, the accuracy is just 8%. After the parameter 2, the accuracy jumps to 58%. Most points of accuracy are in the range of 50%-75%. Even though the optimum accuracy can reach 79%, we can still see an unstable accuracy within all parameters. However, by separately observing each software's classification results, the accuracy of A software can reach 97% after parameter 5. It means K-means can work very well in describing A software but fail to cluster the other two software behaviors. We could exploit this feature to develop another statistical mechanism for accurately identifying A software, but it is out of the scope of this paper. The purpose of this paper is to find a overall optimum algorithm in solving general *partial traffic classification* problem.

5.4.3 EM

In the EM algorithm, only one parameter G is required to perform its mathematics model. G presents how many clusters will be computed based on probability distributions. This part, we tested the parameter G from 2 to 30. All experimental results can be found in Figure 2 (box-and-whisker plot). The best accuracy is parameter 19 with 91%. After the parameter 24, the EM capability in describing clustering goes down sharply. From the Figure 2, we can identify that the parameter range between 18 to 22 can generate better average accuracy of 89%. These data are good reference when performing real case investigations.

6 Protocol identification under SSL/TLS tunnel technology using DBSCAN algorithm

In section 5, we performed three different clustering algorithms to classify protocols, including FTP, real time streaming protocol, and remote framebuffer protocol. The strength of using DBSCAN to identify protocol behaviors were preliminary confirmed in section 5, but we used only specific software pair of both client and server sides with single feature. More experiments should be performed to estimate how DBSCAN can help forensic investigators to identify protocol patterns. Based on DBSCAN, we performed more experiments and added more features in this section to present the application of using DBSCAN clustering algorithms and conditional probability to identify protocol behaviors.

6.1 Experiment set up

In this section, we tested three popular network protocols, including File Transfer Protocol(FTP), Hypertext Transmission Protocol(HTTP), Real Time Streaming Protocol, and one malicious HTTP performed by Agobot[5]. All experimental data is produced and collected as the same mechanism described in section

[5]A typical implementation of botnet software, See`http://www.symantec.com/security_response/writeup.jsp?docid=2004-051816-5418-99` for details.

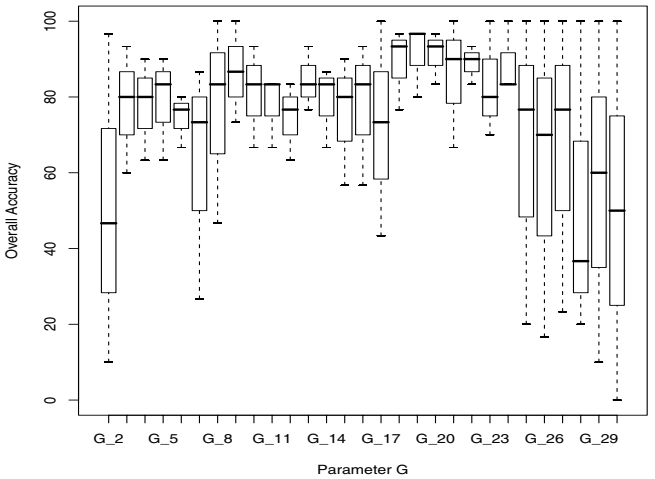


Figure 2: Mclust Accuracy

5. All traffic is encrypted by SSL/TLS technology and only partial traffic is captured for testing our mechanism. Table 2 is the overall software[6] information used for traffic generation.

We used more combinations of both client and sever sides on each protocol to demonstrate our mechanism's capability for dealing with the diversity of protocol behaviors. For each protocol, we chose three different client applications with the same server to have representative protocol traffic sets. In our experiment, we have full control of Agobot, and we commanded our bot to download files from our web server by using HTTP mechanism. Such malicious HTTP traffic is collected during file transfer duration.

We randomly chose one specific software implementation of each protocol to produce 30 learning data sets.

As we use three different client applications communicating with the same server, 90 testing data sets are collected on each protocol. In the malicious traffic set, only 30 learning data sets and 30 testing data sets has been collected because we only used Agobot as our client.

[6]Further information of our testing software:
[TitanFTP]http://www.webdrive.com/
[FTP Voyager]http://www.ftpvoyager.com/
[Classic FTP]http://www.nchsoftware.com/classic/index.html
[EFTP]http://www.encrypted-ftp.com/
[VLC Media Player]http://www.videolan.org/
[Windows Media Player]http://www.windowsmedia.com
[Kantaris]http://www.kantaris.org/
[Mozilla Firefox]http://www.mozilla.com/
[Opera]http://www.opera.com/
[IE]http://www.microsoft.com/windows/products/winfamily/ie/default.mspx

Software Category	Server Side(software version)	Client Side(software version)
(A) File Transfer Protocol	TitanFTP (5.34)	FTP Voyager (15.0.0.1) Classic FTP (1.02) EFTP (3.3.1.145)
(B) Real Time Streaming Protocol	VLC Media player (0.8.6c)	Windows Media Player (9.00.00.3250) VLC Media player (0.8.6c) Kantaris (0.3.7)
(C) HTTP	Apache (1.3.34) on Linux (2.6.13.2)	Mozilla Firefox (2.0.0.14) Opera (9.27) IE (6.0)
(D) Malicious HTTP	Apache (1.3.34) on Linux (2.6.13.2)	Agobot

Table 2: Software Specimens using SSL/TLS

6.2 Feature collection

Only *inter-arrival time* feature is extracted in section 5. This section, we enrich our feature space by adding another 2 TCP level features. These new TCP level features are calculated from the inter-arrival time feature which is produced by the same mechanism introduced in section 5. As DBSCAN can form several clustering groups by using inter-arrival time information, these separated groups can be used as conditions to produce our TCP level conditional probability features.

The first feature is to use the method in section 5: DBSCAN is performed by inter-arrival time information. Afterward, clustering results are presented as separated timing groups. The second feature is to extract the conditional probability of TCP length information. Such TCP information are pre-grouped by using clustering criterion from feature 1.

The second feature of conditional probability in TCP length information can lead to our third feature used in this section. There are several clustering groups of TCP length information found by calculating conditional probability feature, and it can be used as our third feature information. We chose the highest and second frequent group of TCP length information as our third feature. In some cases, however, only one TCP length group can be found. We have to dismiss TCP length information which appears just randomly and is not strong enough to form a group. We set up a threshold of 5% proportion of overall TCP length points for choosing TCP length group.

6.3 Experimental results

We use Euclidean distance metric for classifying our testing data sets. One inter-arrival time feature and two TCP length information features are used to calculate the overall feature distance. In such distance metric, a promising 97.5% accuracy is presented from our experiments. Another interesting result should be pointed out that DBSCAN not only classified traffic protocol but also detected protocol violation. HTTP performed by malicious bot, named Agobot, has distinct clustering information compared to normal HTTP browsing traffic.

7 Results and Discussions

All the experimental results are from our network forensic lab environment, and therefore this paper should be treated as a preliminary survey of clustering algorithms in helping network forensic investigations. Protocol traffic coming from different software implementations is tested to explore the capability of clustering algorithms in solving partial traffic identification problem.

In section 5, three clustering algorithms are applied to identify three protocol implementations. The EM algorithm can have a overall 89% accuracy within the specific parameter rage. Thus, EM could be reliable to cluster inter-arrival time information after precisely experimental testing and setting. However, it would need more refinement for helping classification of protocol applications. Our K-means algorithm is a very basic version of implementation and does not deal with noise data. From the testing results, K-means failed to solve the partial traffic identification problem. By comparing the other 2 algorithms, DBSCAN presented highly satisfied capability in discovering traffic patterns. DBSCAN demonstrated an amazing 99% accuracy of software classification in section 5.

The section 6 is to explore DBSCAN's application in partial traffic identification problem by adding more features. The advantage of DBSCAN is to extract *density-based* clustering information, and it works very well in discovering clustering information in our experiments. In section 6, it successfully extracted clusters of inter-arrival time and TCP length features. Moreover, the density-based algorithm can also automatically dismiss other noise data which is not within the clustering areas. The high accurate results show that good clustering descriptions can be formed by DBSCAN when choosing appropriate features. Such experimental results lead to the concept that network protocol implementation is a reliable subject of applying density-based clustering algorithm.

We used one specific implementation of SSL/TLS software as our tunnel technology in section 5 and 6, and all later experimental results should be considered in such circumstance. The purpose of this paper is to explore appropriate tools for extracting traffic patterns of communication behaviors. The software of SSL/TLS technology is implemented by its own specifications, and such distinct specifications are presented as different traffic patterns. Based on our experimental testing in section 5 and 6, network evidence examiner can rely on clustering algorithm, such as DBSCAN, for dealing with certain cases when highly sophisticated techniques are needed.

8 Conclusion and Future Work

Machine learning techniques have long been the subject of research in the network traffic analysis domain, but network forensics, particularly for encrypted traffic have specific requirements and have thus far not been investigated extensively. The main applications of network analysis are to classify traffic protocols by using full traffic data set with multiple features. Most literature in the domain aims to improve the accuracy or performance for such general traffic classification problems. Motivated by these papers, we believe machine learning applications are promising tools to enhance network forensics inves-

tigations with their specific application requirements such as having only partial data at its disposal along with noisy data sets. The use of unsupervised machine learning applications represent an efficient mechanism for classifying network traffic in such a way as to allow further investigation, potentially at the host level. To this end, we explored clustering algorithms in network forensics applications and, following promising initial results reported in this paper, intend to investigate further refinement of encrypted network traffic behavior, particularly in more noisy environments than those found in the laboratory environment used for the experiments reported here. Similarly, as the training data sets were generated in the same environment as the clustering data, network uncertainties can be assumed to be quite similar — this is not necessarily the case for normal forensic investigations. We are therefore investigating the influence of network uncertainty issues in further work, particularly fragmentation, fragment ordering, fragmentation timeouts, header errors, checksum errors, and duplicate packets at the IP level and duplicate segments, partially duplicate segments, segments arriving before or after the receive window, duplicate acknowledgments, and out-of-order segments for the TCP layer. At the algorithmic level, we will conduct further research on interactions between protocol layers and multivariate analysis of packet information.

References

[BLJL06] George Dean Bissias, Marc Liberatore, David Jensen, and Brian Neil Levine. Privacy Vulnerabilities in Encrypted HTTP Streams . In *Privacy Enhancing Technologies*, Lecture Notes in Computer Science, pages 1–11. Springer Berlin , June 2006.

[BSAS05] Laurent Bernaille, Augustin Soule, Ismael Akodjenou, and Kave Salamatian. Blind Application Recognition Through Behavioral Classification. Technical report, Networks and Performance Analysis Group, February, 2005.

[BTA+06] Laurent Bernaille, Renata Teixeira, Ismael Akodkenou, Augustin Soule, and Kave Salamatian. Traffic Classification on the Fly. *SIGCOMM Comput. Commun. Rev.*, 36(2):23–26, 2006.

[BTS06] Laurent Bernaille, Renata Teixeira, and Kavé Salamatian. Early Application Identification. Conference on Future Networking Technologies. In *CONEXT*, 2006.

[Cas02] Eoghan Casey. Error, Uncertainty and Loss in Digital Evidence . *International Journal of Digital Evidence* , (2), 2002.

[Cas04] Eoghan Casey. Network Traffic as A Source of Evidence: Tool Strengths, Weaknesses, and Future Needs. *Digital Investigation*, 1(1):28–43, 2004.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Royal Statistical Society*, pages 1–38, 1977.

[EAM06] Jeffrey Erman, Martin Arlitt, and Anirban Mahanti. Traffic Classification using Clustering Algorithms. In *MineNet '06: Proceedings of the 2006 SIGCOMM Workshop on Mining Network Data*, pages 281–286, New York, NY, USA, 2006. ACM.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD96)*, pages 226–231, 1996. `http://ifsc.ualr.edu/xwxu/publications/kdd-96.pdf`.

[EMA06] Jeffrey Erman, Anirban Mahanti, and Martin Arlitt. Internet Traffic Identification using Machine Learning. *Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE*, pages 1–6, Nov. 2006.

[EMAW07] Jeffrey Erman, Anirban Mahanti, Martin Arlitt, and Carey Williamson. Identifying and Discriminating Between Web and Peer-to-Peer Traffic in the Network Core. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 883–892, New York, NY, USA, 2007. ACM.

[EPF06] B.E. Endicott-Popovsky and D.A. Frincke. Embedding Forensic Capabilities into Networks: Addressing Inefficiencies in Digital Forensics Investigations. *Information Assurance Workshop, 2006 IEEE*, pages 133–139, 21-23 June 2006.

[FR06] Chris Fraley and Adrian E. Raftery. MCLUST Version 3 for R: Normal Mixture Modeling and Model-Based Clustering . Technical Report 504, Department of Statistics, University of Washington , 2006.

[Han05] Jiawei Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[HW79] J. A. Hartigan and M. A. Wong. A K-Means Clustering Algorithm. *Applied Statistics*, 28:100–108, 1979.

[MHLB04] Anthony McGregor, Mark Hall, Perry Lorier, and James Brunskill. Flow Clustering Using Machine Learning Techniques. In *Passive and Active Network Measurement*, Lecture Notes in Computer Science, pages 205–214. Springer Berlin , May 2004.

[MLC07] Gerhard Mu"nz, Sa Li, and Georg Carle. Traffic Anomaly Detection Using K-Means Clustering. In *Proc. of Leistungs-, Zuverla"ssigkeits- und Verla"sslichkeitsbewertung von Kommunikationsnetzen und Verteilten Systemen, 4. GI/ITG-Workshop MMBnet 2007*, Hamburg, Germany, September 2007.

[Nik06] Bruce J. Nikkel. Improving Evidence Acquisition from Live Network Sources. *Digital Investigation*, 3(2):89–96, 2006.

[Res00] Eric Rescorla. *SSL and TLS: Designing and Building Secure Systems*. Addison-Wesley, Reading, MA, USA, 2000.

[RJ05] Wei Ren and Hai Jin. Modeling the Network Forensics Behaviors. *Security and Privacy for Emerging Areas in Communication Networks, 2005. Workshop of the 1st International Conference on*, pages 1–8, 5-9 Sept. 2005.

[SSW+02] Qixiang Sun, Daniel R. Simon, Yi-Min Wang, Wilf Russell, Venkata N. Padmanabhan, and Lili Qiu. Statistical Identification of Encrypted Web Browsing Traffic. In *SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy*, page 19, Washington, DC, USA, 2002. IEEE Computer Society.

[WD05] Wei Wang and T.E. Daniels. Building Evidence Graphs for Network Forensics Analysis. *Computer Security Applications Conference, 21st Annual*, pages 11 pp.–, 5-9 Dec. 2005.

[WMM06] Charles V. Wright, Fabian Monrose, and Gerald M. Masson. On Inferring Application Protocol Behaviors in Encrypted Network . In *The Journal of Machine Learning Research* , pages 2745–2769. Microtome Publishing, December 2006.

[WW07] Meng-Da Wu and Stephen Wolthusen. Network Forensics of SSL/TLS Encrypted Channels . *ECIW 2007. The 6th European Conference on Information Warfare and Security.* , pages 303–312, 2007.

[XW05] Rui Xu and II Wunsch, D. Survey of Clustering Algorithms. *Neural Networks, IEEE Transactions on*, 16(3):645–678, May 2005.

[ZNA05] S. Zander, T. Nguyen, and G. Armitage. Automated Traffic Classification and Application Identification using Machine Learning. *Local Computer Networks, 2005. 30th Anniversary. The IEEE Conference on*, pages 250–257, 15-17 Nov. 2005.

Building a runtime state tracing kernel

Ananth Chakravarthy, Vinay G. Vaidya

Symbiosis Deemed University
Senapati Bapat marg,
Pune, India

chakri.ananth@gmail.com
vaidya.vinay@gmail.com

Abstract: A process is run by executing a sequence of instructions by the processor. However it is probable that not all of the instructions are executed as there are hundreds of paths that can be taken by the executable to complete its execution. The path chosen is dependent on a host of factors like the environment, user input, the platform etc. As such, at any given instant of time, the process might be in any of the possible states S^n after traversing states $S^1,S^2,S^3..$ where S^1,S^2,S^3 ....S^n,S^{n+1}, $S^{n+2},..S^M$ depict the total M states that can be taken by the executable. There is no mechanism currently inside the Linux kernel to peek into the state of the process to find out which if these states is it currently in and what are the states it has "traversed" to reach the current state while it is executing. If such an effective tracing can be achieved, it would lead to better operating system security. Other advantages are better logs or even building a verifiable software system. This paper looks at the infrastructure that has been developed to realize such a functionality in the Linux kernel and thereby increase the security of the running process. Of particular mention is the framework that has been developed to peek into the state of a running process as it executes and the various mechanisms that could be used to ascertain the state of the running process.

Tags: Security, State tracing, ELF, Reverse Engineering, Linux, System Calls, dynamorio

1 Introduction

There are a host of mechanisms in which security is compromised in the organizations today. Buffer overflow attacks, viruses, trojans, worms, rootkits, social engineering, spy-ware or hacking tools like key-loggers, password rippers, net-cat kind of utilities are only some of the plethora of options that are available today to break into a secure system. While solutions do exist to counter each of these kind of attacks, most of them do not seem to really counter the attack even before it has actually taken for the first time. For example, an anti-virus cannot detect a new virus until it obtains the signature of the virus. All of the drawbacks for the current security tools point out the need for a security framework using which the working of an executable can be verified against some sort of description of the system or the binary which is getting executed. Such a proactive verification of the working of an executable would go a long way in providing the requisite capability for the operating system to monitor and thwart any malicious attempts before the system is compromised. One of the first and foremost feature of such a security framework is to have the capability to find out what is the process state at any given instant of time. The approach taken by the author to counter these attacks is to provide a description of the binary to the kernel so that while the binary is getting executed, the Linux kernel verifies the state of the running binary. If the state does not match to a valid state as described in the description document, then appropriate logic could be triggered from the kernel. The following paper is an attempt to propose and discuss a mechanism to enable process state capturing and querying while the process is getting executed on the operating system.

2 Problem Description

Problem statement

The problem that the paper is trying to solve is "There are no operating system modules inside the Linux kernel that help in effectively tracing the current state of a running process and the path of its state traversal to its current state".

Enabling such a feature in the operating system would help in taking proactive steps in case of suspicious activity. The first step towards building such a self-protecting operating system model is to make the operating system aware of the various states that the binary can traverse while it is executing. Hence it becomes imperative that the operating system modules possess a mechanism of running an executable on a "state aware" basis.

To build a modified Linux kernel that can provide interfaces about the current state of the process to the external world as well other kernel modules is the subject of this paper.

Before trying to build a state aware Linux kernel, the concept of a "state" needs to be defined. The definition of a state could be as simple as beginning of a function entry to the point of exit of that function. Another approach is to define a state as a sequence of instructions which implement a functionality, for example opening a file and writing something to the opened file. However, the current approach in this paper is to define a "state" as a sequence of machine instructions which do not have a "call" or "jmp" category of instructions.

3 Literature Review

Though no direct references were found in Literature to build a state verification Linux Kernel, there are a large number of references with respect to state tracing. Below are some of the references found with respect to tracing.

Currently applications rely on the logging mechanisms to trace the flow of execution. The developers write code as part of the application log to generate log messages to a predetermined log file. The log file is then later analyzed mostly manually as the messages requires human intelligence to interpret the state flow. Another drawback with this approach is that extensive logging commands need to be written by the developer to trace the full state of the application. Moreover, the application would not be able to detect the changes in the flow of the application state flow. For example, consider the following hypothetical state flow from state S^m to S^n . Suppose the developer has written log messages which ascertain the state traversal for both of these states. If this executable has been infected with a virus which introduces a jump instruction at the end of the state S^m to the malicious code and at the end of the malicious code transfers the control to the state S^n. This may be depicted as follows:

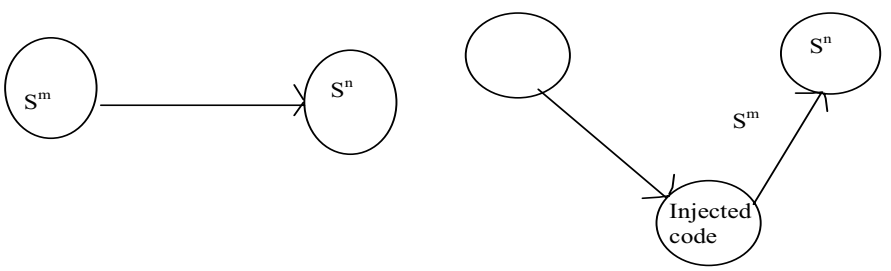


Figure 3.1 showing the normal state flow to the left and the infected state flow to the right

The issue with this approach is that the logs still show what the analyzer is expecting to see and nothing seems to be wrong as per the logs. Another drawback is that the state tracing by logging only works only if we have the source code for the binary for which the analysis is being done.

Linux Trace toolkit (ltt) is an open-source project which involves patching up the standard kernel to introduce changes to keep accounting of various aspects like kernel locks, files open and the network interfaces. Below is a snapshot of the tool.

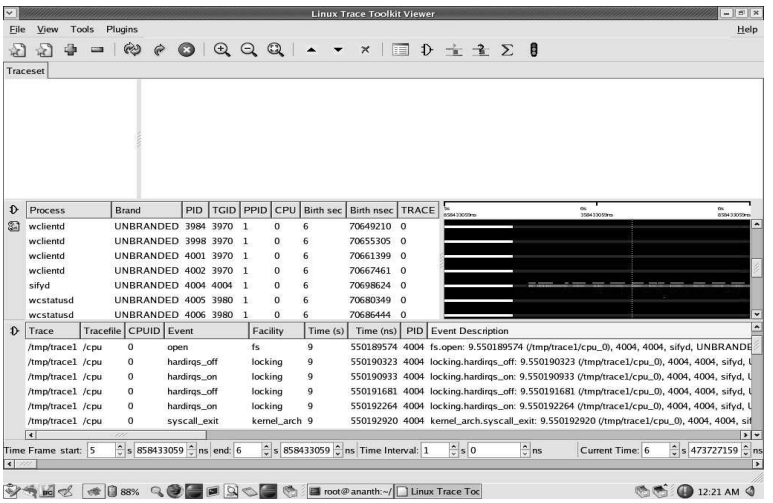


Figure 3.2: snapshot of the LttViewer tool

As the figure depicts, this tool helps in keeping a track of the various resources available in the kernel like the amount spent in the kernel, the number of times the lock was acquired. The downside of this approach is that it ignores the state of the binary in the user space completely. This logic kicks in mostly when the application has issued a system call. However, the tracing mechanism is in the hidden state when the binary is executing in the user space. However, it has a a very novel way of describing facilities, that is declaring new states, that need to be traced. This approach can come in quite handy when we are developing applications from the scratch and want to trace new states as decided by the application developer. The entire module is geared towards logging the state of the kernel and not any particular state identification in the kernel. The other drawback with this approach (for that matter any source code based state tracing approach) is that it still suffers from the redirection problem that is mentioned in Figure 3.1 as given above.

Ptrace approach is another mechanism which is used to trace the binary by making ptrace system calls. This approach involves an attempt by the developer to make ptrace family of system calls which essentially introduces a single step instruction interrupt handler for the binary which can then inject the code to make further calls to read/write the data from particular memory locations. Core dump analysis is another approach to peek into the state of the binary which generated the core dump just before crashing. This approach is mostly useful take a post-mortem approach for analyzing the state of the system after the binary execution crashed and hence cannot be used for effective protection mechanisms.

The approach taken by Yashushi and Sejichi in their paper [10] involves single stepping every instruction in the kernel is not a practical approach. This is be cause single stepping for each and every instruction will prove to be very costly in terms of because it involves having separate hardware which can hold the trace data information. The amounts of data that is generated when this approach is taken is too much to be handled at runtime. It needs to be noted that the amount of time that it takes to analyze and conclude that the binary is in a particular state needs to be completely optimized for performance.

In the paper Dynamically discovering likely program invariants by Michael [11], [12], the approach involves discovering the program state by using the state variables and their values at runtime. The kernel space is neglected in this approach and primarily concentrates on the state of the running program as expected before hand. For example, it can validate the values of certain variables in the binary by running assert statements.

From the work done in the paper Speeding up of synthesis from program traces[13], points out some mechanisms which could be used to build up a program by tracing the instructions while executing the program. Though the work is not directly related to tracing, it provides some insights into identifying the concepts of states and state transitions.

From the work done in the paper, Object oriented Program Tracing and visualization [14], the author identifies the all important state explosion problem and proposes the mechanisms like Pruning, Merging and Slicing to deal with the problem of tracing object-oriented programs.

From the work done in the paper Classification of anomalous traces of privileged and parallel programs by neural networks[15], the authors present the concept of tracing the process state whenever there is a system call. This approach completely ignores the user space execution.

4 Architecture of the runtime state tracing Linux

As seen earlier, one of the primary requisites for building a kernel which can monitor the execution of the binary requires that the control is transferred to the kernel space at frequent points during execution. The current design of the operating system permits the control to be transferred to the kernel only when there is a system call triggered from the user space. The following flow chart gives the current flow for the currently available Linux kernel.

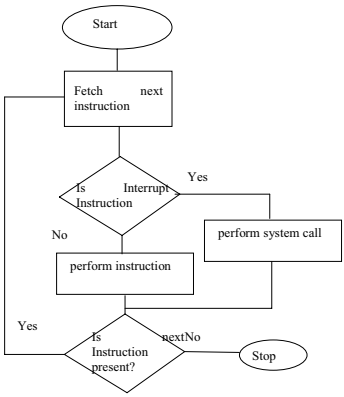


Figure 4.0.a Flow chart giving the current flow of instructions for a binary

The primary requisite for the kernel to track the binary is that it should get the control to determine the current state of the binary. Another requirement is that the mechanism to trigger the context switch to the kernel should be independent of the binary code that is getting loaded as a process. This is to avoid the problem described in Figure 3.1 as there is a risk that the context switch code itself is circumvented by the malicious code if the original binary which has the context switch code is infected with malicious code stubs. Henceforth, the intermediate code that invokes the context switch code is referred to as the interpreter code. Another requirement is that the interpreter that keeps interpreting the binary as it is getting executed needs to define what is a "state" and that definition needs to be congruent as to what the kernel expects when the process is executing in the kernel space. Along with the state definition, tools need to be developed to identify all the states that a binary can be associated with, and this state information if passed to the kernel while the process is about to execute would result in the kernel verifying the state of the process as it executes. Thus a requirement arises that the kernel has a mechanism to peek into the various aspects of the process when the context switch happens to the kernel space as the process keeps executing. Summarizing, here are the changes that need to be done in the kernel and the set of the tools that need to be developed to realize the entire workflow.

- Tool to reverse engineer a binary to identify the complete set of states
- Tool to identify what are the characteristics for each of the states identified in the above step.
- An interpreter which keeps triggering the kernel verification code whenever there is a state transition.
- A modified kernel that accepts calls from the interpreter and verify the state transitions
- A mechanism inside the kernel to verify various aspects of the running process

The following flow chart describes modified flow of control for the proposed Linux kernel and its execution model. The framework involves launching the binary through an interpreter which would fetch blocks of code called as cache, and then fetch the code which hereby is referred to as the injected code in the form of a library. Then both the cache and the code to be injected are executed. The interpreter then saves the context of running binary, executes instructions which form part of the

interpreter itself. It may be noted that the interpreter and the running binary have their own stacks. The interpreter then calls the Linux kernel to pass information like the address of the last instruction that got executed when the context switch was happening from the running process to the interpreter context. The modified Linux kernel then does some verification based on the information passed by the interpreter and also does some book keeping.

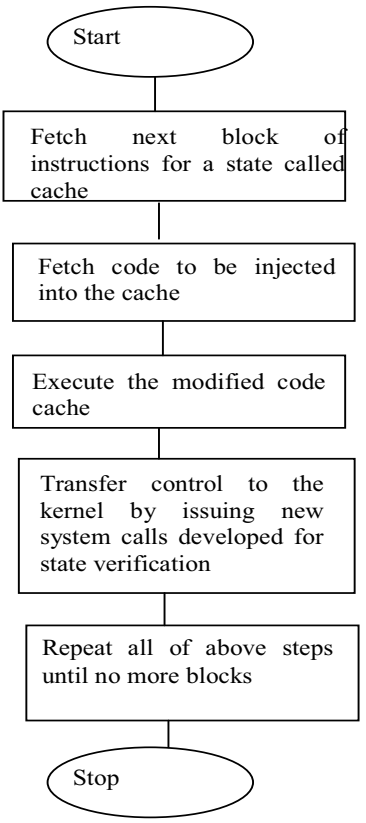


Figure 4.0 giving the flow chart for the current execution and the proposed execution model for the runtime state tracing kernel

The state chart diagram for the approach presented herewith in this paper is shown in figure 4.1 as follows. The normal execution of a single binary is to the left and the execution of the binary using the new approach is presented to the right. (The blocks in the picture are not to scale) . In the picture in the left, the times t_2 and t_4 represent the time spent by the binary in the kernel space (probably executing a system call) and the times t_1,t_3 and t_5 represent the time spent in the user space.

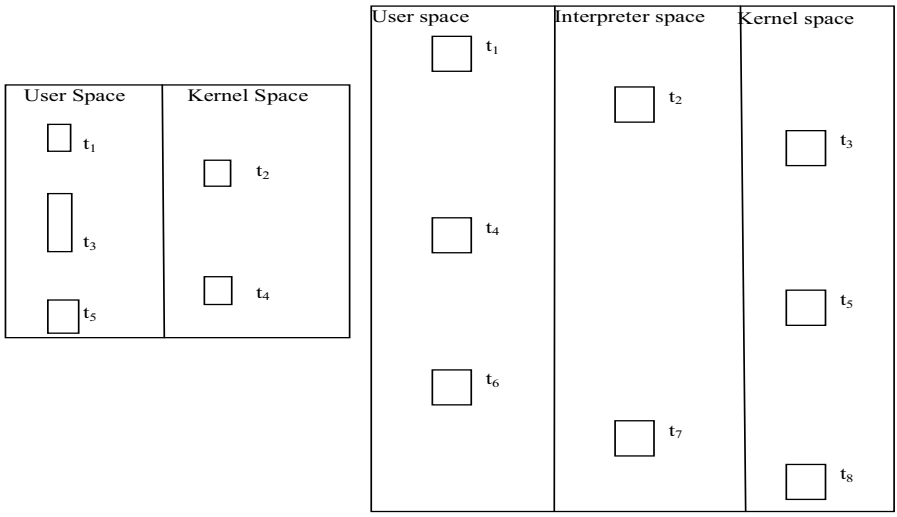


Figure 4.1 Giving the state flow diagram for building the runtime state tracing Linux kernel

In the state flow diagram above, t_i occurs t_j if $i < j$.

The following observations need to be made from the state transition diagram from the two approaches as given above

- The amount of total time taken to execute the binary is definitely going to increase.
- The Interpreter acts as a sandbox under which the binary to be executed is to be run.
- There is some code as part of the interpreter which is executed intermixed with the code of the binary
- The number of system calls may increase proportionally to the number of states.

5 Building the state set

Building the state set involves in reverse engineering the binary. A state may be defined as the collection of sequential instructions that do not branch off due to a jump (conditional/non-conditional) or call instructions. A jump or call instructions is called a transition. A transition path may be taken or not taken based on the execution context. This context is generally decided by a number of factors like user input, state of the data read from external files and a host of other parameters. There are currently more than one executable formats for the Linux platform (ELF, a.out formats). The current effort involves revere engineering the ELF format executables. Following is a brief primer on the ELF executable structure and how it can be reverse engineered to obtain the total set of states.

Another characteristic of the states that are generated is that these states can be assigned a signature if required. It may not be possible that all of the states that are generated can be assigned a signature. It may be noted that the state set is generated on a binary and the signatures that are generated for the states is based only on the static analysis but not on dynamic analysis. The very reason that the signature cannot be extracted for some of the states that the state traversal is based on the fact there are a lot of factors that are decided based on the context like the state of the registers and the value of the memory locations.

This section describes in detail the approach taken by the tool developed to generate the state set. It may be noted that the states and state signature cannot be developed from source code if the developer chooses to mark a state signature because a single line of code translates into a number of states. Before looking into the state generation, one needs to understand the layout of a binary.

An ELF executable/binary is made of sections and segments. Here is the execution view of the ELF format binary.

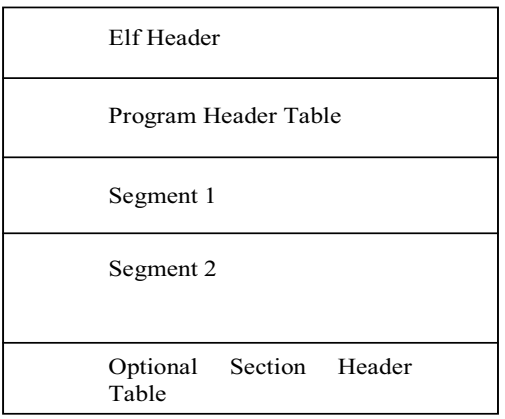


Figure5.1. A block diagram giving the organization of an ELF binary

Segments are loaded into the memory allocated for the execution of the binary. A segment for analysis purposes can be seen as a collection of sections. Each section is made up for a particular kind of information for the execution of the binary. For example, text section contains the instructions that are executed when the control reaches the user space of the binary, the PLT section contains the code to connect the text section based library calls to the actual text section of the library function that is being called. Other sections like the dynamic section contains the dynamic symbols information that are required by the loader to build the image of the binary and the required libraries properly. For more details, the reader is encouraged to the ELF binary specification which gives the meaning of the sections that can be associated with a ELF binary on the Linux Intel platform.

For finding out the sequences of instructions that do not involve a jump/call instructions, the text section needs to be disassembled to get the collection of instructions in the text segment. The PLT section also needs to be disassembled for more advanced analysis of the text section; for example finding out the library function being called. An XML version of the disassembled code complete with the library function call references as generated by the tool developed is given hereunder

```
        08056E01        8B 85 F4 EF FF FF       mov     eax
        [ebp-4108]

        08056E07        05 00 10 00 00  add     eax     0x00001000
        08056E0C        3B 45 14 cmp    eax     [ebp+20]
        08056E0F        0F 83 8E 00 00 00 jnc   0x08056EA3
        08056E15        83 EC 04        sub     esp     0x04
        08056E18        68 00 10 00 00  push    0x00001000
        08056E1D        8D 85 F8 EF FF FF       lea     eax
        [ebp-4104]

        08056E23        50      push    eax
        08056E24        FF 75 0C        push    [ebp+12]
        08056E27        E8 D8 33 FF FF  call    0x0804A204
                <FunctionHint>read</FunctionHint>

        08056E2C        83 C4 10 add    esp     0x10
08056E2F        3D 00 10 00 00  cmp     eax     0x00001000
        08056E34        74 1F   jz      0x08056E55
        08056E36        83 EC 0C        sub     esp     0x0C
        08056E39        68 14 15 09 08  push    0x08091514
        08056E3E        E8 C1 2E FF FF  call    0x08049D04
         <FunctionHint>perror</FunctionHint>            </InstructionList>
</FunctionCodeChunk>
```

Figure 5.2 Disassembled code for a sample text section

On analyzing the figure 5.2, it may be observed that there are four transitions that can take place at the virtual addresses 0x08056E0F, 0x08056E27, 0x08056E34, 0x08056E3E. As per the definition of state as given earlier, the above disassembly would yield four states and four transitions for each of the states identified. Hence the states are as follows: 08056DEE – 08056E0C , 08056E15 – 08056E24, 08056E2C - 08056E2F, 08056E36 – 08056E39.

The tool that has been developed would help in identifying all the possible states for a given binary using the above approach. Here is a chart that gives an estimate of the number of states that could be generated for some of the common binaries on the Linux Intel platform.

Name of the binary	#of states
/bin/ls	2883
/bin/cat	495
/opt/ibm/java2-i386-50/bin/java	2030
/usr/bin/gcc	360
/usr/bin/find	2042

Figure 5.3 Table giving the number of states for some of the binaries available on the RedHat Linux kernel 2.6.19

On generating the set of states that can be reached for a binary, the next step is to generate the state signatures. A state signature forms the building block for verifying the state and hence the state transition when the context switch happens in the kernel space. The signature of a state can be arrived at by using a number of factors or a combination of these factors. For example, the state signature can be specified manually by specifying the signature of a state by using host of factors of the current execution context like number of open files, state of the network interface, number of system calls made till that state is reached, and the time that could be taken to arrive at that state. Additionally, a static analysis of the binary could be performed and then add this set to the list of state signatures.

From a static analysis point of view, some of the factors that could be used to build a state characteristic are as follows:

- Memory state of the registers
- Memory state of some of the global variables
- Memory state of the function variables.

While manually specifying the state signature might be easy(for example, stating that when the control reaches a particular state, the process should have this many files open and this many child processes), it might be too vague because of the following reasons.

Programs are mostly written in a high level programming language and a single line of code might transform into a number of states in the binary format. Hence associating a state signature using manually identifiable characterstics is difficult to arrive at.

It is possible that such a signature can be associated with multiple states. It is possible that the all of these state transitions occur in a serial fashion and hence indistinguishable.

2 Identifying state signature using register states

Identifying the state by peeking into the state of the registers by doing a static analysis assumes that after each transition, the state of some of the registers is going to change or not change at all. (Even a no change can be used to obtain the signature of a state). It is also possible that the contents of the register can go into an indeterminate state. Hence it is possible that not all of the states can be associated with a state signature.

Following is the algorithm that could be used to extract the state signature of a state basing on the register states if it exists.

```
Initialize the EAXVAL, EBXVAL, ECXVAL, EDXVAL, ESPVAL, ESIVAL,
EDIVAL to UNDETERMINATE state.

Initialize the flags to track whether a register has changed; //  To  set  no change in
// register as another signature

For Each Instruction in the total instructions of the state  {

        instr = current Instruction in the instruction list;

        if (instr == PUSH ) decrease the ESPVAL by 4; // Assuming 32 bit
        code

        if (instr == POP ) increase the ESPVAL by 4;

        Then For Each Register EAX, EBX, ECX, EDX, ESI , EDI  Repeat

        if (instr contains current Register) {

                if instr contains “mov” then set current Register as value of
                the second operand
```

if (instr contains “add” && second operand can be resolved as a number) mark that the register value has increased by the value of second operand;

Perform equivalent operations for subtraction, multiplication and division

if instr changes the register to something which cant be resolved statically set register as indeterminate.

If the register goes into an undeterminate state then remove all inferences arrived at till this point

Figure 6.1 Algorithm for extracting the register state for instructions that constitute a state by static analysis.

Following table lists the observations for some of the common executables available on the Linux Intel platform.

Name of the binary	# of states for which signature can be extracted excluding stack changes but including relative register changes(A)	# of states for which signature can be extracted including stack changes (B)	# of states for which signature could be extracted versus the total # of states (A+B)/Total States
/bin/ls	296	682	682/2883 = 23.65 %, (10.26 % excluding stack)
/bin/cat	45	142	142/495 = 28.68 % (9.09 % excluding stack)
/opt/ibm/java2-i386-50/bin/java	152	991	991/2030 = 48.81 % (7.48% excluding stack)
/usr/bin/gcc	32	113	113/360 = 31.38 % (8.88 % excluding stack)
/usr/bin/find	175	492	492/2042 = 24.09 % (8.5 % excluding stack)

Figure 6.2 Table giving the number of states that can be verified versus the total number of states

As can be seen not all of the states can be verified by using the mechanism of register state changes. The above calculation distinguishes between stack based changes versus non stack based changes as stack based changes are relative to previous state of the stack.

2 Identifying state signature using memory state of variables

State signature can also be calculated using the state of some of the global variables whose state changes following transition triggers. For example, in one of the states, the value of a global integer variable might be changed from 4 to 5. To track the changes , the analysis tool needs to analyze instructions that set the value of the global variables that are described in the symbol table which is present as ".symtab" section inside the binary(assuming the binary is not stripped of symbol information). If the file is stripped of symbol information, then the tool needs to analyze the ".text" section for all the instructions that refer the ".data" and the ".bss" sections. Both the ".data" and the ".bss" sections constitute to the data segment in the process image. As per the ELF binary specification, ".bss" section contains the uninitialized data that contributes to the process image. This section does not contribute to any space in the file but occupies memory once the ".bss" section is loaded into the memory. On the other hand, ".data" section contains the data variables that are initialized to some value. Below is the algorithm that is used to obtain the state signature basing the change values to the global data variables.

```
If symboltable is present in the binary then do the following else go to
RAWPARSE:

For each symbol in the symbol table

  Get symbol with the following characteristics from the symbol table

  Symbol type = Object type symbol.

  Applicable Section Header index == Section index of ".bss" section or the
".data" section

  If a match is found get the Symbol size.
  End for.

For each state perform

  For each instruction in the state perform the following

        If the instruction modifies any of the symbols obtained in the previous
block which can be identified explicitly, mark the state change

  End Inner For

End Outer For; goto END

RAWPARSE:

For each state perform

  Obtain the start and end virtual address of the ".bss" and ".data" sections

  For each instruction in the state perform the following

        If the instruction refers an address with the range of the virtual
addresses obtained in step one then

        Obtain the size of the symbol based on the opcode of the instruction
(for further analysis if required)

        See if the instruction helps in generating the state signature mark the
state as recognizable

  End Inner For

End Outer For

END:

  Stop.
```

Figure 6.3 giving the algorithm for state signature extraction based on variables

Following are the observations regarding the state signature extraction based on the value of global variables.

Name of the binary	# of states for which signature can be extracted using global variable change (A)	# of states for which signature could be extracted versus the total # of states (A)/Total States
/bin/ls	1	1/2883 = 0.03 %
/bin/cat	1	1/495 = 0.2%
/opt/ibm/java2-i386-50/bin/java	16	16/2030 = 0.7 %
/usr/bin/gcc	9	9/360 = 2.5 %
/usr/bin/find	60	60/2042 = 2.9 %

Figure 6.4 Table giving the results for state signature using global variables

From the above table it is clear that the number of states that could be detected using the global variables state is very little. Also it maybe noted that these states do not take into account that there are some states whose signature can be calculated using the state of some of the registers during the previous state.

Identifying the state signature using the function variables

The state of the function variables could be used to calculate the state signatures. For example, considering the following piece of dummy code.

```
static int staticfuncWithIntReturn(int one, int two, char three) {
  int localVarOne =6;
  printf("Inside the static Function with int return %d %d %c\n",one,two,three);
  scanf("%d",&localVarOne);
  if (localVarOne > 12) {
        localVarOne +=7;
  }
  else localVarOne +=2;
  printf("The value of the local variable is %d\n",localVarOne);
}
```

Figure 6.5 giving the a piece of code to describe function variables access

It is evident that there is a local variable that is being manipulated at various points in this function. It can also be concluded that this function can be associated with multiple states. The disassembled code for the above function from the tool developed is as follows:

The following observations could be made from Figure 6.6 and Figure 6.5. The function is passed three variables and has got one local variable "localVarOne" which is of type int. Looking at the Intel instruction set volume I, following is the structure of the stack when a function call is being made.

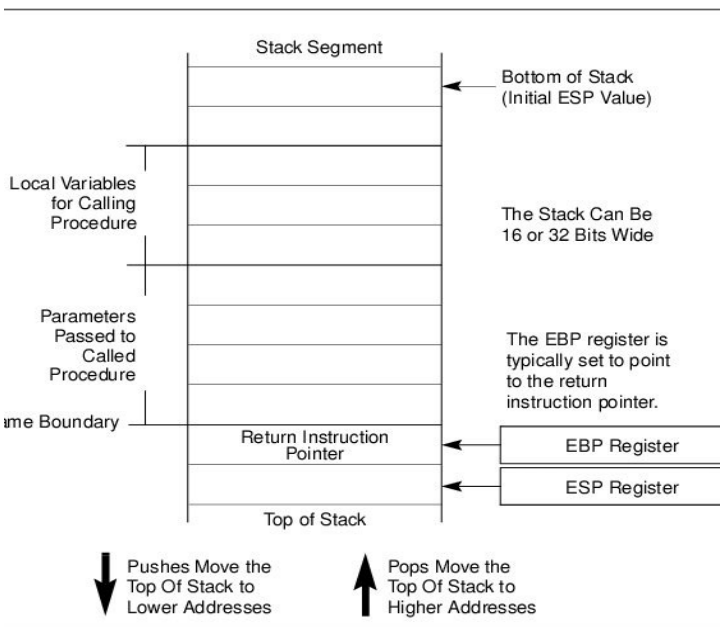


Figure 6.7 Stack layout using the stack call layout (Source Intel Instruction Set Volume I)

Correlating figures 6.5, 6.6 and 6.7, it can be deduced that the local variable "localVarOne" is being referred by the memory location [ebp+248]. However it may be observed that in some locations, the reference is passed to another register and the register is manipulated by using a reference mechanism. Hence the challenge lies in locating such a pattern and identifying the state changes. From the above section of disassembled code in Figure 6.6, it can be observed that the address ranges 0804839C-080483BF , 080483C4- 080483D3, 080483D8- 080483DF, 080483E1-080483E7, 080483E9- 080483FA, 080483FF- 08048403 constitute the six states S^1,S^2,S^3,S^4 ,S^5 and S^6 of the function. The state signature for the state S^1can be verified by checking whether the memory value at the location given by [ebp+248] to be the value 0x00000006. It can also be concluded that the traversal path can be deduced by checking the value of the memory location [ebp+248]. If it has increased by 0x7 then the state S^4 has been traversed whereas if the memory location has increased by 2 then the state S^5 has been traversed. This checking can be initiated in the state S^6 which comes later after traversing one of the states S^4 or S^5.

The following table summarizes the number of states whose signature could be calculated using the state of the function variables by using the approach mentioned above.

The following observations could be made from Figure 6.6 and Figure 6.5. The function is passed three variables and has got one local variable "localVarOne" which is of type int. Looking at the Intel instruction set volume I, following is the structure of the stack when a function call is being made.

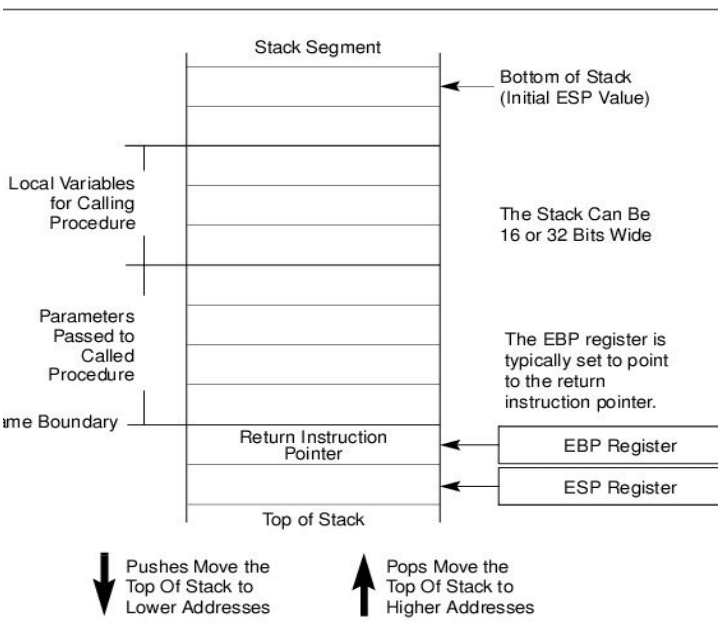


Figure 6.7 Stack layout using the stack call layout (Source Intel Instruction Set Volume I)

Correlating figures 6.5, 6.6 and 6.7, it can be deduced that the local variable "localVarOne" is being referred by the memory location [ebp+248]. However it may be observed that in some locations, the reference is passed to another register and the register is manipulated by using a reference mechanism. Hence the challenge lies in locating such a pattern and identifying the state changes. From the above section of disassembled code in Figure 6.6, it can be observed that the address ranges 0804839C-080483BF , 080483C4- 080483D3, 080483D8- 080483DF, 080483E1-080483E7, 080483E9- 080483FA, 080483FF- 08048403 constitute the six states S^1,S^2,S^3,S^4 ,S^5 and S^6 of the function. The state signature for the state S^1can be verified by checking whether the memory value at the location given by [ebp+248] to be the value 0x00000006. It can also be concluded that the traversal path can be deduced by checking the value of the memory location [ebp+248]. If it has increased by 0x7 then the state S^4 has been traversed whereas if the memory location has increased by 2 then the state S^5 has been traversed. This checking can be initiated in the state S^6 which comes later after traversing one of the states S^4 or S^5.

The following table summarizes the number of states whose signature could be calculated using the state of the function variables by using the approach mentioned above.

Name of the binary	# of states for which signature can be extracted using function variable change (A)	# of states for which signature could be extracted versus the total # of states (A)/Total States
/bin/ls	146	146/2883 = 5.06 %
/bin/cat	32	32/495 = 6.46%
/opt/ibm/java2-i386-50/bin/java	63	63/2030 = 3.1%
/usr/bin/gcc	0	0/360 = 0 %
/usr/bin/find	113	113/2042 = 5.53 %

Figure 6.8 Table giving the number of states whose signature can be calculated using the function variables state .

It maybe noted that there could be some indeterminable states based on the algorithm because of the states use indirect manipulation by first copying the reference of the memory location of the function variable into a different register and then perform operations using references.

Combining the mechanisms to extract the state signature using the various mechanisms mentioned in the previous three sections, the following table summarizes the total number of states versus the total number of states for a binary which can be interpreted at runtime.

Name of the binary	# of states for which signature can be extracted using registers, global variables and function variables	# of states for which signature could be extracted versus the total # of states /Total States
/bin/ls	682+1+146	829/2883 = 28.75 %
/bin/cat	142+1+32	175/495 = 35.35 %
/opt/ibm/java2-i386-50/bin/java	991+16+63	1070/2030 = 52.70 %
/usr/bin/gcc	113+9+0	113/360 = 33.38 %
/usr/bin/find	492+ 60+113	665/2042 = 32.56 %

Figure 6.9 Table giving the number of state signatures that can be verified versus the total number of states

Having seen how the state signatures can be extracted from a binary, the next step lies in building the interpreter which would interpret the execution of the binary and possibly verify the state signature and pass control to the kernel to further verify in case it mandates the kernel verify the state of the running process. The following section explains the interpreter that has been developed and extended using the dynamorio framework.

The Interpreter

It may be noted that the interpreter could be either the loader that is loading the process or a separate process that could get control by interpreting each instruction of the running process just like a normal interpreter. The interpreter needs to have the following characteristics:

It should be able to interrupt the process at the end of each state (It may not necessarily do so for each and every state to increase performance)

It should have the mechanism to inject code into the running process as required.

It should be able to peek into the state of the running process to extract values like register contents and the memory locations

It should be able to make system calls to the underlying kernel so that extra validation could be done inside the kernel. Some of the common verification mechanisms are currently open file handles, network sockets etc.

Dynamorio is a framework that meets all of the above requirements. It is a code caching framework that fetches blocks of code and executes the cached code and then gains control back after executing the block. Before executing the basic block, dynamorio allows the programmer to inject instructions into the code cache and there by mix code from both the verifier module that is verifying the state of the binary and the code from the binary itself. Dynamorio was chosen to showcase that the whole approach of building a verifiable system by using some of the already existing frameworks and not building from the scratch. The overhead of having a code caching framework over a dynamic interpreter has been overlooked to verify the idea of runtime quickly.

Dynamorio is the primary process that is launched by the user when executing the binary. Dynamorio , based on the options passed through environment variables, lets additional libraries to be loaded before the binary is launched. This way, custom code could be injected into the execution path of the binary. For the runtime state tracing framework to work as designed, a library has been developed that has the following characteristics

An xml parser sub-module that can parse the output generated from the tool that generated the state set and their verification signatures. It may be noted that the output of the state generation tool is in XML format and hence the need for the xml parser sub-module.

A mechanism to track state traversal based on the EIP values i.e. the current point of execution of the binary. (The value might be a valid state traversal or invalid the binary is injected with malicious code.) This part of the module just ensures that the control has gone from a state with an address A to another state with address B. It might be verified through other part of the modules of the library that address B is a valid or not valid state traversal.

A sub-module that has mechanisms to peek into the state of the running process. This should have the capability to read the state of the registers and memory locations that are currently allocated to the binary.

A mechanism to talk to the kernel to mark the beginning of the execution of the binary, the transfer of control to various states and the mark the stopping of the execution of the binary.

Since dynamorio takes control during various points of execution, this means that dynamorio saves the context of the running process on the stack designated for dynamorio and this stack is different from the stack that is used by the binary. Hence care needs to be taken in the library that was developed, so that while peeking the code or state of the running binary, it should not be the dynamorio stack but the stack and context of the running binary that should be examined.

The library with the above characteristics is loaded by dynamorio and then dynamorio makes system calls to the underlying modified Linux kernel . The changes made to the Linux kernel are explained in the next section.

Changes to the Linux kernel

After having the interpreter in place, the Linux kernel also needs to be modified because mechanisms need to be in place for

Tracking additional information for the life of the process like the states traversed from the beginning of execution of the binary to any point later in time, the total number of states for the running process and the valid state model for the binary.

Extra API/system call mechanisms to mark the beginning, execution and end of the running process through the intermediary, the dynamorio library, that was developed as described in the previous section.

Track the usage of various resources like the open file handles, the number of sockets open, time spent in the user space and the kernel space.

The following paragraph describes the changes made to the kernel to implement the above mentioned functionality.

"struct task_struct", the data structure that holds the process specific information is modified as follows.

```
struct task_struct {

  volatile long state; /* -1 unrunnable, 0 runnable, >0 stopped */

  struct thread_info *thread_info;
  struct reg_vals * reg_vals_sem;

  struct process_runtime_semantic_desc_node * runtime_history;

  struct process_runtime_semantic_desc_node * current_point_of_execution;

  struct process_static_semantic_state_node * static_model;

  struct process_static_semantic_state_node * current_point_in_execution_model;

  struct process_info * process_metadata_info;

  long int num_of_places;
```

Figure 8.1 The primary data structure for process tracking in the Linux Kernel

As can be seen, the structure in the modified kernel holds the following additional information.

The variable “reg_vals_sem” represents the current value of the running process when the context switch happened from the user space to the kernel space. “runtime_history” represents the states traversed from the beginning of execution till the current point in time. “current_point_of_execution” represents the current node that has been identified using the signature. “static_model” represents the model that has been generated for the binary and the variable “current_point_in_execution_model” represents the state signature that the variable “current_point_execution” should be verified with. The variable “process_metadata_info” represents the information about the binary like the name of the binary, the value of the starting virtual address and other information about the process program header table. The variable “num_of_places” represents the total number of states that can be associated with the binary.

Apart from modifying the task_struct data structure, the kernel needs to have new system calls so that control is transferred to the kernel via the dynamorio interpreter library. Some of the system calls that are implemented are as follows:

- asmlinkage **long sys_semlog(long int** model_desc,**long int** instr_addr, **long int** target_addr, **long int** reg_state)
- asmlinkage **long sys_seminit(char** * name_of_exec ,**long int** ptrToModel,**long int** numberOfPlaces, **long int** ptrToProcInfo)
- asmlinkage **long sys_semdestroy(long int** firstparam ,**long int** entryname,**long int** thirdparam)

The first system call semlog is the mostly used sytem call by the interpreter dynamorio library to mark the state traversal. Some of the parameters that are passed to the kernel through this system call are the current instruction virtual address, the target virtual address which the control would go to in the case of the state transition occurs when the context switch happens to the user space back.

The second system call seminit is used to mark the beginning of execution of the process. Some of the parameters that are passed to the kernel are the name of the process that is being launched, a pointer to a memory location in the user space so that the kernel can copy the model to the dynamorio accessible memory location to implement state signature verification if required. Another parameter “ptrToProcInfo” is used to pass the proc_metadata_info that is used in the kernel task_struct data structure to keep track of the process program header table information.

The last system call is used to free any kernel memory acquired by calling kmalloc kind of kernel memory allocation calls. It may be noted that the primary task is done in the semlog system call which is used to perform various verifications.

Below are some of the system characteristics that could be used to verify the state of the running process. It may be noted that these characteristics does not require a state model to be verified.

The time spent by the process in the kernel and the time spent in the user space (t)

The number of open handles to the file system

The network interfaces open

The number of child processes.

The number of system calls made till that point of execution

The above factors are calculated using the standard kernel data structures in the modified Linux kernel.

Observations

It is evident from the architecture that there is a hit on the performance of the binary because of code caching technique and increased number of system calls. The following observations were made while running some of the common binaries which are available on the Linux platform. The table below summarizes the process accounting information in two scenarios. 1. The execution of the binary under the dynamorio without having the custom library inject any code and 2. The execution of the binary with the code injection also in place.

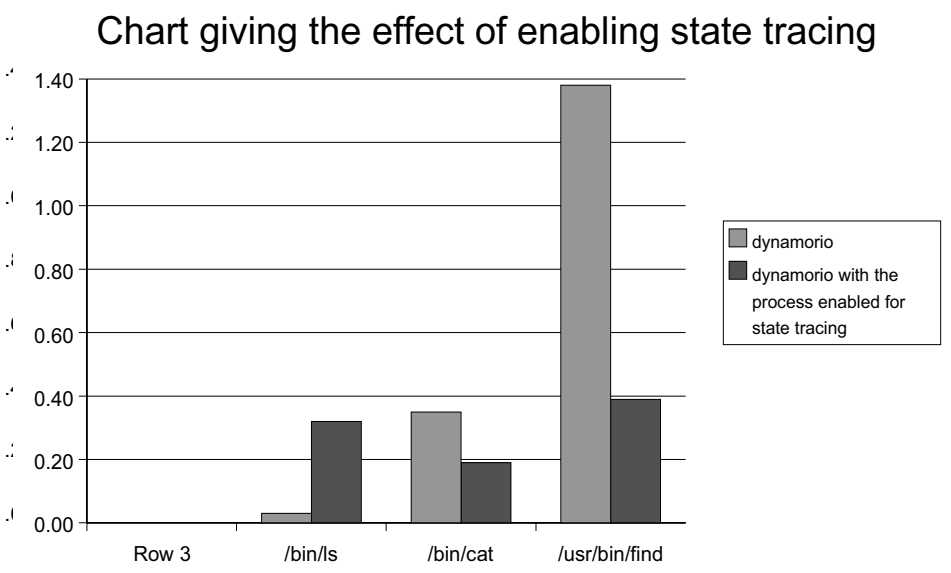


Figure 9.1 Chart giving the performance analysis of the new approach . Time is on y-axis in seconds

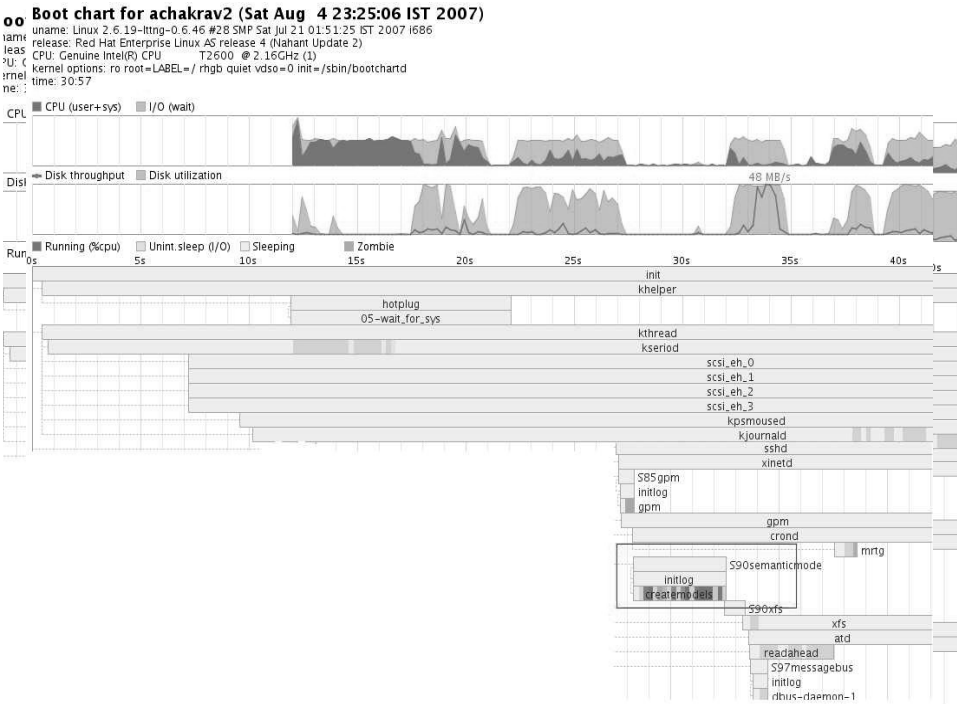
From the chart it is evident that the performance has increased after injecting code. It is because dynamorio enhances the performance of the code caching mechanism once it recognizes the common pattern while executing.

It may be noted that a better performance can be increased by doing most of the verification in the interpreter space rather than the kernel space. This would result in increase of the performance because the number of system calls related to the verification mechanism is going to decrease in proportional to the states whose signature can be verified using the register states. Since this mechanism is applicable on the functionality of the binary but not every binary in general, the following performance improvements have been considered.

Delegate the process of loading the state description files, i.e. the XML files at the time of booting the kernel rather when the process is getting launched for execution.

Delegate the process of verifying the state of the process in the interpreter space and switch to the kernel verification mechanism only when required, for example only when verifying the open file handles, the number of child processes.

The following chart gives the effect of implementing the concept of loading the models at boot time.



The effect of loading the state description files which are in XML format while the kernel is booting rather than when the process is getting launched is seen in the following chart. The chart is shown for a RedHat Linux kernel. The time taken to load the models amounting to a size of 15.2 MB is shown in the chart above. The downside of this approach is that it might a little more time to boot the kernel . From the reading on the chart, a set of models for some of the common binaries with a file size of 15.2 MB amount a boot time delay of 4 seconds.

Regarding the second aspect of improvement, that is by delegating the state verification to the interpreter space rather than the kernel wherever possible will definitely see an increase in the performance. For example if the state signature is to be verified based on the values of the registers than on the state of the kernel resources , then verification could be done in the interpreter space itself.

Conclusions and Future work

As can be seen from the above sections, a state tracing mechanism could be implemented using the approach as discussed in this paper. The first step involves in generating the model by doing a static analysis of the binary and then generating state signatures. These state signatures can then later be verified using an interpreter framework using dynamorio and a modified Linux kernel.

Future work involves in creating an optimized state model for a given binary and a tool which can describe the state model by using a semantic dictionary. By having a model which can be both verified by the kernel while executing and at the same time verifiable by a human can go a long way in improving the security of binaries on operating system.

Bibliography

1. [BGA 03] An infrastructure for adaptive dynamic optimization
   Bruening , D.; Garnett, T.; Amarasinghe, S.;
   Code Generation and Optimization, 2003. CGO 2003. International Symposium on
   23-26 March 2003 Page(s):265 - 275
   Digital Object Identifier 10.1109/CGO.2003.1191551
2. [BKGB 06] Thread-shared software code caches
   Bruening, D.; Kiriansky, V.; Garnett, T.; Banerji, S.;
   Code Generation and Optimization, 2006. CGO 2006. International Symposium on
   26-29 March 2006 Page(s):11 pp.
   Digital Object Identifier 10.1109/CGO.2006.36

3. [BA 05] Maintaining consistency and bounding capacity of software code caches
   Bruening, D.; Amarasinghe, S.;
   Code Generation and Optimization, 2005. CGO 2005. International Symposium on
   20-23 March 2005 Page(s):74 - 85
   Digital Object Identifier 10.1109/CGO.2005.19

4. [] Online impact analysis via dynamic compilation technology
   Breech, B.; Danalis, A.; Shindo, S.; Pollock, L.;
   Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on
   11-14 Sept. 2004 Page(s):453 – 457 Digital Object Identifier 10.1109/ICSM.2004.1357834

5. Dynamorio Home page http://www.cag.lcs.mit.edu/dynamorio/

6. Derek Bruening's Ph.d thesis http://www.burningcutlery.com/derek/phd.html

7. Some notes on adding system calls in Linux http://fossil.wpi.edu/docs/howto_add_systemcall.html

8. ELF Specification is available at http://x86.ddj.com/ftp/manuals/tools/elf.pdf

9. ELF specification explained in detail http://www.cs.ucdavis.edu/~haungs/paper/node10.html

10. [YI 01] A super tracer and an analyzer for analyzing detailed behavior of a Linux on a Pentium family processor (STDB)
    Y. Sugimura; S. Ido;
    Engineering of Computer Based Systems, 2001. ECBS 2001. Proceedings. Eighth Annual IEEE International Conference and Workshop on the
    2001 Page(s):298 - 305
    Digital Object Identifier 10.1109/ECBS.2001.922436

11. [E 01] Summary of dynamically discovering likely program invariants
    Ernst, M.D.;
    Software Maintenance, 2001. Proceedings. IEEE International Conference on
    7-9 Nov. 2001 Page(s):540 - 544
    Digital Object Identifier 10.1109/ICSM.2001.972767

12. [ECGN 00] Quickly detecting relevant program invariants
Ernst, M.D.; Czeisler, A.; Griswold, W.G.; Notkin, D.;
Software Engineering, 2000. Proceedings of the 2000 International Conference on;
4-11 June 2000 Page(s):449 - 458
Digital Object Identifier 10.1109/ICSE.2000.870435

13. [BBP 75] Speeding up the Synthesis of Programs from Traces
Biermann, A.W.; Baum, R.I.; Petry, F.E.;
Computers, IEEE Transactions on
Volume C-24, Issue 2, Feb. 1975 Page(s):122 - 136

14. [LN 97] Object-oriented program tracing and visualization
Lange, D.B.; Nakamura, Y.;
Computer
Volume 30, Issue 5, May 1997 Page(s):63 - 70
Digital Object Identifier 10.1109/2.589912
15. [ZBV 03] Classification of anomalous traces of privileged and parallel programs by neural networks
Zhen Liu; Bridges, S.M.; Vaughn, R.B.;
Fuzzy Systems, 2003. FUZZ '03. The 12th IEEE International Conference on
Volume 2, 25-28 May 2003 Page(s):1225 - 1230 vol.2

16. [HS 02] An instruction set and microarchitecture for instruction level distributed processing Ho-Seop Kim; Smith, J.E.; Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on 25-29 May 2002 Page(s):71 - 81 Digital Object Identifier 10.1109/ISCA.2002.1003563

17. [MS 93] Applying algorithm animation techniques for program tracing, debugging, and understanding Mukherjea, S.; Stasko, J.T.; Software Engineering, 1993. Proceedings., 15th International Conference on 17-21 May 1993 Page(s):456 - 465 Digital Object Identifier 10.1109/ICSE.1993.346020
18. [BK 04] Aspect mining using event traces Breu, S.; Krinke, J.; Automated Software Engineering, 2004. Proceedings. 19th International Conference on 2004 Page(s):310 - 315 Digital Object Identifier 10.1109/ASE.2004.1342754

19. [ED 87] Characterization of Branch and Data Dependencies in Programs for Evaluating Pipeline Performance Emma, P.G.; Davidson, E.S.;
Computers, IEEE Transactions on Volume C-36, Issue 7, Jul 1987 Page(s):859 - 875

20. [BM 03]Compressing extended program traces using value predictors Burtscher, M.; Metha Jeeradit; Parallel Architectures and Compilation Techniques, 2003. PACT 2003. Proceedings. 12th International Conference on 27 Sept.-1 Oct. 2003 Page(s):159 - 169 Digital Object Identifier 10.1109/PACT.2003.1238012

21. [KN 05] Construction and compression of complete call graphs for post-mortem program trace analysis Knupfer, A.; Nagel, W.E.;
Parallel Processing, 2005. ICPP 2005. International Conference on 14-17 June 2005 Page(s):165 - 172 Digital Object Identifier 10.1109/ICPP.2005.28

22. [RR 05] Demonstration of JIVE and JOVE: Java as it happens Reiss, S.P.; Renieris, M.; Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on 15-21 May 2005 Page(s):662 - 663 Digital Object Identifier 10.1109/ICSE.2005.1553642

23. [HMJ 91] Detecting data races from sequential traces Helmbold, D.P.; McDowell, C.E.; Jian-Zhong Wang; System Sciences, 1991. Proceedings of the Twenty-Fourth Annual Hawaii International Conference on Volume ii, 8-11 Jan. 1991 Page(s):408 - 417 vol.2 Digital Object Identifier 10.1109/HICSS.1991.184003

24. [IG 06] Efficient Incremental Optimal Chain Partition of Distributed Program Traces Ikiz, S.; Garg, V.K.; Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on 04-07 July 2006 Page(s):18 - 18 Digital Object Identifier 10.1109/ICDCS.2006.34

25. [L 93] Efficient program tracing Larus, J.R.; Computer Volume 26, Issue 5, May 1993 Page(s):52 - 61 Digital Object Identifier 10.1109/2.211900
26. [PK 93] Multilevel Simulation Of Distributed-memory Program Traces Prost, J.-P.; Kipnis, S.;Simulation Symposium, 1993. Proceedings. 26th Annual March 29- April 1,1993 Page(s):60 – 67

29. [NP 96] The effect of program behavior on fault observability Bowen, N.S.; Pradhan, D.K.; Computers, IEEE Transactions on
Volume 45, Issue 8, Aug. 1996 Page(s):868 - 880 Digital Object Identifier 10.1109/12.536230

30. The open source IDE used in developing the code and the modified Linux kernel http://www.eclipse.org/

31. The Linux Process Manager: The Internals of Scheduling, Interrupts and Signals by John O'Gorman John Wiley & Sons © 2003 For explanation of some of the most commonly used Linux Kernel data structures

32. The eclipse C/C++ plugin site , the IDE plugins used to develop the tool http://www.eclipse.org/cdt/

Formally Specifying Operational Semantics and Language Constructs of Forensic Lucid

Serguei A. Mokhov Joey Paquet Mourad Debbabi
Faculty of Engineering and Computer Science
Concordia University, Montréal, Québec, Canada,
{mokhov,paquet,debbabi}@encs.concordia.ca

Abstract: The Forensic Lucid programming language is being developed for intensional cyberforensic case specification and analysis, including the syntax and operational semantics. In significant part, the language is based on its predecessor and codecessor Lucid dialects, such as GIPL, Indexical Lucid, Lucx, Objective Lucid, and JOOIP bound by the intensional higher-order logic that is behind them. This work continues to formally specify the operational semantics of the Forensic Lucid language extending the previous related work.

1 Introduction

We further define a functional-intensional programming language, called Forensic Lucid. This forensic case specification language is under extensive design and development including its syntax, semantics, their formalization and correctness proofs, the corresponding compiler, run-time environment, and interactive development environment. This work further extends our previous developments in the related work [Mok04, Mok07b, Mok07a, MP08a].

1.1 Problem Statement

A lot of Lucid dialects have been spawned from the 30+-year-old functional intensional programming language called Lucid [WA85, Edw95, Zha97, AW76, AW77a, AW77b, FB91, Du94, GP99, Paq99, WAP05a, Agi95]. Lucid itself was invented with a goal for program correctness verification at the time. While there were a number of operational semantics rules for compilers and run-time environments developed for all those dialects throughout the years. In this work we discuss a new dialect of Lucid that has been created to foster the research in *intensional cyberforensics* (i.e. *multidimensional context-oriented cyberforensic specification and analysis*), called Forensic Lucid, which, in large part, is a union of the syntax and operational semantics inference rules from the comprising languages with the forensic extensions based on the finite-state automate approach [Gla05, GP04]. In order to be a credible tool to use in court to implement relevant tools for the argumentation, the language must have a solid scientific base, a part of which is a complete formalizing the syntax and semantics the language.

1.2 Proposed Solution

Based on the previous work to begin mechanized specification and proofs of Forensic Lucid constructs, their equivalence to the comprising dialects for the correctness aspect verification, in Isabelle [PN07], a prover assistant program [MP08a], we further expand and refine the syntax and operational semantics' inference rules extended with the ones of the comprising Lucid dialects. We proceed with the "core" Lucid dialects such as GIPL [Paq99, PMT08], a conservative subset of Lucx [PMT08] to Objective Lucid [MP05, Mok05], JOOIP [WPM08], MARFL [Mok08] and finally Forensic Lucid, to arrive to a comprehensive set of syntactic and semantic rules covering the dialects.

1.3 Organization

We first rehash the notion of intensional logic and programming for the unaware reader. We present the refined syntax and the semantics of the language properly attributing the inherited language constructs and rules, and the extensions of the language.

1.4 Intensional Programming

1.4.1 Definitions.

Intensional programming (IP) is based on intensional (or multidimensional) logics, which, in turn, are based on Natural Language Understanding (aspects, such as, time, belief, situation, and direction are considered). IP brings in **dimensions** and **context** to programs (eg. space and time in physics or chemistry). Intensional logic adds dimensions to logical expressions; thus, a non-intensional logic can be seen as a constant or a snapshot in all possible dimensions. *Intensions are dimensions* at which a certain statement is true or false (or has some other than a Boolean value). *Intensional operators* are operators that allow us to navigate within these dimensions. Higher-order Intensional Logic (HOIL) [MP08b, MP08a, Ron94] is the one that couples functional programming as that of Lucid with multidimensional dataflows that the intensional programs can query an alter through an explicitly notion of contexts as first-class values [Wan06, PMT08].

1.4.2 An Example of Using Temporal Intensional Logic.

Temporal intensional logic is an extension of temporal logic that allows to specify the time in the future or in the past.

(1) E_1 := it is raining **here today**

Context: {`place`:**here**, `time`:**today**}

(2) E_2 := it was raining **here** *before*(**today**) = *yesterday*

(3) E_3 := it is going to rain *at* (altitude **here** + 500 m) *after*(**today**) = *tomorrow*

Let's take E_1 from (1) above. Then let us fix **here** to **Sydney** and assume it is a *constant*. In the month of February, 2008, with granularity of day, for every day, we can evaluate E_1 to either *true* or *false*:

```
Tags:   1 2 3 4 5 6 7 8 9 ...
Values: F F T T T F F F T ...
```

If one starts varying the **here** dimension (which could even be broken down to X, Y, Z), one gets a two-dimensional evaluation of E_1:

```
City: /   1 2 3 4 5 6 7 8 9 ...
Sydney    F F T T T F F F T ...
Moscow    F F F F T T T F F ...
Ottawa    F T T T T T F F F ...
```

1.4.3 Lucid Summary

Lucid [WA85, Edw95, AW77b, AW76, AW77a] is a dataflow intensional and functional programming language. In fact, it is a family of languages that are built upon intensional logic (which in turn can be understood as a multidimensional generalization of temporal logic) involving context and demand-driven parallel computation model. A program written in some Lucid dialect is an expression that may have subexpressions that need to be evaluated at certain *context*. Given the set of dimension $D = \{dim_i\}$ in which an expression varies, and a corresponding set of indexes or *tags* defined as placeholders over each dimension, the context is represented as a set of $<dim_i : tag_i>$ mappings and each variable in Lucid, called often a *stream*, is evaluated in that defined context that may also evolve using context operators [TPM07, WAP05b, Wan06]. The generic version of Lucid, GIPL [Paq99], defines two basic operators @ and # to navigate in the contexts (switch and query). The GIPL is the first[1] generic programming language of all intensional languages, defined by the means of only two intensional operators `@` and `#`. It has been proven that other intensional programming languages of the Lucid family can be translated into the GIPL [Paq99]. Since the Lucid family of language thrived around intensional logic that makes the notion of context explicit and central, and recently, a first class value [WAP05b, Wan06, PMT08] that can be passed around as function parameters or as return values and have a set of operators defined upon. We greatly draw on this notion by formalizing our evidence and the stories as a contextual specification of the incident to be tested for consistency against the incident model specification. In our specification model we require more than just atomic context values – we need a higher-order context hierarchy to specify different level of detail of the incident and being able to navigate into the "depth" of such a context. Luckily, such a proposition by has already been made [Mok08] and needs some modifications to the expressions of the cyberforensic context.

[1] The second being Lucx.

JLucid, Objective Lucid, and JOOIP. JLucid [Mok05, GMP05] was a first attempt on intensional arrays and "free Java functions" in the GIPSY. The approach used the Lucid language as the driving main computation, where Java methods were peripheral and could be invoked from the Lucid part, but not the other way around. This was the first instance of hybrid programming within the GIPSY. The semantics of this approach was not completely defined, plus, it was only one-sided view (Lucid-to-Java) of the problem. JLucid did not support objects of any kind, but introduced the wrapper class idea for the free Java methods and served as a precursor to Objective Lucid.

Objective Lucid [Mok05, MP05] is an extension of the JLucid language that inherits all of the JLucid's features and introduced Java objects to be available for use by Lucid. Objective Lucid expanded the notion of the Java object (a collection of members of different types) to the array (a collection of members of the same type) and first introduced the dot-notation in the syntax and operational semantics in GIPSY. Like in JLucid, Objective Lucid's focus was on the Lucid part being the "main" program and did not allow Java to call intensional functions or use intensional constructs from within a Java class. Objective Lucid was the first in GIPSY to introduce the more complete operational semantics of the hybrid OO intensional language.

JOOIP [WPM08] greatly complements Objective Lucid by allowing Java to call the intensional language constructs closing the gap and making JOOIP a complete hybrid OO intensional programming language within the GIPSY environment. JOOIP's semantics further refines in a greater detail the operational semantics rules of Lucid and Objective Lucid in the attempt to make them complete.

MARFL. While not of strictly Lucid family or GIPSY, MARFL [Mok08] was nearly entirely influenced by Lucid, and is based on overloaded `@` and `#` operators as well as allows to navigate into the depth of the higher-order contextual space using the dot operator. The latter was indirectly (re-invented in part) influenced by iHTML and libintense [Swo04, SW00].

1.5 General Intensional Programming System (GIPSY)

The GIPSY [VP05, PK00, The08, Lu04, PGW04, Mok05, WP05, PW05] is a platform implemented primarily in Java to investigate properties of the Lucid family of languages and beyond. It executes Lucid programs following a demand-driven distributed generator-worker architecture, and is designed as a modular collection of frameworks where components related to the development (RIPE[2]), compilation (GIPC[3]), and execution (GEE[4]) of Lucid programs are separated allowing easy extension, addition, and replacement of the components. This is a proposed testing and investigation platform for our *Forensic Lucid* language.

[2]Run-time Integrated Programming Environment, implemented in `gipsy.RIPE`
[3]General Intensional Programming Compiler, implemented in `gipsy.GIPC`
[4]General Eduction Engine, implemented in `gipsy.GEE`

2 Forensic Lucid Requirements and Design

This section lists concepts and considerations in the design of the Forensic Lucid language. The language has been studied through a case study [MPD07, MP08a] and another one is under development [MP08b]. The end goal is to define our Forensic Lucid language where its constructs concisely express cyberforensic evidence, which can be initial state of the case (e.g. initial printer state when purchased from the manufacturer as in [GP04]), towards what we have actually observed as a final state (e.g. when an investigator finds the printer with two queue entries $(B_{deleted}, B_{deleted})$). The implementing system also back-traces intermediate results to provide the corresponding event reconstruction path if it exists. The result of the expression in its basic form is either *true* or *false*, i.e. "guilty" or "not guilty" given the context per explanation with the backtrace. There can be multiple backtraces, that correspond to the explanation of the evidence (or lack thereof).

2.1 Features

We define Forensic Lucid to model the evidential statements and other expressions representing the evidence and observations as a higher-order context. An execution trace of a Forensic Lucid program would expose the possibility of the proposed claim with the events in the middle between the final observed event to the beginning of the events. Forensic Lucid aggregates the features of multiple Lucid dialects mentioned earlier needed for these tasks along with its own extensions.

Addition of the context calculus from Lucx for operators on Lucx's context sets (union, intersection, etc.) are used to address to provide a collection of traces. Forensic Lucid inherits the properties of Lucx, Objective Lucid, JOOIP (and their comprising dialects), where the former is for the context calculus, and the latter for the arrays and structural representation of data for modeling the case data structures such as events, observations, and groupings of the related data.

One of the basic requirements is that the complete definition of the syntax, and the operational semantics of Forensic Lucid should be compatible with the basic Lucx and GIPL, i.e. the translation rules or equivalent are to be provided when implementing the language compiler within GIPSY, and such that the GEE can execute it with minimal changes. The most difficult aspect here is, of course, the semantics of Forensic Lucid (luckily, the bulk of it is an aggregation of the semantic rules of the languages we inherit from).

2.2 Forward Tracing vs. Back-tracing

Naturally, the GEE makes demands in the demand-driven evaluation in the order the tree of an intentional program is traversed. Tracing of the demand requests in this case will be "forward tracing". Such tracing is less useful than the mentioned back-tracing when demands are resolved, when dealing with the back-tracing in forensic investigation in an

attempt to reconstruct events from the final state observations. Back-tracing is also naturally present when demands are computed and return results. The latter may not be sufficient in the forensic evaluation, so a set of reverse operators to `next`, `fby`, `asa`, etc. is needed. The development of such operators is discussed further in the syntax and semantics sections.

2.3 Context

We need to provide an ability to encode the stories told by the evidence and witnesses. This will constitute the context of evaluation. The return value of the evaluation would be a collection of backtraces, which contain the "paths of truth". If a given trace contains all truths values, it's an explanation of a story. If there is no such a path, i.e. the trace, there is no enough supporting evidence of the entire claim to be true.

The context for this task for simplicity of the prototype language can be expressed as integers or strings, to which we attribute some meaning or description. The contexts are finite and can be navigated through in both directions of the index, potentially allowing negative tags in our tag sets of dimensions. Alternatively, our contexts can be a finite set of symbolic labels and their values that can internally be enumerated. This approach will be naturally more appropriate for humans and we have a machinery to so in Lucx's implementation in GIPSY [Ton08, PMT08].

We define streams of observations as our context, that can be a simple context or a context set. In fact, in Forensic Lucid we are defining higher-level dimensions and lower-level dimensions. The highest-level one is the *evidential statement*, which is a finite unordered set of observation sequences. The *observation sequence* is a finite *ordered* set of observations. The *observation* is an "eyewitness" of a particular property along with the duration of the observation. As in the FSA [Gla05, GP04], the observations are a tuples of (P, min, opt) in their generic form. The observations in this form, specifically, the property P can be exploded further into Lucx's context set and further into an atomic simple context [Wan06, TPM07]. Context switching between different observations is done naturally with the Lucid @ context switching operator. Consider some conceptual expression of a storyboard in Listing 1 where anything in `[ ... ]` represents a story, i.e. the context of evaluation. `foo` can be evaluated at multiple contexts (stories), producing a collection of final results (e.g. *true* or *false*) for each story as well as a collection of traces.

```
foo @
{
  [ final observed event, possible initial observed event ],
  [                       ],
  [                       ]
}
```

Listing 1: Intensional Storyboard Expression

While the `[...]` notation here may be confusing with respect to `[dimension:tag]` in

Lucid and more specifically in Lucx [Wan06, TPM07], it is in fact a simple syntactical extension to allow higher-level groups of contexts where this syntactical sugar is later translated to the baseline context constructs. The tentative notation of `{[...],...,[...]}` implies a notion similar to the notion of the "context set" in [Wan06, TPM07] except with the syntactical sugar mentioned earlier where we allow syntactical grouping of properties, observations, observation sequences, and evidential statements as our context sets.

The generic observation sequence can be expanded [GP04] into the context stream using the *min* and *opt* values, where they will translate into index values. Thus, $obs = (A,3,0)(B,2,0)$ expands the property labels A and B into a finite stream of five indexed elements: *AAABB*. Thus, a Forensic Lucid fragment in Listing 2 would return the third A of the *AAABB* context stream in the observation portion of o. Therefore, possible evaluations to check for the properties can be as shown in Figure 1.

```
// Give me observed property at index 2 in the observation sequence obs
o @.obs 2
where
  // Higher-level dimension in the form of (P,min,opt)
  observation o;
  // Equivalent to writing = { A, A, A, B, B };
  observation sequence obs = (A,3,0)(B,2,0);
  where
    // Properties A and B are arrays of computations
    // or any Expressions
    A = [c1,c2,c3,c4];
    B = E;
    ...
  end;
end;
```

Listing 2: Observation Sequence With Duration

The property values of A and B can be anything that context calculus allows. The dimension type `observation sequence` is a finite ordered context tag set [PMT08] that allows an integral "duration" of a given tag property. This may seem like we allow duplicate tag values that are unsound in the classical Lucid semantics; however, we find our way around little further in the text with the implicit index tag. The semantics of the arrays of computations is not a part of either GIPL or Lucx; however, the arrays are provided by JLucid and Objective Lucid. We need the notion of the arrays to evaluate multiple computations at the same context. Having an array of computations is conceptually equivalent of running an a Lucid program under the same context for each array element in a separate instance of the evaluation engine and then the results of those expressions are gathered in one ordered storage within the originating program. Arrays in Forensic Lucid are needed to represent a set of results, or *explanations* of evidential statements, as well as denote some properties of observations. We will explore the notion of arrays in Forensic Lucid much greater detail in the near future work. In the FSA approach computations c_i correspond to the state q and event i that enable transition. For Forensic Lucid, we can have c_i as theoretically any Lucid expression E.

In Figure 1 we are illustrating a possibility to query for the sub-dimension indices by raw

```
Observed property (context): A A A B B
        Sub-dimension index: 0 1 2 3 4

o @.obs 0 = A
o @.obs 1 = A
o @.obs 2 = A
o @.obs 3 = B
o @.obs 4 = B

To get the duration/index position:

o @.obs A = 0 1 2
o @.obs B = 3 4
```

Figure 1: Handling Duration of an Observed Property in the Context

property where it persists that produces a finite stream valid indices that can be used in subsequent expressions, or, alternatively by supplying the index we can get the corresponding raw property at that index. The latter feature is still under investigation of whether it is safe to expose it to Forensic Lucid programmers or make it implicit at all times at the implementation level. This is needed to remedy the problem of "duplicate tags": as previously mentioned, observations form the context and allow durations. This means multiple duplicate dimension tags with implied subdimension indexes should be allowed as the semantics of a traditional Lucid approaches do not allow duplicate dimension tags. It should be noted however, that the combination of the tag and its index in the stream is still unique and can be folded into the traditional Lucid semantics.

2.4 Concrete Forensic Lucid Syntax

The concrete syntax of the Forensic Lucid language is presented in Figure 2. It is influenced by the productions from Lucx [WAP05b, Wan06], JLucid and Objective Lucid [Mok05, GMP05, MP05], and Indexical Lucid [Paq99]. Some of the syntactical definitions can be, perhaps, implemented as a collection of macros. The `evidential statement`, `observation sequence`, and `observation` dimension types can be translated into `dimension` by some translation rules flattening them into simple contexts and context sets. The GIPSY compiler framework (GIPC) allows for the introduction of such semantic translation rules to define new language variants. We will use this feature as much as possible, though some of our syntactic constructs may have some underlying semantic details that cannot be translated into generic Lucid primitives, in which case we need to expand the existing semantics.

```
(01)                E  ::=   id
(02)                    |    E(E,...,E)                        #LUCX
(03)                    |    E[E,...,E](E,...,E)               #GIPL
(04)                    |    if E then E else E fi
(05)                    |    # E
(06)                    |    E @ E  E                          #GIPL
(07)                    |    E @ E                             #LUCX
(08)                    |    E where Q end;
(09)                    |    [E:E,...,E:E]                     #LUCX
(10)                    |    E bin-op E                        #INDEXICAL
(11)                    |    un-op E                           #INDEXICAL
(12)                    |    E i-bin-op E                      #INDEXICAL
(13)                    |    i-un-op E                         #INDEXICAL
(14)                    |    bounds
(15)                    |    embed(URI, METHOD, E, E, ...)     #JLUCID
(16)                    |    E[E,...,E]                        #JLUCID
(17)                    |    [E,...,E]                         #JLUCID
(18)                    |    E.id                              #OBJECTIVE
(19)                    |    E.id(E,...,E)                     #OBJECTIVE

(20)                Q  ::=   dimension id,...,id;
(21)                    |    evidential statement id,...,id [ = ES ];
(22)                    |    observation sequence id,...,id [ = OS ];
(23)                    |    observation id,...,id [ = O ];
(24)                    |    id = E;
(25)                    |    id(id,....,id) = E;               #LUCX
(26)                    |    id[id,...,id](id,....,id) = E;    #GIPL
(27)                    |    E.id = E;                         #OBJECTIVE
(28)                    |    id.id,...,id(id,...,id) = E;      #OBJECTIVE
(29)                    |    QQ

(30)                ES ::=   { OS,...,OS } # evidential statement
(31)                OS ::=   { O,...,O }   # observation sequence
(32)                 O ::=   ( E, E, E )   # (property, min, opt)
                        |    $             # no-observation (Ct, 0, infinitum)
                        |    \0( E )       # zero-observation (P, 0, 0), where P = E

(33)             bin-op ::= arith-op | logical-op | bitwise-op
(34)              un-op ::= + | -

(35)           arith-op ::=  +  | - | *  | /  | %  | ^
(36)         logical-op ::=  <  | > | >= | <= | == | in | && | "||" | !
(37)         bitwise-op ::= "|" | & | ~  | !| | !&

(38)           i-bin-op ::= @ | i-bin-op-forw | i-bin-op-back | i-logic-bitwise-op | i-forensic-op

(39)      i-bin-op-forw ::= fby   |  upon  | asa   | wvr
                          |  nfby  | nupon  | nasa  | nwvr

(40)      i-bin-op-back ::= pby   | rupon  | ala   | rwvr
                          |  npby  | nrupon | nala  | nrwvr

(41) i-logic-bitwise-op ::= and   | or     | xor
                          |  nand  | nor    | nxor
                          |  band  | bor    | bxor

(42)            i-un-op ::= i-bin-un-forw | i-bin-un-back | #

(43)      i-bin-un-forw ::= first  | next  | iseod
                          |  second | nnext | neg | not

(44)      i-bin-un-back ::= last    | prev  | isbod
                          |  prelast | nprev

(45)      i-forensic-op ::= combine | product | psi | invpsi

(46)             bounds ::= eod | bod | +inf | -inf
```

Figure 2: Concrete Forensic Lucid Syntax

2.5 Transition Function

A transition function determines how the context of evaluation changes during computation. A general issue exists that we have to address is that the transition function ψ is problem-specific. In the FSA approach, the transition function is the labeled graph itself. In the first prototype, we follow the graph to model our Forensic Lucid equivalent. In general, Lucid has already basic operators to navigate and switch from one context to another, which represent the basic transition functions in themselves (the intensional operators such as @, #, `iseod`, `first`, `next`, `fby`, `wvr`, `upon`, and `asa` as well as their inverse operators). However, a specific problem being modeled requires more specific transition function than just plain intensional operators. In this case the transition function is a Forensic Lucid function where the matching state transition modeled through a sequence of intensional operators.

A question arises a of how to explicitly model the transition function ψ and its backtrace Ψ^{-1} in the new language. A possible approach is to use predefined macros in Lucid syntax [MPD07]. In fact, the forensic operators are just pre-defined functions that rely on traditional and inverse Lucid operators as well as context switching operators that achieve something similar to the transitions. Once modeled, it would be the GEE actually execution ψ within GIPSY. In fact, the intensional operators of Lucid represent the basic building blocks for ψ and Ψ^{-1}. We provide a first implementation of Ψ^{-1} in [MPD07].

```
alice_claim @ es
where
  evidential statement es = [ printer, manuf, alice ];

  observation sequence printer = F;
  observation sequence manuf = [Oempty, $];
  observation sequence alice = [Oalice, F];

  observation F = (``B_deleted'', 1, 0);
  observation Oalice = (P_alice, 0, +inf);
  observation Oempty = (``empty'', 1, 0);

  // No ``add_A''
  P_alice = unordered {``add_B'', ``take''};

  invpsiacme(F, es);
end;
```

Listing 3: Developing the Pinter Case "main"

2.6 Primitive Operators

The basic set of the classic intensional operators is extended with the similar operators, but inverted in one of their aspects: either negation of trueness or reverse of direction of navigation. Here we provide an informal definition of these operators alongside with the

classical ones (to remind the reader what they do and enlighten the unaware reader). The reverse operators have a restriction that they must work on the bounded streams at the positive infinity. This is not a stringent limitation as the our contexts of observations and evidence in this work are always finite, so they all have the beginning and the end. What we need is an ability to go back in the stream and, perhaps, negate in it with classical-like operators, but reversed. The operators have been defined so far in [MP08a], we only summarize their definitions through the @ and # operators in Figure 3.

2.7 Forensic Operators

The operators presented here are based on the discussion of the combination function and others that form more-than-primitive operations to support the required implementation.

- **`combine`** corresponds to the *comb* function described earlier. It is defined in Listing 4. It is a preliminary context-enhanced version.

```
/**
 * Append given e to each element
 * of a given stream e under the
 * context of d.
 *
 * @return the resulting combined stream
 */
combine(s, e, d) =
  if iseod s then eod;
  else (first s fby.d e) fby.d combine(next s, e, d);
```

Listing 4: The **`combine`** Operator

- **`product`** tentatively corresponds to the cross-product of context, translated from that of the LISP example and added with context. It is defined in Listing 5.

```
/**
 * Append elements of s2 to element of s1
 * in all possible combinations.
 */
product(s1, s2, d) =
  if iseod s2 then eod;
  else combine(s1, first s2) fby.d product(s1, next s2)
```

Listing 5: The **`product`** Operator

The translated examples show recursion that we are not prepared to deal with in the current Lucid semantics, and will address that in the future work. The two illustrated operators are the first of the a few more to follow in the final language prototype.

$$
\begin{aligned}
\texttt{first}\ X &= X@0 && (1)\\
\texttt{last}\ X &= X@(\#@(\#\texttt{iseod}(\#)-1)) && (2)\\
\texttt{next}\ X &= X@(\#+1) && (3)\\
\texttt{prev}\ X &= X@(\#-1) && (4)\\
X\ \texttt{fby}\ Y &= \textbf{if}\ \#=0\ \textbf{then}\ X\ \textbf{else}\ Y@(\#-1) && (5)\\
&= \textbf{if}\ \texttt{isbod}\ X\ \textbf{then}\ X\ \textbf{else}\ \texttt{prev}\ Y\\
X\ \texttt{pby}\ Y &= \textbf{if}\ \texttt{iseod}\ \#\ \textbf{then}\ X\ \textbf{else}\ Y@(\#+1) && (6)\\
&= \textbf{if}\ \texttt{iseod}\ Y\ \textbf{then}\ X\ \textbf{else}\ \texttt{next}\ Y\\
X\ \texttt{wvr}\ Y &= X@T\ \texttt{where} && (7)\\
&\qquad T = U\ \texttt{fby}\ U@(T+1)\\
&\qquad U = \textbf{if}\ Y\ \textbf{then}\ \#\ \textbf{else}\ \texttt{next}\ U\\
&\texttt{end}\\
X\ \texttt{rwvr}\ Y &= X@T\ \texttt{where} && (8)\\
&\qquad T = U\ \texttt{pby}\ U@(T-1)\\
&\qquad U = \textbf{if}\ Y\ \textbf{then}\ \#\ \textbf{else}\ \texttt{prev}\ U\\
&\texttt{end}\\
X\ \texttt{nwvr}\ Y &= X@T\ \texttt{where} && (9)\\
&\qquad T = U\ \texttt{fby}\ U@(T+1)\\
&\qquad U = \textbf{if}\ Y==0\ \textbf{then}\ \#\ \textbf{else}\ \texttt{next}\ U\\
&\texttt{end}\\
X\ \texttt{rnwvr}\ Y &= X@T\ \texttt{where} && (10)\\
&\qquad T = U\ \texttt{pby}\ U@(T-1)\\
&\qquad U = \textbf{if}\ Y==0\ \textbf{then}\ \#\ \textbf{else}\ \texttt{prev}\ U\\
&\texttt{end}\\
X\ \texttt{asa}\ Y &= \texttt{first}\ (X\ \texttt{wvr}\ Y) && (11)\\
X\ \texttt{nasa}\ Y &= \texttt{first}\ (X\ \texttt{nwvr}\ Y) && (12)\\
X\ \texttt{ala}\ Y &= \texttt{last}\ (X\ \texttt{rwvr}\ Y) && (13)\\
X\ \texttt{nala}\ Y &= \texttt{last}\ (X\ \texttt{nrwvr}\ Y) && (14)\\
X\ \texttt{upon}\ Y &= X@W\ \texttt{where} && (15)\\
&\qquad W = 0\ \texttt{fby}\ (\textbf{if}\ Y\ \textbf{then}\ (W+1)\ \textbf{else}\ W)\\
&\texttt{end}\\
X\ \texttt{rupon}\ Y &= X@W\ \texttt{where} && (16)\\
&\qquad W = 0\ \texttt{pby}\ (\textbf{if}\ Y\ \textbf{then}\ (W-1)\ \textbf{else}\ W)\\
&\texttt{end}\\
X\ \texttt{nupon}\ Y &= X@W\ \texttt{where} && (17)\\
&\qquad W = 0\ \texttt{fby}\ (\textbf{if}\ Y==0\ \textbf{then}\ (W+1)\ \textbf{else}\ W)\\
&\texttt{end}\\
X\ \texttt{nrupon}\ Y &= X@W\ \texttt{where} && (18)\\
&\qquad W = 0\ \texttt{pby}\ (\textbf{if}\ Y==0\ \textbf{then}\ (W-1)\ \textbf{else}\ W)\\
&\texttt{end}\\
\texttt{neg}\ X &= -X && (19)\\
\texttt{not}\ X &= \textbf{if}\ X\ \textbf{then}\ !X\ \textbf{else}\ X && (20)\\
X\ \texttt{and}\ Y &= X\&\&Y && (21)\\
X\ \texttt{or}\ Y &= X||Y && (22)\\
X\ \texttt{xor}\ Y &= \texttt{not}((X\ \texttt{and}\ Y)\ \texttt{or}\ \texttt{not}\ (X\ \texttt{or}\ Y)) && (23)
\end{aligned}
$$

Figure 3: Operators Translated to GIPL-Compatible Definitions

2.8 Operational Semantics

As previously mentioned, the operational semantics of Forensic Lucid for the large part is viewed as a composition of the semantic rules of Indexical Lucid, Objective Lucid, and Lucx along with the new operators and definitions. Here we list the existing combined semantic definitions to be used the new language, specifically extracts of operational semantics from GIPL [Paq99], Objective Lucid [Mok05], and Lucx [Wan06] are in Figure 4, Figure 5, and Figure 7 respectively. The explanation of the rules and the notation are given in great detail in the cited works and are trimmed in this article. For convenience of the reader they are recited here. The Objective Lucid semantic rules were affected and refined by some of the semantic rules of JOOIP [WPM08]. The new rules of the operational semantics of Forensic Lucid cover the operators primarily, including the reverse and logical stream operators as well as forensic-specific operators. We use the same notation as the referenced languages to maintain consistency in defining our rules.

In the implementing system, GIPSY, the GIPL is the generic counterpart of all the Lucid programming languages. Like Indexical Lucid, which it is derived from, it has only the two standard intensional operators: `E @ C` for evaluating an expression `E` in context `C`, and `#d` for determining the position in dimension `d` of the current context of evaluation in the context space [Paq99]. SIPLs are Lucid dialects (Specific Intensional Programming Languages) with their own attributes and objectives. Theoretically, all SIPLs can be translated into the GIPL [Paq99]. All the SIPLs conservatively extend the GIPL syntactically and semantically. The remainder of this section presents a relevant piece of Lucx as a conservative extension to GIPL. The semantics of GIPL is presented in Figure 4. The excerpt of semantic rules of Lucx are then presented as a conservative extension to GIPL and Lucx in Figure 7. Following is the description of the GIPL semantic rules as presented in [Paq99]:

$$\mathscr{D} \vdash E : v$$

tells that under the *definition environment* $\mathscr{D}$, expression E would evaluate to value v.

$$\mathscr{D}, \mathscr{P} \vdash E : v$$

specifies that in the definition environment $\mathscr{D}$, and in the *evaluation context* $\mathscr{P}$ (sometimes also referred to as a *point* in the context space), expression E evaluates to v. The definition environment $\mathscr{D}$ retains the definitions of all of the identifiers that appear in a Lucid program, as created with the semantic rules 13-16 in Figure 4. It is therefore a partial function

$$\mathscr{D} : \textbf{Id} \rightarrow \textbf{IdEntry}$$

where **Id** is the set of all possible identifiers and **IdEntry**, has five possible kinds of value, one for each of the kinds of identifier: 1. *Dimensions* define the coordinate pairs, in which one can navigate with the `#` and `@` operators. Their **IdEntry** is simply (`dim`). 2. *Constants* are external entities that provide a single value, regardless of the context of evaluation.

$$\mathbf{E_{cid}} \quad : \quad \frac{\mathscr{D}(id) = (\texttt{const}, c)}{\mathscr{D}, \mathscr{P} \vdash id : c} \tag{24}$$

$$\mathbf{E_{opid}} \quad : \quad \frac{\mathscr{D}(id) = (\texttt{op}, f)}{\mathscr{D}, \mathscr{P} \vdash id : id} \tag{25}$$

$$\mathbf{E_{did}} \quad : \quad \frac{\mathscr{D}(id) = (\texttt{dim})}{\mathscr{D}, \mathscr{P} \vdash id : id} \tag{26}$$

$$\mathbf{E_{fid}} \quad : \quad \frac{\mathscr{D}(id) = (\texttt{func}, id_i, E)}{\mathscr{D}, \mathscr{P} \vdash id : id} \tag{27}$$

$$\mathbf{E_{vid}} \quad : \quad \frac{\mathscr{D}(id) = (\texttt{var}, E) \qquad \mathscr{D}, \mathscr{P} \vdash E : v}{\mathscr{D}, \mathscr{P} \vdash id : v} \tag{28}$$

$$\mathbf{E_{op}} \quad : \quad \frac{\mathscr{D}, \mathscr{P} \vdash E : id \qquad \mathscr{D}(id) = (\texttt{op}, f) \qquad \mathscr{D}, \mathscr{P} \vdash E_i : v_i}{\mathscr{D}, \mathscr{P} \vdash E(E_1, \ldots, E_n) : f(v_1, \ldots, v_n)} \tag{29}$$

$$\mathbf{E_{fct}} \quad : \quad \frac{\mathscr{D}, \mathscr{P} \vdash E : id \qquad \mathscr{D}(id) = (\texttt{func}, id_i, E') \qquad \mathscr{D}, \mathscr{P} \vdash E'[id_i \leftarrow E_i] : v}{\mathscr{D}, \mathscr{P} \vdash E(E_1, \ldots, E_n) : v} \tag{30}$$

$$\mathbf{E_{c_T}} \quad : \quad \frac{\mathscr{D}, \mathscr{P} \vdash E : true \qquad \mathscr{D}, \mathscr{P} \vdash E' : v'}{\mathscr{D}, \mathscr{P} \vdash \texttt{if}\ E\ \texttt{then}\ E'\ \texttt{else}\ E'' : v'} \tag{31}$$

$$\mathbf{E_{c_F}} \quad : \quad \frac{\mathscr{D}, \mathscr{P} \vdash E : false \qquad \mathscr{D}, \mathscr{P} \vdash E'' : v''}{\mathscr{D}, \mathscr{P} \vdash \texttt{if}\ E\ \texttt{then}\ E'\ \texttt{else}\ E'' : v''} \tag{32}$$

$$\mathbf{E_{tag}} \quad : \quad \frac{\mathscr{D}, \mathscr{P} \vdash E : id \qquad \mathscr{D}(id) = (\texttt{dim})}{\mathscr{D}, \mathscr{P} \vdash \#E : \mathscr{P}(id)} \tag{33}$$

$$\mathbf{E_{at}} \quad : \quad \frac{\mathscr{D}, \mathscr{P} \vdash E' : id \qquad \mathscr{D}(id) = (\texttt{dim}) \qquad \mathscr{D}, \mathscr{P} \vdash E'' : v'' \qquad \mathscr{D}, \mathscr{P}\dagger[id \mapsto v''] \vdash E : v}{\mathscr{D}, \mathscr{P} \vdash E\ @E'\ E'' : v} \tag{34}$$

$$\mathbf{E_{w}} \quad : \quad \frac{\mathscr{D}, \mathscr{P} \vdash Q : \mathscr{D}', \mathscr{P}' \qquad \mathscr{D}', \mathscr{P}' \vdash E : v}{\mathscr{D}, \mathscr{P} \vdash E\ \texttt{where}\ Q : v} \tag{35}$$

$$\mathbf{Q_{dim}} \quad : \quad \frac{}{\mathscr{D}, \mathscr{P} \vdash \texttt{dimension}\ id : \mathscr{D}\dagger[id \mapsto (\texttt{dim})], \mathscr{P}\dagger[id \mapsto 0]} \tag{36}$$

$$\mathbf{Q_{id}} \quad : \quad \frac{}{\mathscr{D}, \mathscr{P} \vdash id = E : \mathscr{D}\dagger[id \mapsto (\texttt{var}, E)], \mathscr{P}} \tag{37}$$

$$\mathbf{Q_{fid}} \quad : \quad \frac{}{\mathscr{D}, \mathscr{P} \vdash id(id_1, \ldots, id_n) = E : \mathscr{D}\dagger[id \mapsto (\texttt{func}, id_i, E)], \mathscr{P}} \tag{38}$$

$$\mathbf{QQ} \quad : \quad \frac{\mathscr{D}, \mathscr{P} \vdash Q : \mathscr{D}', \mathscr{P}' \qquad \mathscr{D}', \mathscr{P}' \vdash Q' : \mathscr{D}'', \mathscr{P}''}{\mathscr{D}, \mathscr{P} \vdash Q\ Q' : \mathscr{D}'', \mathscr{P}''} \tag{39}$$

Figure 4: GIPL Semantics

Examples are integers and Boolean values. Their **IdEntry** is $(\texttt{const}, c)$, where c is the value of the constant. 3. *Data operators* are external entities that provide memoryless functions. Examples are the arithmetic and Boolean functions. The constants and data operators are said to define the *basic algebra* of the language. Their **IdEntry** is $(\texttt{op}, f)$,

$$\mathbf{E_{c-vid}} \quad : \quad \frac{\begin{array}{c}\mathscr{D},\mathscr{P}\vdash E : id \quad \mathscr{D},\mathscr{P}\vdash E' : id' \\ \mathscr{D}(id) = (\texttt{class, cid, }\underline{\texttt{cdef}}) \quad \mathscr{D}(id') = (\texttt{classv, cid.cvid, }\underline{\texttt{vdef}}) \\ \mathscr{D},\mathscr{P}\vdash \texttt{<cid.cvid>}: v\end{array}}{\mathscr{D},\mathscr{P}\vdash E.E' : v} \tag{40}$$

$$\mathbf{E_{c-fct}} \quad : \quad \frac{\begin{array}{c}\mathscr{D},\mathscr{P}\vdash E : id \quad \mathscr{D},\mathscr{P}\vdash E' : id' \quad \mathscr{D},\mathscr{P}\vdash E_1,\ldots,E_n : v_1,\ldots,v_n \\ \mathscr{D}(id) = (\texttt{class, cid, }\underline{\texttt{cdef}}) \quad \mathscr{D}(id') = (\texttt{classf, cid.cfid, }\underline{\texttt{fdef}}) \\ \mathscr{D},\mathscr{P}\vdash \texttt{<cid.cfid}(v_1,\ldots,v_n)\texttt{>}: v\end{array}}{\mathscr{D},\mathscr{P}\vdash E.E'(E_1,\ldots,E_n) : v} \tag{41}$$

$$\mathbf{E_{ffid}} \quad : \quad \frac{\begin{array}{c}\mathscr{D},\mathscr{P}\vdash E : id \quad \mathscr{D},\mathscr{P}\vdash E_1,\ldots,E_n : v_1,\ldots,v_n \\ \mathscr{D}(id) = (\texttt{freefun, ffid, }\underline{\texttt{ffdef}}) \\ \mathscr{D},\mathscr{P}\vdash \texttt{<ffid}(v_1,\ldots,v_n)\texttt{>}: v\end{array}}{\mathscr{D},\mathscr{P}\vdash E(E_1,\ldots,E_n) : v} \tag{42}$$

$$\mathbf{\#JAVA_{objid}} \quad : \quad \frac{\underline{\texttt{cdef}} = \texttt{Class cid }\{\ldots\}}{\mathscr{D},\mathscr{P}\vdash \underline{\texttt{cdef}} : \mathscr{D}\dagger[\texttt{cid}\mapsto(\texttt{class, cid, }\underline{\texttt{cdef}})],\mathscr{P}} \tag{43}$$

$$\mathbf{\#JAVA_{objvid}} \quad : \quad \frac{\underline{\texttt{cdef}} = \texttt{Class cid }\{\ldots\underline{\texttt{vdef}}\ldots\} \qquad \underline{\texttt{vdef}} = \texttt{public }\mathit{type}\texttt{ vid;}}{\mathscr{D},\mathscr{P}\vdash \underline{\texttt{cdef}} : \mathscr{D}\dagger[\texttt{cid.vid}\mapsto(\texttt{classv, cid.vid, }\underline{\texttt{vdef}})],\mathscr{P}} \tag{44}$$

$$\mathbf{\#JAVA_{objfid}} \quad : \quad \frac{\underline{\texttt{cdef}} = \texttt{Class cid }\{\ldots\underline{\texttt{fdef}}\ldots\} \qquad \underline{\texttt{fdef}} = \texttt{public }\mathit{frttype}\ \ \texttt{ffid}(\mathit{fargtype_1}\ \mathit{farg_{id_1}},\ldots,\mathit{fargtype_n}\ \mathit{farg_{id_n}})}{\mathscr{D},\mathscr{P}\vdash \underline{\texttt{cdef}} : \mathscr{D}\dagger[\texttt{cid.fid}\mapsto(\texttt{classf, cid.fid, }\underline{\texttt{fdef}})],\mathscr{P}} \tag{45}$$

$$\mathbf{\#JAVA_{ffid}} \quad : \quad \frac{\underline{\texttt{ffdef}} = \mathit{frttype}\ \ \texttt{ffid}(\mathit{fargtype_1}\ \mathit{farg_{id_1}},\ldots,\mathit{fargtype_n}\ \mathit{farg_{id_n}})}{\mathscr{D},\mathscr{P}\vdash \underline{\texttt{ffdef}} : \mathscr{D}\dagger[\texttt{ffid}\mapsto(\texttt{freefun, ffid, }\underline{\texttt{ffdef}})],\mathscr{P}} \tag{46}$$

Figure 5: Extract of Operational Semantics of Objective Lucid

$$\mathbf{E_{E.did}} \quad : \quad \frac{\mathscr{D}(E.id) = (\texttt{dim})}{\mathscr{D},\mathscr{P}\vdash E.id : id.id} \tag{47}$$

Figure 6: Higher-Order Context Dot Operator of MARFL

where f is the function itself. 4. *Variables* carry the multidimensional streams. Their **IdEntry** is $(\texttt{var}, E)$, where E is the Lucid expression defining the variable. It should be noted that this semantics makes the assumption that all variable names are unique. This constraint is easy to overcome by performing compile-time renaming or using a nesting level environment scope when needed. 5. *Functions* are non-recursive GIPL user-defined functions. Their **IdEntry** is $(\texttt{func}, id_i, E)$, where the id_i are the formal parameters to the function and E is the body of the function. In this paper we do not discuss the semantics of recursive functions.

The evaluation context $\mathscr{P}$, which is changed when the `@` operator is evaluated, or a dimension is declared in a `where` clause, associates a *tag* (i.e. an index) to each relevant dimension. It is, therefore, a partial function

$$\mathscr{P} : \mathbf{Id} \rightarrow \mathbf{N}$$

Each type of identifiers can only be used in the appropriate situations. Identifiers of type `op`, `func`, and `dim` evaluate to themselves (Figure 4, rules 25,26,27). Constant

$$\mathbf{E_{\#(cxt)}} \quad : \quad \frac{}{\mathscr{D},\mathscr{P} \vdash \# : \mathscr{P}} \tag{48}$$

$$\mathbf{E_{construction(cxt)}} \quad : \quad \frac{\begin{array}{l}\mathscr{D},\mathscr{P} \vdash E_{d_j} : id_j \qquad \mathscr{D}(id_j) = (\texttt{dim}) \\ \mathscr{D},\mathscr{P} \vdash E_{i_j} : v_j \qquad \mathscr{P}' = \mathscr{P}_0 \dagger [id_1 \mapsto v_1] \dagger \ldots \dagger [id_n \mapsto v_n]\end{array}}{\mathscr{D},\mathscr{P} \vdash [E_{d_1} : E_{i_1}, E_{d_2} : E_{i_2}, \ldots, E_{d_n} : E_{i_n}] : \mathscr{P}'} \tag{49}$$

$$\mathbf{E_{at(cxt)}} \quad : \quad \frac{\mathscr{D},\mathscr{P} \vdash E' : \mathscr{P}' \qquad \mathscr{D},\mathscr{P} \dagger \mathscr{P}' \vdash E : v}{\mathscr{D},\mathscr{P} \vdash E \ @ \ E' : v} \tag{50}$$

$$\mathbf{E_{.}} \quad : \quad \frac{\mathscr{D},\mathscr{P} \vdash E_2 : id_2 \qquad \mathscr{D}(id_2) = (\texttt{dim})}{\mathscr{D},\mathscr{P} \vdash E_1.E_2 : tag(E_1 \downarrow \{id_2\})} \tag{51}$$

$$\mathbf{E_{tuple}} \quad : \quad \frac{\mathscr{D},\mathscr{P} \vdash E : id \qquad \mathscr{D} \dagger [id \mapsto (\texttt{dim})] \qquad \mathscr{P} \dagger [id \mapsto 0] \qquad \mathscr{D},\mathscr{P} \vdash E_i : v_i}{\mathscr{D},\mathscr{P} \vdash \langle E_1, E_2, \ldots, E_n \rangle E : v_1 \ \mathit{fby.id} \ v_2 \ \mathit{fby.id} \ \ldots \ v_n \ \mathit{fby.id} \ \texttt{eod}} \tag{52}$$

$$\mathbf{E_{select}} \quad : \quad \frac{E = [\texttt{d} : \texttt{v}'] \qquad E' = \langle \texttt{E}_1, \ldots, \texttt{E}_n \rangle \texttt{d} \mathscr{P}' = \mathscr{P} \dagger [d \mapsto v'] \qquad \mathscr{D},\mathscr{P}' \vdash E' : v}{\mathscr{D},\mathscr{P} \vdash select(E,E') : v} \tag{53}$$

$$\mathbf{E_{at(s)}} \quad : \quad \frac{\mathscr{D},\mathscr{P} \vdash \mathscr{C} : \{\mathscr{P}_1, \ldots, \mathscr{P}_2\} \qquad \mathscr{D},\mathscr{P}_{i:1\ldots m} \vdash E : v_i}{\mathscr{D},\mathscr{P} \vdash E \ @C : \{v_1, \ldots, v_m\}} \tag{54}$$

$$\mathbf{C_{box}} \quad : \quad \frac{\begin{array}{l}\mathscr{D},\mathscr{P} \vdash E_{d_i} : id_i \qquad \mathscr{D}(id_i) = (\texttt{dim}) \\ \{E_1, \ldots, E_n\} = dim(\mathscr{P}_1) = \ldots = dim(\mathscr{P}_m) \\ E' = \texttt{f}_p(\texttt{tag}(\mathscr{P}_1), \ldots, \texttt{tag}(\mathscr{P}_m)) \qquad \mathscr{D},\mathscr{P} \vdash E' : true\end{array}}{\mathscr{D},\mathscr{P} \vdash Box[E_1, \ldots, E_n | E'] : \{\mathscr{P}_1, \ldots, \mathscr{P}_m\}} \tag{55}$$

$$\mathbf{C_{set}} \quad : \quad \frac{\mathscr{D},\mathscr{P} \vdash E_{w:1\ldots m} : \mathscr{P}_m}{\mathscr{D},\mathscr{P} \vdash \{E_1, \ldots, E_m\} : \{\mathscr{P}_1, \ldots, \mathscr{P}_w\}} \tag{56}$$

$$\mathbf{C_{op}} \quad : \quad \frac{\mathscr{D},\mathscr{P} \vdash E : id \qquad \mathscr{D}(id) = (\texttt{cop}, f) \qquad \mathscr{D},\mathscr{P} \vdash C_i : v_i}{\mathscr{D},\mathscr{P} \vdash E(C_1, \ldots, C_n) : f(v_1, \ldots, v_n)} \tag{57}$$

$$\mathbf{C_{sop}} \quad : \quad \frac{\mathscr{D},\mathscr{P} \vdash E : id \qquad \mathscr{D}(id) = (\texttt{sop}, f) \qquad \mathscr{D},\mathscr{P} \vdash C_i : \{v_{i_1}, \ldots, v_{i_k}\}}{\mathscr{D},\mathscr{P} \vdash E(C_1, \ldots, C_n) : f(\{v_{1_1}, \ldots, v_{1_s}\}, \ldots, \{v_{n_1}, \ldots, v_{n_m}\})} \tag{58}$$

Figure 7: Extract of Operational Semantics of Lucx

identifiers (`const`) evaluate to the corresponding constant (Figure 4, rule 24). Function calls, resolved by the $\mathbf{E_{fct}}$ rule (Figure 4, rule 30), require the renaming of the formal parameters into the actual parameters (as represented by $E'[id_i \leftarrow E_i]$). The function $\mathscr{P}' = \mathscr{P} \dagger [id \mapsto v'']$ specifies that $\mathscr{P}'(x)$ is v'' if $x = id$, and $\mathscr{P}(x)$ otherwise. The rule for the `where` clause, $\mathbf{E_w}$ (Figure 4, rule 35), which corresponds to the syntactic expression E `where` Q, evaluates E using the definitions Q therein. The additions to the definition environment $\mathscr{D}$ and context of evaluation $\mathscr{P}$ made by the **Q** rules (Figure 4, rules 36,37,38) are local to the current `where` clause. This is represented by the fact that the $\mathbf{E_w}$ rule returns neither $\mathscr{D}$ nor $\mathscr{P}$. The $\mathbf{Q_{dim}}$ rule adds a dimension to the definition environment and, as a convention, adds this dimension to the context of evaluation with tag 0 (Figure 4, rule 36). The $\mathbf{Q_{id}}$ and $\mathbf{Q_{fid}}$ simply add variable and function identifiers along with their definition to the definition environment (Figure 4, rules 37,38).

As a conservative extension to GIPL, Lucx's semantics introduces the notion of *context* as a building block into the semantic rules, i.e. *context as a first-class value*, as described by the rules in Figure 7. In Lucx, semantic rule 49 (Figure 7) creates a context as a semantic

item and returns it as a context $\mathscr{P}$ that can then be used by rule 50 to navigate to this context by making it override the current context. GIPL's semantic rule 29 is still valid for the definition of the context operators, where the actual parameters evaluate to values v_i that are contexts $\mathscr{P}_i$. The semantic rule 48 expresses that the # symbol evaluates to the current context. When used as a parameter to the context calculus operators, this allows for the generation of contexts relative to the current context of evaluation.

3 Conclusion

Through the series of discussions, definitions of the syntax, semantics, and some examples of the Forensic Lucid we believe we are on the right track to show the benefits of the intensional approach to the cyberforensic, which is promising to be more usable and can be improved even further by providing a graphical DFG editor for the investigators. As far as implementing system concerned it has advantages of parallelizing the computation and introduce the notion of context that is absent in the FSA approach of Gladyshev et al. [Gla05, GP04]. We took a lot of advantages of the existing concepts, syntax and semantic rules and constructs from the intensional programming and the Lucid family of languages that are in place, whose theory for the most part are said to be correct. We are gearing towards completion of the design of the Forensic Lucid language and its compiler and run-time environment based on the GIPSY system and its multi-tier architecture [Paq08].

The proposed practical approach in the cyberforensics field can also be used in a normal investigation process involving crimes not necessarily associated with information technology.

3.1 Future Work

The near-future work will consist primarily of the following items:

- Complete semantics of all the mentioned Lucid dialects and their formalization with Isabelle.
- Implementation of the Forensic Lucid compiler, run-time and interactive development environments.

4 Acknowledgments

This work was funded in part by NSERC and the Faculty of Engineering and Computer Science, Concordia University. Thanks to many of the GIPSY project team members for their valuable contributions, suggestions, and reviews, including Dr. Peter Grogono, Xin Tong, Aihua Wu, Emil Vassev, and Amir Pourteymour.

References

[Agi95] I. Agi. GLU for multidimensional signal processing. In *ISLIP'95: The Eighth International Symposium on Languages for Intensional Programming, Sydney, Australia*, 1995.

[AW76] Edward A. Ashcroft and William W. Wadge. Lucid - A Formal System for Writing and Proving Programs. volume 5. SIAM J. Comput. no. 3, 1976.

[AW77a] Edward A. Ashcroft and William W. Wadge. Erratum: Lucid - A Formal System for Writing and Proving Programs. volume 6(1):200. SIAM J. Comput., 1977.

[AW77b] Edward A. Ashcroft and William W. Wadge. Lucid, a nonprocedural language with iteration. *Communication of the ACM*, 20(7):519–526, July 1977.

[Du94] Weichang Du. Object-oriented Implementation of Intensional Language. In *Proceedings of the 7th International Symposium on Lucid and Intensional Programming*, pages 37–45. SRI International, Menlo Park, California, USA, September 1994.

[Edw95] Edward Ashcroft and Anthony Faustini and Raganswamy Jagannathan and William Wadge. *Multidimensional, Declarative Programming*. Oxford University Press, London, 1995.

[FB91] B. Freeman-Benson. Lobjcid: Objects in Lucid. In *Proceedings of the 1991 Symposium on Lucid and Intensional Programming*, pages 80–87. SRI International, Menlo Park, California, USA, April 1991.

[Gla05] Pavel Gladyshev. Finite State Machine Analysis of a Blackmail Investigation. In *International Journal of Digital Evidence*. Technical and Security Risk Services, Sprint 2005, Volume 4, Issue 1, 2005.

[GMP05] Peter Grogono, Serguei Mokhov, and Joey Paquet. Towards JLucid, Lucid with Embedded Java Functions in the GIPSY. In *Proceedings of the 2005 International Conference on Programming Languages and Compilers (PLC 2005), Las Vegas, USA*, pages 15–21. CSREA Press, June 2005.

[GP99] Jean-Raymond Gagné and John Plaice. Demand-Driven Real-Time Computing. World Scientific, September 1999.

[GP04] Pavel Gladyshev and Ahmed Patel. Finite State Machine Approach to Digital Event Reconstruction. In *Digital Investigation Journal*, volume 2, 2004.

[Lu04] Bo Lu. *Developing the Distributed Component of a Framework for Processing Intensional Programming Languages*. PhD thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, March 2004.

[Mok04] Serguei A. Mokhov. Lucid, the Intensional Programming Language and its Semantics in PVS. Semantics of Programming Languages Course Project Report, April 2004.

[Mok05] Serguei A. Mokhov. Towards Hybrid Intensional Programming with JLucid, Objective Lucid, and General Imperative Compiler Framework in the GIPSY. Master's thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, October 2005. ISBN 0494102934.

[Mok07a] Serguei Mokhov. *Intensional Cyberforensics – a PhD Proposal*. Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, December 2007.

[Mok07b] Serguei Mokhov. Intensional Forensics – the Use of Intensional Logic in Cyberforensics. Technical report, Concordia Institute for Information Systems Engineering, Concordia University, Montreal, Canada, January 2007. ENGR6991 Technical Report.

[Mok08] Serguei A. Mokhov. Towards Syntax and Semantics of Hierarchical Contexts in Multimedia Processing Applications using MARFL. In *Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference (COMPSAC)*, pages 1288–1294, Turku, Finland, July 2008. IEEE Computer Society.

[MP05] Serguei Mokhov and Joey Paquet. Objective Lucid – First Step in Object-Oriented Intensional Programming in the GIPSY. In *Proceedings of the 2005 International Conference on Programming Languages and Compilers (PLC 2005), Las Vegas, USA*, pages 22–28. CSREA Press, June 2005.

[MP08a] Serguei A. Mokhov and Joey Paquet. Formally Specifying and Proving Operational Aspects of Forensic Lucid in Isabelle. Technical Report 2008-1-Ait Mohamed, Department of Electrical and Computer Engineering, Concordia University, August 2008. In Theorem Proving in Higher Order Logics (TPHOLs2008): Emerging Trends Proceedings.

[MP08b] Serguei A. Mokhov and Joey Paquet. Using the General Intensional Programming System (GIPSY) for Evaluation of Higher-Order Intensional Logic (HOIL) Expressions. Submitted for publication at SAC'09, 2008.

[MPD07] Serguei A. Mokhov, Joey Paquet, and Mourad Debbabi. Designing a Language for Intensional Cyberforensic Analysis. Unpublished, 2007.

[Paq99] Joey Paquet. *Scientific Intensional Programming*. PhD thesis, Department of Computer Science, Laval University, Sainte-Foy, Canada, 1999.

[Paq08] Joey Paquet. A Multi-Tier Architecture for the Distributed Eductive Execution of Hybrid Intensional Programs. Submitted for publication at SAC'09, 2008.

[PGW04] Joey Paquet, Peter Grogono, and Ai Hua Wu. Towards a Framework for the General Intensional Programming Compiler in the GIPSY. In *Proceedings of the 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2004). Vancouver, Canada.* ACM, October 2004.

[PK00] Joey Paquet and Peter Kropf. The GIPSY Architecture. In *Proceedings of Distributed Computing on the Web, Quebec City, Canada*, 2000.

[PMT08] Joey Paquet, Serguei A. Mokhov, and Xin Tong. Design and Implementation of Context Calculus in the GIPSY Environment. In *Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference (COMPSAC)*, pages 1278–1283, Turku, Finland, July 2008. IEEE Computer Society.

[PN07] Lawrence C. Paulson and Tobias Nipkow. Isabelle: A Generic Proof Assistant. University of Cambridge and Technical University of Munich, 2007. `http://isabelle.in.tum.de/`, last viewed: December 2007.

[PW05] Joey Paquet and Ai Hua Wu. GIPSY – A Platform for the Investigation on Intensional Programming Languages. In *Proceedings of the 2005 International Conference on Programming Languages and Compilers (PLC 2005), Las Vegas, USA*, pages 8–14. CSREA Press, June 2005.

[Ron94] Panagiotis Rondogiannis. *Higher-Order Functional Languages and Intensional Logic*. PhD thesis, Department of Computer Science, University of Victoria, Victoria, Canada, 1994.

[SW00] Paul Swoboda and William W. Wadge. Vmake, ISE, and IRCS: General Tools for the Intensionalization of Software Systems. In M. Gergatsoulis and P. Rondogiannis, editors, *Intensional Programming II*. World-Scientific, 2000.

[Swo04] Paul Swoboda. *A Formalisation and Implementation of Distributed Intensional Programming*. PhD thesis, The University of New South Wales, Sydney, Australia, 2004.

[The08] The GIPSY Research and Development Group. The General Intensional Programming System (GIPSY) Project. Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, 2002-2008. `http://newton.cs.concordia.ca/~gipsy/`, last viewed April 2008.

[Ton08] Xin Tong. Design and Implementation of Context Calculus in the GIPSY. Master's thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, April 2008.

[TPM07] Xin Tong, Joey Paquet, and Serguei A. Mokhov. Context Calculus in the GIPSY. Unpublished, 2007.

[VP05] Emil Vassev and Joey Paquet. A Generic Framework for Migrating Demands in the GIPSY's Demand-Driven Execution Engine. In *Proceedings of the 2005 International Conference on Programming Languages and Compilers (PLC 2005), Las Vegas, USA*, pages 29–35. CSREA Press, June 2005.

[WA85] William Wadge and Edward Ashcroft. *Lucid, the Dataflow Programming Language*. Academic Press, London, 1985.

[Wan06] Kaiyu Wan. *Lucx: Lucid Enriched with Context*. PhD thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, 2006.

[WAP05a] Kaiyu Wan, Vasu Alagar, and Joey Paquet. A Context theory for Intensional Programming. In *Workshop on Context Representation and Reasoning (CRR05), Paris, France*, July 2005.

[WAP05b] Kaiyu Wan, Vasu Alagar, and Joey Paquet. Lucx: Lucid Enriched with Context. In *Proceedings of the 2005 International Conference on Programming Languages and Compilers (PLC 2005), Las Vegas, USA*, pages 48–14. CSREA Press, June 2005.

[WP05] Ai Hua Wu and Joey Paquet. Object-Oriented Intensional Programming in the GIPSY: Preliminary Investigations. In *Proceedings of the 2005 International Conference on Programming Languages and Compilers (PLC 2005), Las Vegas, USA*, pages 43–47. CSREA Press, June 2005.

[WPM08] Aihua Wu, Joey Paquet, and Serguei A. Mokhov. Object-Oriented Intensional Programming: Intensional Classes Using Java and Lucid. Submitted for publication to SAC'09, 2008.

[Zha97] Q. Zhao. Implementation of an object-oriented intensional programming system. Master's thesis, Department of Computer Science, University of New Brunswick, Canada, 1997.

GI-Edition Lecture Notes in Informatics

P-1 Gregor Engels, Andreas Oberweis, Albert Zündorf (Hrsg.): Modellierung 2001.

P-2 Mikhail Godlevsky, Heinrich C. Mayr (Hrsg.): Information Systems Technology and its Applications, ISTA'2001.

P-3 Ana M. Moreno, Reind P. van de Riet (Hrsg.): Applications of Natural Language to Information Systems, NLDB'2001.

P-4 H. Wörn, J. Mühling, C. Vahl, H.-P. Meinzer (Hrsg.): Rechner- und sensorgestützte Chirurgie; Workshop des SFB 414.

P-5 Andy Schürr (Hg.): OMER – Object-Oriented Modeling of Embedded Real-Time Systems.

P-6 Hans-Jürgen Appelrath, Rolf Beyer, Uwe Marquardt, Heinrich C. Mayr, Claudia Steinberger (Hrsg.): Unternehmen Hochschule, UH'2001.

P-7 Andy Evans, Robert France, Ana Moreira, Bernhard Rumpe (Hrsg.): Practical UML-Based Rigorous Development Methods – Countering or Integrating the extremists, pUML'2001.

P-8 Reinhard Keil-Slawik, Johannes Magenheim (Hrsg.): Informatikunterricht und Medienbildung, INFOS'2001.

P-9 Jan von Knop, Wilhelm Haverkamp (Hrsg.): Innovative Anwendungen in Kommunikationsnetzen, 15. DFN Arbeitstagung.

P-10 Mirjam Minor, Steffen Staab (Hrsg.): 1st German Workshop on Experience Management: Sharing Experiences about the Sharing Experience.

P-11 Michael Weber, Frank Kargl (Hrsg.): Mobile Ad-Hoc Netzwerke, WMAN 2002.

P-12 Martin Glinz, Günther Müller-Luschnat (Hrsg.): Modellierung 2002.

P-13 Jan von Knop, Peter Schirmbacher and Viljan Mahni_ (Hrsg.): The Changing Universities – The Role of Technology.

P-14 Robert Tolksdorf, Rainer Eckstein (Hrsg.): XML-Technologien für das Semantic Web – XSW 2002.

P-15 Hans-Bernd Bludau, Andreas Koop (Hrsg.): Mobile Computing in Medicine.

P-16 J. Felix Hampe, Gerhard Schwabe (Hrsg.): Mobile and Collaborative Busi-ness 2002.

P-17 Jan von Knop, Wilhelm Haverkamp (Hrsg.): Zukunft der Netze –Die Verletzbarkeit meistern, 16. DFN Arbeitstagung.

P-18 Elmar J. Sinz, Markus Plaha (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2002.

P-19 Sigrid Schubert, Bernd Reusch, Norbert Jesse (Hrsg.): Informatik bewegt – Informatik 2002 – 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI) 30.Sept.-3.Okt. 2002 in Dortmund.

P-20 Sigrid Schubert, Bernd Reusch, Norbert Jesse (Hrsg.): Informatik bewegt – Informatik 2002 – 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI) 30.Sept.-3.Okt. 2002 in Dortmund (Ergänzungsband).

P-21 Jörg Desel, Mathias Weske (Hrsg.): Promise 2002: Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen.

P-22 Sigrid Schubert, Johannes Magenheim, Peter Hubwieser, Torsten Brinda (Hrsg.): Forschungsbeiträge zur "Didaktik der Informatik" – Theorie, Praxis, Evaluation.

P-23 Thorsten Spitta, Jens Borchers, Harry M. Sneed (Hrsg.): Software Management 2002 – Fortschritt durch Beständigkeit

P-24 Rainer Eckstein, Robert Tolksdorf (Hrsg.): XMIDX 2003 – XML-Technologien für Middleware – Middleware für XML-Anwendungen

P-25 Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Commerce – Anwendungen und Perspektiven – 3. Workshop Mobile Commerce, Universität Augsburg, 04.02.2003

P-26 Gerhard Weikum, Harald Schöning, Erhard Rahm (Hrsg.): BTW 2003: Datenbanksysteme für Business, Technologie und Web

P-27 Michael Kroll, Hans-Gerd Lipinski, Kay Melzer (Hrsg.): Mobiles Computing in der Medizin

P-28 Ulrich Reimer, Andreas Abecker, Steffen Staab, Gerd Stumme (Hrsg.): WM 2003: Professionelles Wissensmanagement – Erfahrungen und Visionen

P-29 Antje Düsterhöft, Bernhard Thalheim (Eds.): NLDB'2003: Natural Language Processing and Information Systems

P-30 Mikhail Godlevsky, Stephen Liddle, Heinrich C. Mayr (Eds.): Information Systems Technology and its Applications

P-31 Arslan Brömme, Christoph Busch (Eds.): BIOSIG 2003: Biometric and Electronic Signatures

P-32 Peter Hubwieser (Hrsg.): Informatische Fachkonzepte im Unterricht – INFOS 2003

P-33 Andreas Geyer-Schulz, Alfred Taudes (Hrsg.): Informationswirtschaft: Ein Sektor mit Zukunft

P-34 Klaus Dittrich, Wolfgang König, Andreas Oberweis, Kai Rannenberg, Wolfgang Wahlster (Hrsg.): Informatik 2003 – Innovative Informatikanwendungen (Band 1)

P-35 Klaus Dittrich, Wolfgang König, Andreas Oberweis, Kai Rannenberg, Wolfgang Wahlster (Hrsg.): Informatik 2003 – Innovative Informatikanwendungen (Band 2)

P-36 Rüdiger Grimm, Hubert B. Keller, Kai Rannenberg (Hrsg.): Informatik 2003 – Mit Sicherheit Informatik

P-37 Arndt Bode, Jörg Desel, Sabine Rathmayer, Martin Wessner (Hrsg.): DeLFI 2003: e-Learning Fachtagung Informatik

P-38 E.J. Sinz, M. Plaha, P. Neckel (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2003

P-39 Jens Nedon, Sandra Frings, Oliver Göbel (Hrsg.): IT-Incident Management & IT-Forensics – IMF 2003

P-40 Michael Rebstock (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2004

P-41 Uwe Brinkschulte, Jürgen Becker, Dietmar Fey, Karl-Erwin Großpietsch, Christian Hochberger, Erik Maehle, Thomas Runkler (Edts.): ARCS 2004 – Organic and Pervasive Computing

P-42 Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Economy – Transaktionen und Prozesse, Anwendungen und Dienste

P-43 Birgitta König-Ries, Michael Klein, Philipp Obreiter (Hrsg.): Persistance, Scalability, Transactions – Database Mechanisms for Mobile Applications

P-44 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): Security, E-Learning. E-Services

P-45 Bernhard Rumpe, Wofgang Hesse (Hrsg.): Modellierung 2004

P-46 Ulrich Flegel, Michael Meier (Hrsg.): Detection of Intrusions of Malware & Vulnerability Assessment

P-47 Alexander Prosser, Robert Krimmer (Hrsg.): Electronic Voting in Europe – Technology, Law, Politics and Society

P-48 Anatoly Doroshenko, Terry Halpin, Stephen W. Liddle, Heinrich C. Mayr (Hrsg.): Information Systems Technology and its Applications

P-49 G. Schiefer, P. Wagner, M. Morgenstern, U. Rickert (Hrsg.): Integration und Datensicherheit – Anforderungen, Konflikte und Perspektiven

P-50 Peter Dadam, Manfred Reichert (Hrsg.): INFORMATIK 2004 – Informatik verbindet (Band 1) Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 20.-24. September 2004 in Ulm

P-51 Peter Dadam, Manfred Reichert (Hrsg.): INFORMATIK 2004 – Informatik verbindet (Band 2) Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 20.-24. September 2004 in Ulm

P-52 Gregor Engels, Silke Seehusen (Hrsg.): DELFI 2004 – Tagungsband der 2. e-Learning Fachtagung Informatik

P-53 Robert Giegerich, Jens Stoye (Hrsg.): German Conference on Bioinformatics – GCB 2004

P-54 Jens Borchers, Ralf Kneuper (Hrsg.): Softwaremanagement 2004 – Outsourcing und Integration

P-55 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): E-Science und Grid Adhoc-Netze Medienintegration

P-56 Fernand Feltz, Andreas Oberweis, Benoit Otjacques (Hrsg.): EMISA 2004 – Informationssysteme im E-Business und E-Government

P-57 Klaus Turowski (Hrsg.): Architekturen, Komponenten, Anwendungen

P-58 Sami Beydeda, Volker Gruhn, Johannes Mayer, Ralf Reussner, Franz Schweiggert (Hrsg.): Testing of Component-Based Systems and Software Quality

P-59 J. Felix Hampe, Franz Lehner, Key Pousttchi, Kai Ranneberg, Klaus Turowski (Hrsg.): Mobile Business – Processes, Platforms, Payments

P-60 Steffen Friedrich (Hrsg.): Unterrichtskonzepte für inforrmatische Bildung

P-61 Paul Müller, Reinhard Gotzhein, Jens B. Schmitt (Hrsg.): Kommunikation in verteilten Systemen

P-62 Federrath, Hannes (Hrsg.): „Sicherheit 2005“ – Sicherheit – Schutz und Zuverlässigkeit

P-63 Roland Kaschek, Heinrich C. Mayr, Stephen Liddle (Hrsg.): Information Systems – Technology and ist Applications

P-64 Peter Liggesmeyer, Klaus Pohl, Michael Goedicke (Hrsg.): Software Engineering 2005

P-65 Gottfried Vossen, Frank Leymann, Peter Lockemann, Wolffried Stucky (Hrsg.): Datenbanksysteme in Business, Technologie und Web

P-66 Jörg M. Haake, Ulrike Lucke, Djamshid Tavangarian (Hrsg.): DeLFI 2005: 3. deutsche e-Learning Fachtagung Informatik

P-67 Armin B. Cremers, Rainer Manthey, Peter Martini, Volker Steinhage (Hrsg.): INFORMATIK 2005 – Informatik LIVE (Band 1)

P-68 Armin B. Cremers, Rainer Manthey, Peter Martini, Volker Steinhage (Hrsg.): INFORMATIK 2005 – Informatik LIVE (Band 2)

P-69 Robert Hirschfeld, Ryszard Kowalcyk, Andreas Polze, Matthias Weske (Hrsg.): NODe 2005, GSEM 2005

P-70 Klaus Turowski, Johannes-Maria Zaha (Hrsg.): Component-oriented Enterprise Application (COAE 2005)

P-71 Andrew Torda, Stefan Kurz, Matthias Rarey (Hrsg.): German Conference on Bioinformatics 2005

P-72 Klaus P. Jantke, Klaus-Peter Fähnrich, Wolfgang S. Wittig (Hrsg.): Marktplatz Internet: Von e-Learning bis e-Payment

P-73 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): “Heute schon das Morgen sehen“

P-74 Christopher Wolf, Stefan Lucks, Po-Wah Yau (Hrsg.): WEWoRC 2005 – Western European Workshop on Research in Cryptology

P-75 Jörg Desel, Ulrich Frank (Hrsg.): Enterprise Modelling and Information Systems Architecture

P-76 Thomas Kirste, Birgitta König-Riess, Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Informationssysteme – Potentiale, Hindernisse, Einsatz

P-77 Jana Dittmann (Hrsg.): SICHERHEIT 2006

P-78 K.-O. Wenkel, P. Wagner, M. Morgenstern, K. Luzi, P. Eisermann (Hrsg.): Land- und Ernährungswirtschaft im Wandel

P-79 Bettina Biel, Matthias Book, Volker Gruhn (Hrsg.): Softwareengineering 2006

P-80 Mareike Schoop, Christian Huemer, Michael Rebstock, Martin Bichler (Hrsg.): Service-Oriented Electronic Commerce

P-81 Wolfgang Karl, Jürgen Becker, Karl-Erwin Großpietsch, Christian Hochberger, Erik Maehle (Hrsg.): ARCS´06

P-82 Heinrich C. Mayr, Ruth Breu (Hrsg.): Modellierung 2006

P-83 Daniel Huson, Oliver Kohlbacher, Andrei Lupas, Kay Nieselt and Andreas Zell (eds.): German Conference on Bioinformatics

P-84 Dimitris Karagiannis, Heinrich C. Mayr, (Hrsg.): Information Systems Technology and its Applications

P-85 Witold Abramowicz, Heinrich C. Mayr, (Hrsg.): Business Information Systems

P-86 Robert Krimmer (Ed.): Electronic Voting 2006

P-87 Max Mühlhäuser, Guido Rößling, Ralf Steinmetz (Hrsg.): DELFI 2006: 4. e-Learning Fachtagung Informatik

P-88 Robert Hirschfeld, Andreas Polze, Ryszard Kowalczyk (Hrsg.): NODe 2006, GSEM 2006

P-90 Joachim Schelp, Robert Winter, Ulrich Frank, Bodo Rieger, Klaus Turowski (Hrsg.): Integration, Informationslogistik und Architektur

P-91 Henrik Stormer, Andreas Meier, Michael Schumacher (Eds.): European Conference on eHealth 2006

P-92 Fernand Feltz, Benoît Otjacques, Andreas Oberweis, Nicolas Poussing (Eds.): AIM 2006

P-93 Christian Hochberger, Rüdiger Liskowsky (Eds.): INFORMATIK 2006 – Informatik für Menschen, Band 1

P-94 Christian Hochberger, Rüdiger Liskowsky (Eds.): INFORMATIK 2006 – Informatik für Menschen, Band 2

P-95 Matthias Weske, Markus Nüttgens (Eds.): EMISA 2005: Methoden, Konzepte und Technologien für die Entwicklung von dienstbasierten Informationssystemen

P-96 Saartje Brockmans, Jürgen Jung, York Sure (Eds.): Meta-Modelling and Ontologies

P-97 Oliver Göbel, Dirk Schadt, Sandra Frings, Hardo Hase, Detlef Günther, Jens Nedon (Eds.): IT-Incident Mangament & IT-Forensics – IMF 2006

P-98 Hans Brandt-Pook, Werner Simonsmeier und Thorsten Spitta (Hrsg.): Beratung in der Softwareentwicklung – Modelle, Methoden, Best Practices

P-99 Andreas Schwill, Carsten Schulte, Marco Thomas (Hrsg.): Didaktik der Informatik

P-100 Peter Forbrig, Günter Siegel, Markus Schneider (Hrsg.): HDI 2006: Hochschuldidaktik der Informatik

P-101 Stefan Böttinger, Ludwig Theuvsen, Susanne Rank, Marlies Morgenstern (Hrsg.): Agrarinformatik im Spannungsfeld zwischen Regionalisierung und globalen Wertschöpfungsketten

P-102 Otto Spaniol (Eds.): Mobile Services and Personalized Environments

P-103 Alfons Kemper, Harald Schöning, Thomas Rose, Matthias Jarke, Thomas Seidl, Christoph Quix, Christoph Brochhaus (Hrsg.): Datenbanksysteme in Business, Technologie und Web (BTW 2007)

P-104 Birgitta König-Ries, Franz Lehner, Rainer Malaka, Can Türker (Hrsg.) MMS 2007: Mobilität und mobile Informationssysteme

P-105 Wolf-Gideon Bleek, Jörg Raasch, Heinz Züllighoven (Hrsg.) Software Engineering 2007

P-106 Wolf-Gideon Bleek, Henning Schwentner, Heinz Züllighoven (Hrsg.) Software Engineering 2007 – Beiträge zu den Workshops

P-107 Heinrich C. Mayr, Dimitris Karagiannis (eds.) Information Systems Technology and its Applications

P-108 Arslan Brömme, Christoph Busch, Detlef Hühnlein (eds.) BIOSIG 2007: Biometrics and Electronic Signatures

P-109 Rainer Koschke, Otthein Herzog, Karl-Heinz Rödiger, Marc Ronthaler (Hrsg.) INFORMATIK 2007 Informatik trifft Logistik Band 1

P-110 Rainer Koschke, Otthein Herzog, Karl-Heinz Rödiger, Marc Ronthaler (Hrsg.) INFORMATIK 2007 Informatik trifft Logistik Band 2

P-111 Christian Eibl, Johannes Magenheim, Sigrid Schubert, Martin Wessner (Hrsg.) DeLFI 2007: 5. e-Learning Fachtagung Informatik

P-112 Sigrid Schubert (Hrsg.) Didaktik der Informatik in Theorie und Praxis

P-113 Sören Auer, Christian Bizer, Claudia Müller, Anna V. Zhdanova (Eds.) The Social Semantic Web 2007 Proceedings of the 1st Conference on Social Semantic Web (CSSW)

P-114 Sandra Frings, Oliver Göbel, Detlef Günther, Hardo G. Hase, Jens Nedon, Dirk Schadt, Arslan Brömme (Eds.) IMF2007 IT-incident management & IT-forensics Proceedings of the 3rd International Conference on IT-Incident Management & IT-Forensics

P-115 Claudia Falter, Alexander Schliep, Joachim Selbig, Martin Vingron and Dirk Walther (Eds.) German conference on bioinformatics GCB 2007

P-116 Witold Abramowicz, Leszek Maciszek (Eds.) Business Process and Services Computing 1st International Working Conference on Business Process and Services Computing BPSC 2007

P-117 Ryszard Kowalczyk (Ed.) Grid service engineering and manegement The 4th International Conference on Grid Service Engineering and Management GSEM 2007

P-118 Andreas Hein, Wilfried Thoben, Hans-Jürgen Appelrath, Peter Jensch (Eds.) European Conference on ehealth 2007

P-119 Manfred Reichert, Stefan Strecker, Klaus Turowski (Eds.) Enterprise Modelling and Information Systems Architectures Concepts and Applications

P-120 Adam Pawlak, Kurt Sandkuhl, Wojciech Cholewa, Leandro Soares Indrusiak (Eds.) Coordination of Collaborative Engineering - State of the Art and Future Challenges

P-121 Korbinian Herrmann, Bernd Bruegge (Hrsg.) Software Engineering 2008 Fachtagung des GI-Fachbereichs Softwaretechnik

P-122 Walid Maalej, Bernd Bruegge (Hrsg.) Software Engineering 2008 - Workshopband Fachtagung des GI-Fachbereichs Softwaretechnik

P-123 Michael H. Breitner, Martin Breunig, Elgar Fleisch, Ley Pousttchi, Klaus Turowski (Hrsg.)
Mobile und Ubiquitäre Informationssysteme – Technologien, Prozesse, Marktfähigkeit
Proceedings zur 3. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2008)

P-124 Wolfgang E. Nagel, Rolf Hoffmann, Andreas Koch (Eds.)
9th Workshop on Parallel Systems and Algorithms (PASA)
Workshop of the GI/ITG Speciel Interest Groups PARS and PARVA

P-125 Rolf A.E. Müller, Hans-H. Sundermeier, Ludwig Theuvsen, Stephanie Schütze, Marlies Morgenstern (Hrsg.)
Unternehmens-IT: Führungsinstrument oder Verwaltungsbürde
Referate der 28. GIL Jahrestagung

P-126 Rainer Gimnich, Uwe Kaiser, Jochen Quante, Andreas Winter (Hrsg.)
10th Workshop Software Reengineering (WSR 2008)

P-127 Thomas Kühne, Wolfgang Reisig, Friedrich Steimann (Hrsg.)
Modellierung 2008

P-128 Ammar Alkassar, Jörg Siekmann (Hrsg.)
Sicherheit 2008
Sicherheit, Schutz und Zuverlässigkeit
Beiträge der 4. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI)
2.-4. April 2008
Saarbrücken, Germany

P-129 Wolfgang Hesse, Andreas Oberweis (Eds.)
Sigsand-Europe 2008
Proceedings of the Third AIS SIGSAND European Symposium on Analysis, Design, Use and Societal Impact of Information Systems

P-130 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
1. DFN-Forum Kommunikations-technologien Beiträge der Fachtagung

P-131 Robert Krimmer, Rüdiger Grimm (Eds.)
3rd International Conference on Electronic Voting 2008
Co-organized by Council of Europe, Gesellschaft für Informatik and E-Voting.CC

P-132 Silke Seehusen, Ulrike Lucke, Stefan Fischer (Hrsg.)
DeLFI 2008:
Die 6. e-Learning Fachtagung Informatik

P-133 Heinz-Gerd Hegering, Axel Lehmann, Hans Jürgen Ohlbach, Christian Scheideler (Hrsg.)
INFORMATIK 2008
Beherrschbare Systeme – dank Informatik
Band 1

P-134 Heinz-Gerd Hegering, Axel Lehmann, Hans Jürgen Ohlbach, Christian Scheideler (Hrsg.)
INFORMATIK 2008
Beherrschbare Systeme – dank Informatik
Band 2

P-135 Torsten Brinda, Michael Fothe, Peter Hubwieser, Kirsten Schlüter (Hrsg.)
Didaktik der Informatik – Aktuelle Forschungsergebnisse

P-136 Andreas Beyer, Michael Schroeder (Eds.)
German Conference on Bioinformatics GCB 2008

P-137 Arslan Brömme, Christoph Busch, Detlef Hühnlein (Eds.)
BIOSIG 2008: Biometrics and Electronic Signatures

P-138 Barbara Dinter, Robert Winter, Peter Chamoni, Norbert Gronau, Klaus Turowski (Hrsg.)
Synergien durch Integration und Informationslogistik
Proceedings zur DW2008

P-140 Oliver Göbel, Sandra Frings, Detlef Günther, Jens Nedon, Dirk Schadt (Eds.)
IMF 2008 - IT Incident Management & IT Forensics

The titles can be purchased at:

Köllen Druck + Verlag GmbH
Ernst-Robert-Curtius-Str. 14 · D-53117 Bonn
Fax: +49 (0)228/9898222
E-Mail: druckverlag@koellen.de