High-Efficient Intrusion Detection Infrastructure

Thomas Holz, Michael Meier, Hartmut König

Brandenburg University of Technology Cottbus
Department of Computer Science
PF 10 13 44
03013 Cottbus
Germany
{thh,mm,koenig}@informatik.tu-cottbus.de

Abstract: In recent years research activities in computer network security focus more actively on the development of effective methods in intrusion detection. The reason for this development is the rapidly increasing potential of threats to social, economical, and military information stored in information technology (IT) systems. Powerful and practically applicable mechanisms are required to protect critical infrastructures. Intrusion detection systems have been proven as a powerful means for the detection of IT security violations. They provide protection of computer and network resources by automatic detection of security violations. Some of these systems are able to initiate appropriate intrusion response actions. The crucial point for real-time applications, especially for host-based audit analysis, is the detection speed. In the paper we present the distributed intrusion detection infrastructure HEIDI which tackles this problem. HEIDI provides a module system based on sensors and agents to set up tailored intrusion detection systems for real-time applications. The basic features of the HEIDI approach are a distributed analysis functionality, the handling of overload situations, and a dynamic configurability. Furthermore, the problem of time-consuming audit analysis is compensated by integration of StraFER, a new signature match algorithm.

1 Motivation

The rapid advance of communication technologies in many areas of the human society accelerates the shift of many social processes on such systems, in particular on the Internet. This brings numerous benefits to the users, but it also increases their dependencies on these technologies. These dependencies as well as the technological complexity of the systems themselves create an increasing potential of threats for these systems which make them more and more vulnerable. The constantly growing number of computers in the Internet gives hackers, crackers, terrorists, and subversive insiders better and better opportunities for attacks. This concerns not only the critical infrastructures of industrial states, the trade, the information system, or the health care system, but in particular the military protection of the society [1, 2].

To counter these threats besides preventive measures such as authentication of communication partners, encryption of communication, and access control to resources, means on the technological level are required which allow to detect and indicate security violations to evaluate and to possibly confine the damage.

Intrusion detection systems (IDS) have proved to be an effective instrument for this purpose. Systems with real-time capabilities provide automated protection of computer and network resources and allow the detection of ongoing security violations. Intrusion detection systems are currently one of the few reactive security mechanisms to counter threats on the communication infrastructure. They have been developed since 20 years and successfully deployed in practice [3]. Various commercial solutions are available. The effectiveness of commercial systems in real-life environments, however, is limited. They mainly confine themselves to detecting simply structured security violations in a post-mortem mode. For the deployment in large computer networks, they are less suited, especially for tight time constraints. Growing communication infrastructures and increasing user requirements raise new problems (e.g. encryption, switching technologies) and demands (e.g. privacy) which are not covered by existing concepts.

In the paper we present the distributed intrusion detection infrastructure HEIDI that is currently being developed at Department of Computer Science at Brandenburg University of Technology Cottbus. It is based on the experience we gathered with the intrusion detection system AID [4] developed in our group and other systems as well. The objective of the HEIDI approach is to get a module system to flexibly set up intrusion detection systems for real-time applications. The HEIDI concept is based on the use of sensors and agents which can be combined to set up a tailored intrusion detection system which meets the requirements of a given application environment. Unlike other approaches, which try to reduce the amount of audit data, the HEIDI concept aims at the complete evaluation of all audit data. This is achieved by using a combined analysis distribution, the delegation of processing functionality in overload situations and the application of a new optimized signature match algorithm. The paper describes the basic principles of the HEIDI approach. Section 2 formulates requirements to the design of modern intrusion detection systems, which inspired the HEIDI development. Section 3 introduces the basic features of the HEIDI approach. It gives examples for the surveillance of different IT systems with HEIDI-based architectures. In Section 4 we describe the new match algorithm StraFER which is applied in HEIDI agents. The conclusion summarizes the achievements of the approach and gives an outlook on next research steps.

2   Requirements to modern Intrusion Detection Systems

The security function intrusion detection deals with the monitoring of IT systems to detect security violations. Due to the large amount of incoming audit data this analysis can be only efficiently processed if automated evaluation support is given. The decision which activities can be considered as security violations in a given context is determined by the applied security policy. For the detection of security violations, mainly two complementary approaches are applied: anomaly detection and misuse detection. Anomaly detection aims at the exposure of abnormal user behavior. Misuse detection focuses on the automatic detection of known attacks. These attacks are described by patterns, so called signatures, required to identify an attack in an audit data stream. Misuse detection has revealed to be the more effective approach. It is only considered in the sequel.

Intrusion detection systems are usually applied to monitor critical servers and complete (local) computer networks. The observation of desktop computers represents more an ex-

ception. For the observation, stand-alone and distributed intrusion detection systems can be distinguished. Intrusion detection systems are often deployed in connection with other security mechanisms like firewalls to support a superior security management. Therefore distributed solutions are more and more becoming a typical characteristic of modern intrusion detection systems. The deployment of intrusion detection systems in practice has revealed a couple of problems such as the high amount of audit data, privacy, response mechanisms, insufficient system protection, high false alarm rates and others for which different solutions have been proposed. In the discussion here we focus on the following problems: detection efficiency, and system adaptation and maintenance. We consider in particular

1. the use of distributively acting IDS components,
2. the efficient development of signatures, and
3. the integration of efficient analysis methods.

In this context, our main interest aims at the minimization of the time intervals between the appearance of security violations and their detection to improve the chances of effective intrusion response actions. Unlike network based intrusion detection systems, host based systems often provide no chance to stop an ongoing attack, because audit records are mostly only reflections of completed security relevant actions. In case of relatively long attacks, however, a fast audit analysis may be able to reduce the arising damage.

2.1 Distributively acting IDS components

Modern distributed intrusion detection systems consist of a set of modules for capturing, preprocessing, analyzing the audit data, and for archiving them if required. In addition, most commercial products offer an appropriate management functionality. As far as efficiency is concerned, the primary aspect of such a system is the distribution of time-consuming analysis functionality. Two categories of intrusion detection systems can be distinguished: systems with a centralized and a decentralized analysis. Systems with a centralized analysis are simpler to implement, but the analysis function represents a performance bottleneck and a single point of failure. Further large amounts of data have to be transferred between a local host and the central processing unit. These effects we could also observe while testing our own centralized system AID (*Adaptive Intrusion Detection system*) [4]. If several processing units are used synchronization is required, but load distribution and efficiency are essentially better. Furthermore the processing units can be aligned in a hierarchical manner, e.g. in a two-layered scheme, where the units at the lower level perform a detection of local attacks and a unit at the upper level is responsible for finding all distributed security violations.

Most approaches, however, apply the centralized processing paradigm [5]. Only a few systems use the distributed approach as, for instance, DIDS, CSM, AAFID, and EMERALD. In this context DIDS (*Distributed Intrusion Detection System*) was the first intrusion detection system that combined local audit evaluations, network monitoring and a central alarm correlation [6]. CSM (*Cooperating Security Managers*) implemented an unusual idea [7]. Here every monitored host contains a security manager. When a user logs in first time on a certain host the security manager of this host is responsible for recording and

analyzing all subsequent actions of this user, even if he moves to another host within the network. The roaming of users, however, cause an enormous data transfer. The approach is therefore not applicable to a fast inspection of large amounts of audit data. The AAFID (*Autonomous Agents for Intrusion Detection*) concept, on the other hand, addresses the problem of system load caused by using intrusion detection systems. The main idea here consists in the application of many small, specialized and hierarchically grouped entities [8]. These components (filters, agents, transceivers and monitors) are too limited in their performance to meet the requirements of an efficient audit analysis. The fourth systems, EMERALD (*Event Monitoring Enabling Responses to Anomalous Live Disturbances*), is a military sponsored development which aims at the application of a flexible set of complex modules, the EMERALD monitors. These monitors integrate both detection and response functionality, and are designed to be interoperable with many other security functions at a very high degree [9]. They can be connected among each other within a three-layered hierarchy. An EMERALD intrusion detection system is able to protect large networks, especially in critical enterprise environments. This approach, as well as others, does not aim at a high audit analysis speed. There are no documented performance data available for these systems so far.

Further problems that are related to the design of distributed intrusion detection systems are a high runtime adaptability, robustness, availability and fault tolerance. In particular in sensitive environments there is a fundamental operational need for the survivability of the intrusion detection systems under various conditions. Every time interval, in which the system is not running, represents a threatening situation. Modern distributed intrusion detection architectures used for the protection of critical infrastructures should ensure this dynamic adaptability as extensively as possible [10].

2.2   Efficient development of signatures

The goal of misuse detection systems is to automatically find traces of known attacks in audit data streams. The signatures have to be expressed in a representation that can be used as basis for the analysis process. The derivation of signatures based on a given security policy has often proved to be a crucial point in audit-based misuse detection. The problem can be divided in two steps:

1. to determine attack patterns to be searched for during the analysis, and
2. to encode the patterns in an appropriate form.

The determination of patterns that are significant for an attack usually involves an in-depth investigation of the constituent actions of known attacks. This step requires a lot of expertise about attacks and audit functions. The identified signatures have to be expressed in an adequate representation that can be used as basis for the analysis process. Since the representation of signatures is specific to the used detection mechanisms most intrusion detection systems use their own language to encode signatures. Many existing intrusion detection systems (e.g. EMERALD, ASAX [11], AID) use rule-based languages to encode signatures. Others use state transition diagrams (STAT [12]) or petri net representations (IDIOT [13]).

Although these approaches have shown their usefulness, they lack some important features. Most intrusion detection systems require not only a description of the signatures but also details on the manner the detection process has to work. Often adequate means to build abstraction in developing signatures are missed. This makes the development and maintenance of signatures complicated and error-prone.

2.3 Integration of efficient analysis methods

The analysis of the audit data is still the bottleneck for intrusion detection systems, especially for real-time applications. Although analysis techniques have improved continuously, the capability of knowledge-based methods, which are usually applied in intrusion detection systems, remains limited. The increasing amount of audit data often wipes out the progress. Audit bursts like, for instance, 100 Mbps for usual PCs running Microsoft Windows 2000 requires special optimized algorithms.

Concerning the analysis techniques misuse detection systems can be roughly divided into two groups. The first group of systems uses the specified signatures to generate program code (e.g. in C/C++) that searches for attack patterns using the specified criteria. Systems of the second group use the expert system approach to analyze audit data.

Examples for intrusion detection systems of the first group are IDIOT and the STAT tool suite. These systems create a program module for each signature that after its compilation can be used to search for the specified patterns in audit data. A performance issue here is that different signatures are checked or executed independently. Thus it is scarcely possible to take advantage of the optimization potential that can be used if signatures are analyzed together. For example, techniques like condition sharing to avoid redundant calculations of conditions in a signature set cannot be used.

AID, ASAX, and EMERALD's P-BEST component [14] are examples for systems of the second group. Here the inference mechanism and match algorithm of the underlying expert system shell determine the used analysis technique. The most popular and applied match algorithm of these systems is RETE [15] (others are TREAT [16] and LEAPS [17]). The issue is that match algorithms of expert system shells are general-purpose algorithms. It is well known that a match algorithm may outperform all other algorithms in a specific application domain [18]. Therefore it depends on the concrete application domain which algorithm performs best. To our knowledge it is an unanswered question which match algorithm is best suited for misuse detection expert systems. Further the question arises, whether a special-purpose match algorithm for misuse detection systems can be constructed that noticeably outperforms general-purpose algorithms by taking advantage of known information about the application domain.

3 The HEIDI approach

The objective of the HEIDI approach (*High-Efficient Intrusion Detection Infrastructure*) is to provide an infrastructure for setting up tailored intrusion detection systems to speed up the detection capability. The term "infrastructure" means that a module system is defined which can be adapted to a specific intrusion detection architecture for a given target

environment and application scenario, respectively. The main characteristics of such an architecture are the placement of necessary HEIDI modules and the general specification of their interconnectivity [19]. Further refinements of the architecture towards a real intrusion detection system can be introduced by the integration of target-specific adaptations, e.g. interfaces for capturing host-specific audit data.

3.1   Functional overview

HEIDI distinguishes 3 basic components: sensors, agents, and user interfaces. The sensors collect and preprocess audit data. The agents provide the analysis units. They can cooperate among each other. The user interfaces serve for system management and user interactions.

3.1.1   HEIDI sensors

HEIDI sensors are specialized modules to collect and to handle audit data. They aim at a fast reading, preprocessing and forwarding of the audit data. Sensors can be placed at different points of the monitored hosts depending of the applied security policy. Different sensors at one host are coordinated by the supervising local agent. Figure 1 depicts the generic structure of a sensor.

HEIDI sensors consist of permanent components (illustrated in darker grey), e.g. the read interface, the transformation unit for data converting, and the transfer buffer. Other components are optional (illustrated in lighter grey), e.g. the pseudonymization unit for encrypting user identifying references in the audit data like the user ID, the group ID and others to ensure user privacy. Most components of the sensor are connected by a data processing pipeline. They are supervised by the control and configuration component. After reading the information from the local host, the data are preprocessed and forwarded to the local HEIDI agent.

3.1.2   HEIDI agents

After the fast capturing of the audit data by the sensors the second step to maximize the detection speed is to ensure an efficient analysis of these data. This is the task of the HEIDI agents. Beside the application of optimized analysis algorithms (see Section 4) they use an appropriate distribution of data. This distribution is based on a classification of the signatures into local and distributed contexts. To detect signatures with a local context only locally preprocessed data are analyzed, while for signatures with a distributed context data from various agents are demanded.

The most efficient way to perform such an analysis in a network is to apply a combined execution scheme. Similarly to systems like DIDS, AAFID, and EMERALD, HEIDI prefers to match signatures with local context on the corresponding host. The detection of distributed attacks takes place on a central location. Unlike any other known system, however, HEIDI applies this hybrid concept in a stringent manner to achieve a maximal
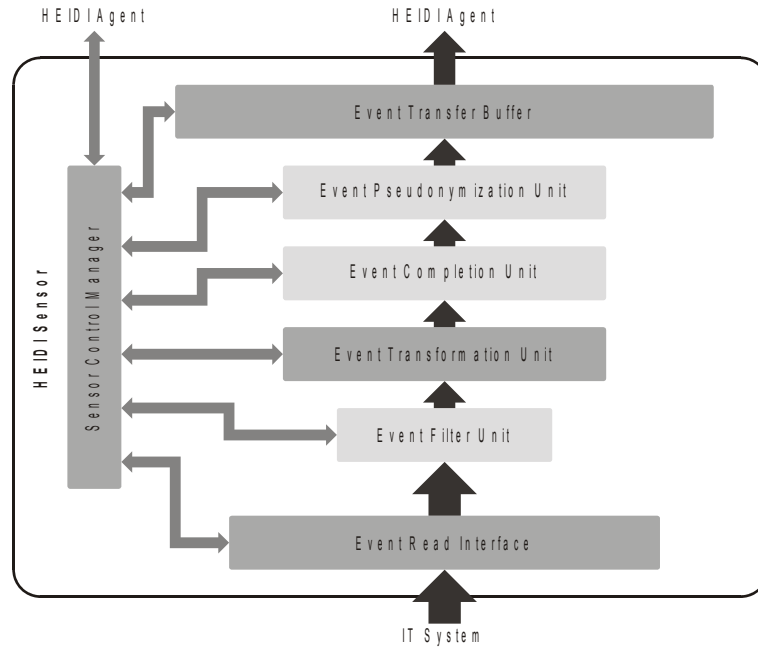
Figure 1: Structure of a HEIDI sensor

local concentration and a minimal need for network traffic and delay. For this purpose we extended the notion of signature. In context of HEIDI signatures are not only used for mapping complete security violation sequences. A signature can also represent a partial sequence of such an attack. This extension enables a hierarchy of agents to split the detection process for a distributed attack into a number of local sub-detections and a small amount of central combining. This principle is not applicable to all distributed attacks, but some critical security violations, like several doorknob rattling variants, can be easily detected this way.

For the execution of all local and central detection processes, each involved host contains a single stationary HEIDI agent. Figure 2 shows the structure of an agent.

A HEIDI agent receives the preprocessed information from all local sensors. It contains a central data processing pipeline which consists of an input, a processing, and an output unit. Input and output units are responsible for receiving, synchronizing and transmitting the security relevant data. The analysis of the audit data takes place in the processing unit. Several analysis processes can run in parallel, e.g. a complete signature detection, a fast signature detection for particularly critical attacks, and a simple audit statistics. The analysis algorithm applied currently is StraFER which is introduced in Section 4. In addition to the detection capabilities an agent can contain an active response unit which is able to initiate appropriate local countermeasures.

Figure 2: Structure of a HEIDI agent

3.1.3 HEIDI user interfaces

A HEIDI user interface is a graphical application which enables a security operator to perform several tasks in the context of a given HEIDI intrusion detection system. The most important tasks are the configuration of the system and the visualization of the detection results. Furthermore, a user interface can act as a link between a security management and a HEIDI intrusion detection system.

Every HEIDI agent provides a single interface for the connection with a user interface. This connection can be either local or remote. Thus a user interface can dynamically connect to a number of agents. Since several user interfaces can be attached to a HEIDI system at the same time every interface has to periodically read the corresponding configuration data.

3.1.4 Handling of overload situations

For assuring a continuous, robust, and efficient intrusion detection operation, HEIDI uses an adaptive mechanism to compensate temporary overload situations. In conventional systems such overload situations, e.g. an audit burst, normally stop the execution of the intrusion detection system or cause a crash. To avoid this HEIDI agents are able to delegate analyzing functionality to other agents. A destination agent receives the preprocessed

data and the analysis state. The delegation functionality, the required number of desti-nation agents, and the duration of the delegation depend on several conditions. They are calculated and negotiated dynamically. Normally, a re-delegation is carried out when the overload situation has disappeared. To estimate the load situation in the intrusion detection system a HEIDI agent contains a monitor which evaluates the performance of both the host and some time-consuming agent components (see Figure 2).

### 3.2   Setting up intrusion detection architectures

HEIDI sensors and agents can be combined to set up a hierarchical intrusion detection architecture for a given target environment, e.g. a host or a local area network. Depend-ing on the network structure and the applied security policy, special sensors and internal communication schemes can be configured. In this context, connectivity and data stream configuration have a special importance. For every security violation to be detected, it must be determined which subset of modules is involved and where the data analysis is appropriately located. To offer a flexible and efficient setup, an agent also can act as a transceiver, i.e. it does not analyze, or as a delegation server. The latter, which describes a HEIDI agent on demand, is required for enterprise networks with high failure safety requirements.

Figure 3 shows two different intrusion detection architectures. The left example shows a two-layered hierarchy, the right example a three-layered. All illustrated hosts (the greater rectangles) are equipped with two sensors (smaller embedded rectangles) and the corre-sponding agent (greater embedded rectangle). Streams of preprocessed audit and result information are illustrated as arrows, whereas the thickness of the arrows serves as an indicator for the transfer rates between the modules. In the left example all agents are working on detection processes, and the agent at the upper level is responsible for finding attacks with distributed context. In the right example, the white illustrated agents do not perform any analysis, so that their superior agent has to deal with a relatively high amount of incoming data. In this example, an agent at the third level serves as an overall result collector.
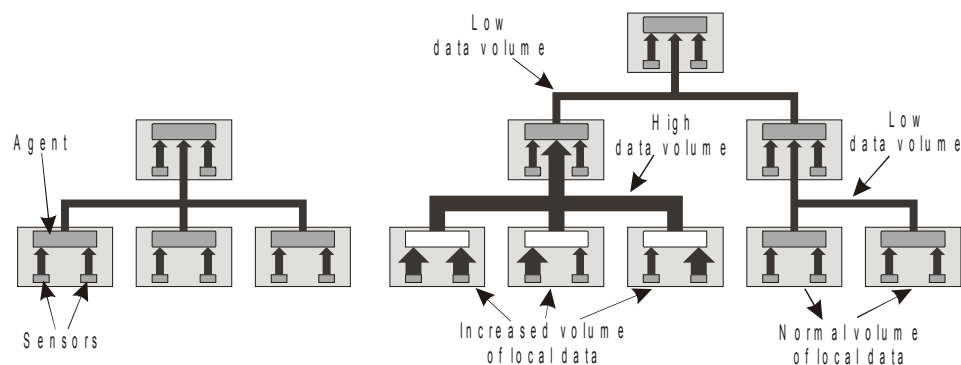
Figure 3: Examples of two HEIDI-based intrusion detection architectures

Figure 4 shows the expansion of the left architecture from Figure 3 by the integration of two delegation servers. The hosts, on which these delegation servers run, do not have sensors. In Figure 4 the upper depicted delegation server is configured to exclusively help the upper-level agent in the regular detection hierarchy. The lower server is dedicated to handle overload situations for all low-level hosts in the regular detection hierarchy. The left example shows a burst situation at two low-level hosts. The lower server overtakes in this case the analysis of the data. In the right example, there are also two hosts in an overload state. One of them is the upper-level host in the regular detection hierarchy. In this case the processing capacities of both delegation servers are used.
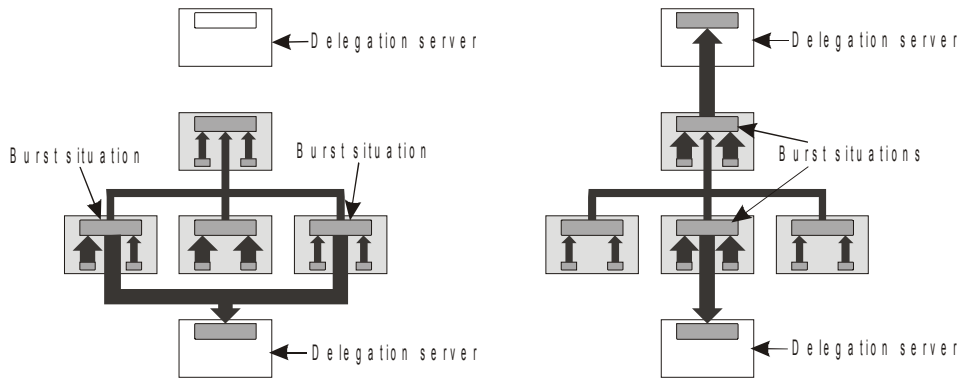
Figure 4: Scenario with two different overload situations for a single architecture

4   Efficient detection components

Beside the distributed collection and processing of audit data HEIDI provides means for an efficient analysis. Unlike many other systems HEIDI agents do not use a rule-based analysis. Instead the IDS specific analysis algorithm StraFER has been developed. This algorithm takes advantage of knowledge about the static structure of attack signatures to reduce the processing time. StraFER is based on the event description language SHEDEL which has been defined to facilitate the description and introduction of signatures into agents. In the sequel we first shortly introduce SHEDEL. Thereafter we give an overview of the StraFER algorithm.

4.1   The attack description language SHEDEL

To simplify the development and maintenance of signatures we developed the attack description language SHEDEL (*Simple Hierarchical Event Description Language*). The main objective of SHEDEL is to provide means to describe signatures without determining the detection process.

The main abstraction in SHEDEL is the event. An event can represent an audit record, an attack signature, a Meta attack signature and so on. Describing a signature in SHEDEL

means to specify an event. An event consists of a collection of one ore more sub-events, also called steps, which are related to each other, temporally or by their properties. There is a set of basic events which cannot be divided in sub-events. They represent the basic recognizable units, e.g. audit records or network packets. Thus it is possible to specify a hierarchy of events that can be used to describe a pattern of audit records. To support the response to an attack (or any event) it is possible to link an action to sub-events of an event specification.

An event is characterized by a name and a set of properties called features. The name is used as reference for other events. The features contain relevant information about an event. A basic event contains all information of the corresponding audit record, e.g. the names of involved users and objects are represented as features of the event. For non-basic events, the features must be defined. A feature can be defined by referring to a feature of a step. Thus a feature definition is a pair consisting of a name and a feature of a step contained by this event.

For the successful completion of an attack, its constituent actions usually must be executed in a specific order. To describe the signature of such an attack in SHEDEL it must be possible to specify the order of steps of an event. This can be done by defining a predecessor of a step. Additionally to the correct order of actions typically further conditions must be fulfilled to complete an attack successfully. An example for such a condition is that the same user must execute all actions of an attack. To describe signatures for such kind of attacks SHEDEL allows the definition of conditions within an event.

Figure 5 shows the description of an event E that consists of the steps A and B. The features start and filename of E are defined using the features time and file of step A. The username feature of E is mapped to the user feature of step B. The right side of Figure 5 shows the resulting event hierarchy.
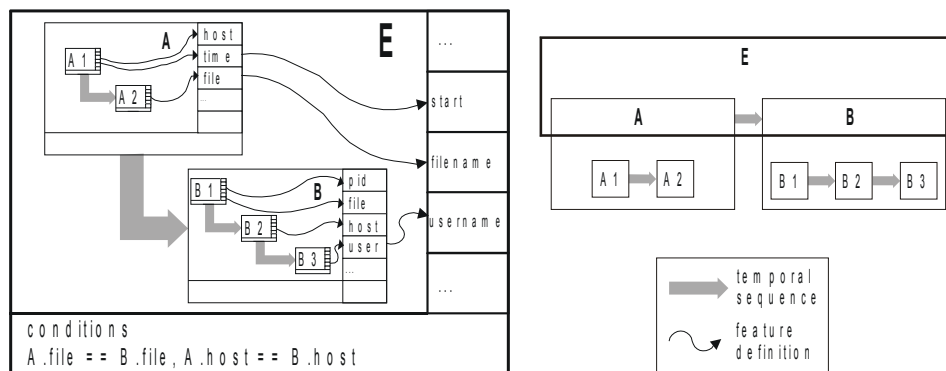
Figure 5: Definition of an event and the resulting event hierarchy

The description of the illustrated event E in SHEDEL looks as follows:

```
EVENT E
{
  STEP step1
    INITIAL           // step1 is the initial step
    TYPE A
  STEP step2
    EXIT              // step2 is the final (completing) step
    TYPE B
    REQUIRES step1    // step1 is predecessor of step2
  ACTIONS
    ...               // any actions
  CONDTIONS
    step1.file == step2.file,
    step1.host == step2.host
  FEATURES
    start = step1.time,
    filename = step1.file,
    username = step2.user
}
```

Note that no operational details about the detection process are contained in this description. Using hierarchical event description it is possible to define adequate abstractions to reason about intrusions. So it is simple to define a generalization of different signatures or to specialize a signature. A detailed discussion of SHEDEL and example signatures are contained in [20].

4.2   An algorithm for event-based misuse detection

Proceeding from signatures described in SHEDEL, an algorithm is needed, which matches the specified patterns against a stream of audit data. Because of the reasons discussed in Section 2.3 neither a simple code generation approach nor a (general-purpose) expert system shell is used for this purpose. Instead we develop a special-purpose algorithm, named StraFER (*Straight-Forward Event Recognition*), that is based on descriptions in SHEDEL. In this section we first present a straight-forward algorithm for matching event-based signatures. Second we discuss some ideas for optimizations of such an algorithm.

4.2.1   A straight-forward approach for event-based misuse detection

A stream of audit records and a set of SHEDEL event descriptions form the input for the algorithm. Audit records are treated as observable input events. They are represented by SHEDEL basic events.

The analysis unit first reads all specified event descriptions and creates for each description a corresponding event object (an initial instance of the event description). During processing the straight-forward analysis algorithm does the following for each input event:

For each event object E

1. Check
   - if there are steps S of the type of the input event,
   - if all required predecessor steps P of these steps S have already occurred,
   - if all conditions, which refer to features of steps S respectively features of the input event and can already be evaluated, are fulfilled.

2. If these checks were successful
   - a copy of the event object E is created,
   - required information on the step S are saved in the copied event object,
   - the actions linked to step S are executed.
3. If step S is the last (and completing) step of event object E
   - the features of this event are assigned,
   - the occurrence of the event represented by event object E is signaled,
   - the event object is removed.

Occurred events are used as input event in the next cycle. After all occurred events are processed as input events the next audit record is used as input event.

A basic version of the StraFER algorithm is implemented. It detects all attack patterns specified as event descriptions; but it consumes a relatively high amount of memory and processing time.

4.2.2   4.2.2 Ideas for optimizing event-based analysis

The StraFER algorithm as discussed above checks all conditions of all event objects. Each of these checks consumes time. The runtime can be decreased

- by not checking each event object in every cycle,
- by not checking all conditions of each event object, and
- by optimizing the condition evaluation, for instance, by avoiding multiple calculations of the same function with the same parameters.

StraFER tries to realize the above points by taking advantage of knowledge about the structure and contents of its input data, especially the event based signature descriptions. Further some state information is maintained during the analysis to reduce the process runtime.

StraFER currently uses the following ideas:

1. Using the type of an input event the set of event objects, which have to be checked, is reduced to those event objects that contain a step of this type.
2. Steps of an event object usually are in temporal relation to each other. Thus the set of event objects that have to be checked can be reduced to those event objects with no or already occurred predecessor steps.
3. Events that represent only a partial signature (and thus have not linked an action to its final step) can be treated in a special way. Event B in Figure 5 shall be such an event. The occurrence of the final step B3 of event B can imply the occurrence of event B. Event B is used as step in event E and has an event A as its predecessor step. As long as the event A does not have occurred the event B3 does not have any effect with respect to the detection of event E. In this cases events like B3 are called events with no effect. Such steps and the conditions associated with them do not have to be evaluated.

4. Conditions of an event object that describe relations between features of different steps are called inter-event-conditions while conditions that refer to features of the current input event are called intra-event-conditions. Inter-event-conditions have only to be evaluated if they
   - refer to features of steps with a type equal to the type of the current input event and
   - refer to features of already occurred steps.
5. During the matching runtime a couple of event objects of an event description are created that represent different partial matches of a signature. For example the execution of signature actions by different users result in multiple event objects representing partial signature matches of these users. Thus for an input event a step of a signature has to be matched for multiple event objects. Since intra-event-conditions of a step are identical for all event objects they need to be matched only once, and the evaluation result can be recycled for all event objects.

The ideas mentioned in point 1 and 2 reduce the number of event objects. Additionally the number of conditions is decreased using the concepts in point 3, 4 and 5.

To implement the above ideas different calculations are required. A main part of them can be done at compile time (during the reading of the event descriptions) using information about the static structure of signatures. During runtime only a few modifications of control data are needed. Thus a noticeably runtime improvement can be expected. We are currently introduce these optimizations into the StraFER implementation.

5 Conclusion

In the paper we have presented the basics of the intrusion detection infrastructure HEIDI. The HEIDI approach aims at a module system to set up efficient and tailored intrusion detection systems for local area networks. The module system provides a set of specialized sensors for audit data capturing and flexible agents for data analysis. The analysis combines local and central signature matching processes. Furthermore, a HEIDI-based intrusion detection system is capable to react to overload situations by delegating analysis functionality among the communicating agents.

So far only a very few intrusion detection approaches or systems have the potential to overcome the efficiency problem of the host-oriented intrusion detection. It has shown that only decentralized analysis approaches like DIDS, AAFID, and EMERALD are capable to meet near real-time requirements. Unfortunately, none of these systems aim at an efficient intrusion detection solution, but from an architectural point of view some aspects are comparable with HEIDI. The systems CSM and EMERALD are characterized by the application of large and complex modules. This feature is similar to the HEIDI agents. In contrast to this AAFID uses a great number of small an specialized entities. In HEIDI the sensors play a similar role. Since HEIDI uses both complex and small modules it is also comparable to DIDS. Depending on the different development targets each of this systems has special module-intern structures. Regarding the module interconnection capabilities HEIDI seems to be as potentially as EMERALD and AAFID, while DIDS and CSM are functionally limited in this context.

The implementation of the HEIDI infrastructure modules is still in progress. Currently the implementation of various sensors, e.g. for capturing audit data under Sun Solaris and Microsoft Windows NT/2000, and for TCP/IP segments are available. After finishing the implementation of the HEIDI agent we plan to set up a first example intrusion detection system which corresponds our system AID [4]. By comparing the two AID variants we will evaluate the performance of the HEIDI concept. Thereafter the usability of the HEIDI concept will be investigated with different IDS architectures.

To support efficient data analysis with HEIDI agents the new algorithm StraFER that utilize knowledge about the static structure of signatures has been proposed. A basic version of StraFER is available. The next step will be to evaluate the impact of the discussed optimizations on the processing time. We plan to directly compare the StraFER implementation with the rule based analysis component of AID. Based on this further optimizations will be introduced to the algorithm.

References

[1] Clinton Administration (ed.): The Clinton Administration's Policy on Critical Infrastructure Protection : Presidental Decision Directive 63, 1998.

[2] Denning, Dorothy E.: Information Warfare and Security. Addison Wesley Longman, Inc., Reading, 1999.

[3] Meier, Michael; Holz, Thomas: Intrusion Detection Systems List and Bibliography. http://www-rnks.informatik.tu-cottbus.de/en/security/ids.html, 2003.

[4] Sobirey, Michael; Richter, Birk; Koenig, Hartmut: The Intrusion Detection System AID - Architecture, and experiences in automated audit analysis. In: Horster, Patrick (ed.): Proceedings of the IFIP TC6/TC11 International Conference on Communications and Multimedia Security at Essen, Germany, 23rd - 24th September 1996. Chapman & Hall, London, 1996, pp. 278-290.

[5] Axelsson, Stefan: Research in Intrusion Detection Systems: A Survey. Goeteborg, Chalmers University of Technology, Technical Report No. 98-17, 1998.

[6] Snapp, Steven R.; Smaha, Stephen E.; Teal, Daniel M.; Grance, Tim: The DIDS (distributed intrusion detection system) prototype. In: USENIX Association (ed.): Proceedings of the Summer 1992 USENIX Conference. USENIX Association, Berkeley, 1992, pp. 227-233.

[7] White, Gregory B.; Pooch, Udo W.: Cooperating security managers: Distributed intrusion detection systems. In: Computers & Security 15 (1996), No. 5, pp. 441-450.

[8] Spafford, Eugene H.; Zamboni, Diego: Intrusion detection using autonomous agents. In: Computer Networks 34 (2000), No. 4, pp. 547-570.

[9] Porras, Phillip A.; Neumann, Peter G.: EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In: NIST (ed.); NCSC of the NSA (ed..): Proceedings of the 20th NISSC, 1997. National Institute of Standards and Technology, Gaithersburg, 1997, pp. 353-365.

[10] Holz, Th.; Meier, M.; Koenig, H.: Bausteine für effiziente Intrusion Detection Systeme. In PIK 25 (2002) 3, 144-157.

[11] Mounji, Abdelaziz: Languages and Tools for Rule-Based Distributed Intrusion Detection. University of Namur, Belgium, Computer Security Institute, PhD Thesis, 1997.

[12] Vigna, Giovanni; Eckmann, Steven T.; Kemmerer, Richard A.: The STAT Tool Suite. In: Proceedings of DISCEX, Hilton Head, South Carolina. IEEE Computer Society Press, Los Alamitos, California, 2000.

[13] Kumar, Sandeep: Classification and Detection of Computer Intrusions. Purdue University, PhD Thesis, 1995.

[14] Lindqvist, Ulf; Porras, Phillip A.: Detecting Computer and Network Misuse with the Production-Based Expert System Toolset (P-BEST). In: Proceedings of the IEEE Symposium on Security and Privacy. IEEE Computer Society Press, Los Alamitos, California, 1999, pp. 146-161.

[15] Forgy, Ch. L.: Rete: A Fast Algorithm for the Many Pattern / Many Object Pattern Match Problem. In: Artificial Intelligence, No. 10, 1982, pp. 17-37.

[16] Miranker, Daniel P.: TREAT: A better match algorithm for AI production systems. In: Proceedings of AAAI 87 Conference on Artificial Intelligence, 1987, pp. 42-47.

[17] Miranker, Daniel P.; Brant, D. A.: An Algorithmic Basis for Integrating Production Systems and Large Databases. In: Proceedings of the Sixth International Conference on Data Engineering, February 5-9, 1990, Los Angeles, California, USA. IEEE Computer Society, 1990, pp. 353-360.

[18] Wang, Y.; Hanson, E. N.: A Performance Comparison of the Rete and TREAT Algorithms for Testing Database Rule Conditions. In: Golshani, Forouzan (ed.): Proceedings of the Eighth International Conference on Data Engineering, 1992. IEEE Computer Society, 1992, pp. 88-97.

[19] Holz, Th.; Meier, M.; Koenig, H.: An Efficient Intrusion Detection System Design. In: Proceedings of the Information Security for South Africa Conference, Muldersdrift, South Africa, July 10-12, 2002.

[20] Meier, Michael; Bischof, Niels; Holz, Thomas: SHEDEL - A Simple Hierarchical Event Description Language for Specifying Attack Signatures. To appear in: Proceedings of the IFIP International Conference on Information Security, Cairo, Egypt, 2002, pp. 559-571.