Internet Voting System Security Auditing from System Development through Implementation: Best Practices from Electronic Voting Deployments

L. Jay Aceto[1], Michelle M. Shafer[2], Edwin B. Smith III[3], Cyrus J. Walker[2]

[1]RedPhone Corporation
9595 Sherburne Farm Road
Marshall, VA 20115, USA
jay.aceto@redphonecorporation.com

[2]Data Defenders, LLC.
10 W. 35th Street, Ste. 9F5-1
Chicago, IL 60616, USA
michelle.m.shafer@gmail.com, cyrus.walker@data-defenders.com

[3]Dominion Voting Systems
1201 18th Street
Denver, CO 80202, USA
ed.smith@dominionvoting.com

Abstract: There are many security challenges associated with the use of Internet voting solutions. While we are not advocating for the use of Internet voting in this paper, we do assert that if an Internet voting solution is going to be used, its deployment must be undertaken with continuous security auditing in place – security auditing that begins with the development of the Internet voting system by the manufacturer or election jurisdiction and continues throughout the system's use in the field.

1 Introduction

There are many security challenges associated with the use of Internet voting solutions. While we are not advocating for the use of Internet voting in this paper, we do assert that if an Internet voting solution is going to be used, its deployment must be undertaken with continuous security auditing in place – security auditing that begins with the development of the Internet voting system by the manufacturer or election jurisdiction and continues throughout the system's use in the field.

One aspect of an election security audit is real-time election forensics, which are currently being used by some election jurisdictions to monitor deployed Direct Recording Electronic (DRE) voting systems.[1] Real-time election forensics is a powerful tool in helping to prevent intrusions as well as identifying damage if a successful intrusion results. It assists the voting jurisdiction in maintaining confidence in the deployed system, and it has the advantage of being executed concurrently with the deployment, deployment testing, and use of the system.

The goal of this paper is to demonstrate how real-time election forensics and other security methodologies successfully used with electronic voting systems can also be used to mitigate risks and detect issues with Internet voting solutions.

2 Highlights of the Product Development Process

2.1 Determining What to Develop

Determining what to develop in relation to Internet voting systems requires a high degree of skill in the product management arena, higher than what is considered the norm in most product development situations, due to existing and emergent standards and threats related to Internet voting systems. The U.S. Election Assistance Commission (EAC) has published draft UOCAVA voting systems guidelines.[2] Requirements and standards published by the EAC form only one part of the requirements for any Internet voting system to be used in the United States. Each state has unique requirements when it comes to conducting elections. A voting system that is expected to be national in scope must include these requirements no matter how esoteric they may seem in statute, and the developers of that system must reconcile conflicting state requirements. Furthermore, the voting system should be able to be used by the entire population, fulfilling the needs of persons with disabilities as well as persons with literacy challenges.

Looking at the development organization, it is imperative to adopt an adaptive requirements development methodology such as the one outlined in the Capability Maturity Model Integration (CMMI) at Maturity Level 3[3]. CMMI Requirements Development, including intense surveillance for emergent information system threats, is a suitable process for deriving system requirements.

---

1 Walker, Cyrus J., *Forensics: The Vital Link in Election Integrity: A Case Study on Cook County, IL*, www.data-defenders.com/wp-content/uploads/pdfs/EIFA-casestudy-online.pdf, 2010.

2 National Institute of Standards and Technology, *High-Level Guidelines for UOCAVA Voting Systems*, www.nist.gov/itl/vote/upload/High-level-Guidelines-Draft-2011-06-21.doc, 2011-06-21 Draft.

3 Software Engineering Institute, Carnegie Mellon University, *Capability Maturity Model Integration* (CMMI) www.sei.cmu.edu/cmmi, 2012.

2.2 Determining How to Develop the System

There are a number of development methods to choose from. It does not matter so much which development method is chosen. Whether it be Waterfall[4], Agile[5], Extreme Programming (XP)[6], or some hybrid approach, all of these methods can lead to functionally secure code. There are publications that describe, independent of development method, how to write secure software[7]. What is most important is that the development method is documented, understood by developers and their management, adhered to, and auditable.

After some foundational training, the developer can be trained on the actual product architecture and the portion of the product they are developing. This same training scheme can be utilized for product testers, with additional material regarding test planning, test methods and automation, the formation of test cases, scripts, and artifacts.

2.3 Risk Management

Once the development method is chosen and the staff trained, it is not time to develop the product but rather to move into risk management for the forthcoming system. Bridging "what to develop" and "how to develop it" (the development method to be used) is the major step in system development known as risk management. Risk management, configuration management, and emergent threat management form the foundation for a robustly developed system. If this triad is not continuously functioning, there can be no secure system development or eventual deployment. Risk management is the process for identifying, analyzing, and communicating risk and accepting, avoiding, transferring, or controlling it at an acceptable level considering associated costs and benefits of any actions taken. Risk management will not preclude an adverse event from occurring; however, it enables organizations to focus on those things that are likely to bring the greatest harm, and employ approaches that are likely to mitigate or prevent those incidents. There are a number of risk management frameworks[8]. ISO 27001[9] requires that organizations adopt the standard practice of risk management with regard to management of its information security.

---

4 Waterfall Model, *Waterfall Model: Advantages, Examples, Phases and More About Software Development*, www.waterfall-model.com, 2012.

5 Poppendieck, Mary and Poppendieck, Tom, *Lean Software Development: An Agile Toolkit*, Addison-Wesley Professional; New York, 2003.

6 Beck, Kent and Andres, Cynthia: *Extreme Programming Explained: Embrace* Change (2nd Edition). Pearson Education; New Jersey, 2005.

7 Howard, Michael and LeBlanc, David, Writing *Secure Code* (Microsoft Press, 2002) is one such publication.

8 Quality Progress*, Safe and Secure: A Case Study*, Vol. 45 number 12 (Jan 2012), 16 – 23.

9 ISO 27001, ISO, Switzerland.

2.4 How to Test the System

Before considering testing the voting system and its component parts, the process used to develop the product must be audited to ensure compliance to its process documentation and to ensure that the documented process has the potential to lead to a secure voting system. This process audit approach has a parallel in-system verification and validation. Verification ensures that the product meets specification; and validation attempts to ensure that the product will work in practice.[10] The process auditor will likewise assess that the process actually employed (as seen through its artifacts) matches its governing documentation. It is likely that a larger group, such as the established or prospective customer of the system or a body such as the EAC, would seek to establish that the process has the potential of birthing a system that meets specifications and can demonstrate a required level of security.[11]

The stages of testing are well known and will not be detailed here except to provide some additions unique to an organization developing secure systems. Product testing typically starts with Unit Testing, sometimes referred to as Developer Testing. A unit is the smallest testable piece of a system[12]. Unit testing is a key activity within an Extreme Programming development environment. Code needs to be assessed during development to ensure that functionally secure code is being produced according to the established development process. Agile development methods provide for a similar outcome by requiring the developer to have work product that is usable or demonstrable after they finish the prescribed work in a given iteration of the product.

Component testing follows unit testing. This phase tests a discrete part or parts of a system – network infrastructure, firewall, and application software. Throughout these portions of the overall test program, it is useful to run static code analysis tools and to utilize other tests, likely customized for the system under development, to further ensure that the basics are being covered. "The basics" implies a code that contains no buffer overflows, dead code, poor stylistic construction, or other fundamental flaws that may or may not be uncovered through downstream functional testing. A system integration test follows to answer the question – can you conduct an election on the system? Voting systems can be developed according to the 2005 VVSG, be secure beyond imagination, and yet completely incapable of processing a jurisdiction's election.

Now that there is a nascent voting system, an intersection of process and product needs to be tested to answer the question of emergent threats. Can the development and configuration management processes manage the emergent threat environment while maintaining configuration control? This is an extremely important question to answer as

---

[10] The ISO 9000 series of standards provide definitions and uses of verification and validation in product realization processes.

[11] IEEE Standards Board, IEEE Standard for Software Unit Testing: An American National Standard, ANSI/IEEE Std 1008-1987 in IEEE Standards: Software Engineering, Volume Two: Process Standards; 1999 Edition; published by The Institute of Electrical and Electronics Engineers, Inc. Software Engineering Technical Committee of the IEEE Computer Society.

[12] Stephens, Matt and Rosenberg, Doug, *Design Driven Testing: Test Smarter, Not Harder*. Springer Science; New York, 2010.

the system moves through the remaining test phases and into deployment and use. Did the manufacturer enact appropriate policies to deal with emergent threats? Is there an adequate level of surveillance and expertise to deal with the emergent threats and transfer the needed upgrades to the product? Are these processes scalable so that the deployed system can also see the same degree of success against emergent threats that the evolving (pre-release) system enjoyed?

At a defined point in its development, that point being defined by a release process and acceptance criteria, the system begins verification testing. In a sense, verification testing has been in progress throughout the development of the product, answering the question – does the product meet the specifications, especially functional security specifications? In this phase, in contrast, the system undergoes verification as a system in an environment mimicking deployment. Verification continues to include security testing and other sorts of negative path test cases; however, most of the work at this stage will be "happy path", examining parameters such as accuracy, but not under stress or attempts to misuse the system. Validation, on the other hand, will be tied to conditions the system will face in deployment. This means adversarial testing, volume/stress testing while maintaining a secure posture and required accuracy, and enhanced accessibility and usability testing (not just line by line VVSG compliance, but sessions with a body of test subjects). While there must be bi-directional traceability from validation test cases to product requirements, the test manager will see validation activities mushroom relative to the number of activities and hours spent in unit, component, system integration, and verification testing. Significant problems during validation would likely result in re-architecture and subsequent re-development of the voting system, or possibly lead to it being scrapped in favor of an entirely new approach. The ability to develop creative test cases that test beyond conventional ways of thinking about system use is quite valuable to ensuring a secure system.

3 Security Testing of Voting Systems Methodology

3.1 Information Gathering – Internal and External Processes and Procedures

It is a well-established fact that organizations that have defined practices for their internal and external processes are less vulnerable to attack, faster to react if attacked, and forensically capable of identifying the vector of the attack (not to mention more efficient and ultimately more competitive with a higher degree of software quality assurance[13]). Organizations that clearly follow established internal and external processes are also easier for third parties to evaluate. When determining whether security vulnerabilities exist, or if and where improvements can be made that minimize vulnerabilities, having documented, established internal and external processes is vital.

[13] Capability Maturity Model Integration (CMMI), Software Engineering Institute, Carnegie Mellon. www.sei.cmu.edu/cmmi.

A quick review of the AICPA website [14] will show that process evaluation is a two-step effort; first you document and list the processes, than you evaluate them. Failure to adopt formal development and testing methodologies such as the CMMI, ISO27000 and 9000, or the Open Web Application Security Project (OWASP)[15] slows system development, causes redesign, redevelopment, failure to meet security requirements, and significantly increases the final cost of the delivered system.

Each of these methodologies has defined a process for capturing and evaluating the internal and external processes that voting system evaluators can use to uncover risks throughout the software lifecycle. It is essential that the testing effort be continuous, not a point-in-time analysis of an application's security profile. Security must be integrated early to be most successful and must be continuous to be relevant to the changing landscape of threats and vulnerabilities. Analyzing internal and external processes and requirements becomes a gap analysis between corporate processes and industry-recognized processes and best practices.

3.2 Identification and Analysis of High-level Components and Information Flow

"White hat" testing, which involves the support of the voting system manufacturer's staff up to senior leadership, is often employed. Under these circumstances, network diagrams and system component lists, including operating system versions, router Internetwork Operating System (IOS) versions, firewall logs, ports, protocols and services, etc., are demanded by testers so that an accurate inventory of all components that support the voting system exists. This is a portion of the testing and verification phase focused more on the implementation environment.

Both passive (examination) and active (testing) techniques exist for discovering devices on a network. Passive techniques use a network sniffer, such as NMAP, to monitor network traffic and record the IP addresses of the active hosts. These sniffers can report which ports are in use and which operating systems have been used on the network. Passive discovery can also identify the relationships between hosts—including which hosts communicate with each other, how frequently their communication occurs, and the type of traffic that is taking place—and is usually performed from a host on the internal network where it can monitor host communications. This is done without sending out a single probing packet. Passive discovery takes more time to gather information than active discovery, and hosts that do not send or receive traffic during the monitoring period might not be reported accurately. Both active and passive discovery have benefits and potential drawbacks but are very important to utilize.

---

[14] American Institute of CPAs, *Statements on Auditing Standards*, www.aicpa.org/Research/Standards/AuditAttest/Pages/SAS.aspx

[15] *The Open Web Application Security Project* (OWASP), www.owasp.org

3.3 Develop Misuse Cases for Violating the Assumptions

Misuse cases come within the security requirements process, which consists of (1) identifying critical assets, (2) defining security goals, (3) identifying threats, (4) identifying and analyzing risks, and (5) defining security requirements. Unlike the software development process, where the focus is on "use cases," the security testing focus is on "misuse cases," or more specifically, how to break the system and/or usurp the security and gain access to data or system administrative functions. After identifying the operating systems, manufacturer of components within the system, and internal and external processes, we look at ways we can covertly or overtly take control or alter voting data either at rest or in transit. A misuse case describes "a sequence of actions, including variants, which a system or other entity can perform, interacting with misusers of the entity and causing harm to some stakeholder if the sequence is allowed to complete." The details of use cases are usually captured in text-based forms or templates. These are important because they encourage developers to write clear, simple action sequences. The focus of misuse cases is on the disruption of any one of three primary objectives: the confidentiality, availability or integrity of the system, and supporting data. The corruption of any one will result in a system failure and lost voter confidence. Therefore, misuse cases should always be targeting one of these three security objectives.

3.4 Identification of Threats and Attack Exposures

The threat modeling process can be broken down into three high-level steps, which include decomposing the application, determining and prioritizing threats, and the identification of potential mitigations. The first step in the threat modeling process is to gain an understanding of the application and how it interacts with external entities by leveraging misuse, abuse, and use cases to understand how the application is intended to be used; identifying entry vector points to see where a potential attacker could interact with the application (voter, poll worker, or system administrator etc.); identifying assets, i.e., hardware, operating systems, internal and external processes that the attacker would be interested in, and identifying trust levels that represent the access rights the application will grant to external entities. The data flow diagram should show the different paths through the system, highlighting the privilege boundaries. Development organizations may overlook this diagram.

In the second phase, identified threats are categorized and ranked using a methodology like the NIST approach outlined in the NIST SP800-30 Risk Management Guide for Information Systems or the threat categorization methodology developed by Microsoft called STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of privileges). Another very useful approach is the Application Security Framework (ASF), which defines threat categories such as auditing and logging, authentication, authorization, configuration management, data protection in storage and transit, data validation, and exception management. No matter which one is used, the goal of the threat categorization is to identify threats from both the attacker's and the defender's perspective.

Finally, countermeasures and mitigation must be examined. A lack of protection against a threat might indicate a vulnerability whose risk exposure could be mitigated with the implementation of a countermeasure. Such countermeasures can be identified using threat-countermeasure mapping lists. Once a risk ranking is assigned to the threats, it is possible to sort threats from high risk to low risk and prioritize the mitigation effort based on cost, impact, end-user use cases, etc.

3.5 Election System Threat Model Analysis

The threat model analysis for an election system indicates there are two equally potent threat sources:

- The Malicious Insider - One with malicious intentions, who developed a portion of the system and/or has been granted direct access to the deployed system. The malicious insider is the more dangerous and potent of the two threat sources.
- The Malicious Outsider - The Malicious Outsider, one with malicious intentions who attempts to gain access to the systems from outside the system operator's domain of control.

Each threat source has two main goals: minimizing exposure and maximizing impact. The means by which either threat source attempts to execute their threats against the electronic voting system depends on the state of threat model variables.

The threat opportunity for the malicious insider is generally at its peak during phase 1 and phase 2 of an election jurisdiction's election management workflow as shown in figure 1. Generally, in these phases of the election management workflow, the majority of the components of the electronic voting system are being prepared for use in an election, requiring the greatest amount of system access. As a result, a skilled and prepared malicious insider could infiltrate the system and insert foreign components, such as code, into the electronic voting system to cause it perform in a way that violates its predetermined and intended functionality.

The threat opportunity for the malicious outsider is generally at its peak during phase 3 of an election jurisdiction's election management workflow (figure 1). In this phase of the election management workflow, any publically accessible components of the electronic voting system are deployed into the field for use in an election. A skilled and prepared malicious outsider could gain access to these publically accessible components such as DREs and insert foreign components such as code into these components to cause it perform in a way that violates its predetermined and intended functionality.[16]

[16] There are a number of reports in the California Top to Bottom Review of Voting Systems from 2007. The referenced material can be found in the various source code review and red team reports from that Review. These are located at: http://www.sos.ca.gov/voting-systems/oversight/top-to-bottom-review.htm.

All these types of threats could go undetected if there are no regular checks and balances in place to validate the operational integrity of each voting system component at each step in the election management workflow.

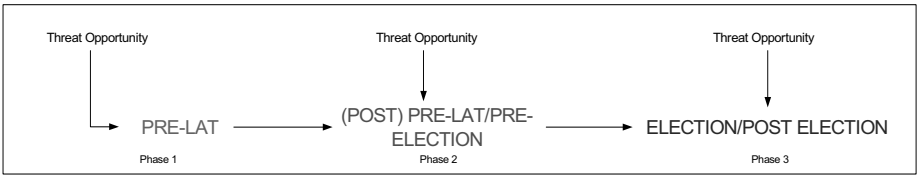

Fig. 3.: Simplistic Election Management Workflow Threat Model

4 The Importance of Election System Risk Analysis to the Forensic Auditing Process

Election system forensic auditing is a tool that can be used to mitigate the risks or operational threats against a voting system. This tool is best used when implemented as part of an overall election system risk management and mitigation strategy.

While the computer forensics process examines every part of an election system, voting secrecy is still maintained because election systems do not include voter identifiable information with ballots. The goal of the computer forensic process is to examine the election system from the bit level to detect how the smallest changes made to a system may have negative implications. It is analogous to examining the trees in a forest: once you find that out-of-place tree then you can examine the tree as well as the forest that the tree grows in. In the election system world, once a subset of data is discovered to be out of place, then the data itself can be examined as well as other characteristics, such as other occurrences of the data sample in other aspects of the systems and the impact of the data on sample on the system. The forensic auditing process can use threat modeling information as part of an operational/functional baseline for each component of the electronic voting system and incorporated threat signatures, which can be used to identify the manifestation of a threat against an election system component. This enables the most accurate validation of the operational integrity of an election system to ensure that no threats could negatively impact the operations of the electronic voting system.

Once the risk assessment has been completed, forensic auditing can be used to examine every component of the electronic voting system at the bit level, even dynamic software files heretofore considered untouchable by analytical tools. The forensic auditing process starts by developing an accurate baseline of the operations of the voting system.

4.1 Election System Baselining

System baselining is used to establish a functional benchmark of a system that can then be used to measure and determine the operational integrity of the system during actual use. The system baselining process can be used to establish functional benchmarks of DRE-based or Internet-based voting system. A typical system baseline consists of the following components:

- File System Structure
- Static and Dynamic File Delineation
- Dynamic File Range of Change
- Identified System State Transition

While not every function or capability of a static file is executed during routine system operations, the static file itself will not change at any time during routine system operations unless some other program function legitimately caused it to change, for example, program or system updates. Therefore, the behavior of a static file is limited and can easily be characterized.

Dynamic files are designed to change based on routine system operations. The presence of dynamic files should not be intimidating as, generally, the range of change of the dynamic file is limited and based on the routine system operation, which is limited, and as such, the range of change can be defined and measured. Log files are considered dynamic in practice and under the EAC definition can be found in VVSG 2005, Volume I, section 7.4.

One threat common to all system models is the threat against dynamic files. Because dynamic files are generally designed to change during normal routine system operation, if a malicious change is made to a dynamic file, that change would be difficult to identify unless the expected changes of a dynamic file have been delineated and used to validate actual changes made to dynamic files during routine system use.

File behavior is limited based on the limited set of routine system operations; thus, file behavior can be measured, captured, and used to validate future file behavior measurements to determine if those measurements are based on legitimate or malicious system activities.

4.2 Forensic Auditing Process Implementation

Forensic auditing is not about trust or the lack thereof; it is about validation. The only way to absolutely guarantee the operational integrity of a system is to completely eliminate all access to the system. That is clearly not feasible. Therefore, if there is any access to the system, validation of the operational integrity must also be executed to ensure that the operational integrity of the system.

One valuable benefit of forensic auditing is that no component of the forensic auditing process needs to be installed on any component of the electronic voting system during the audit process.

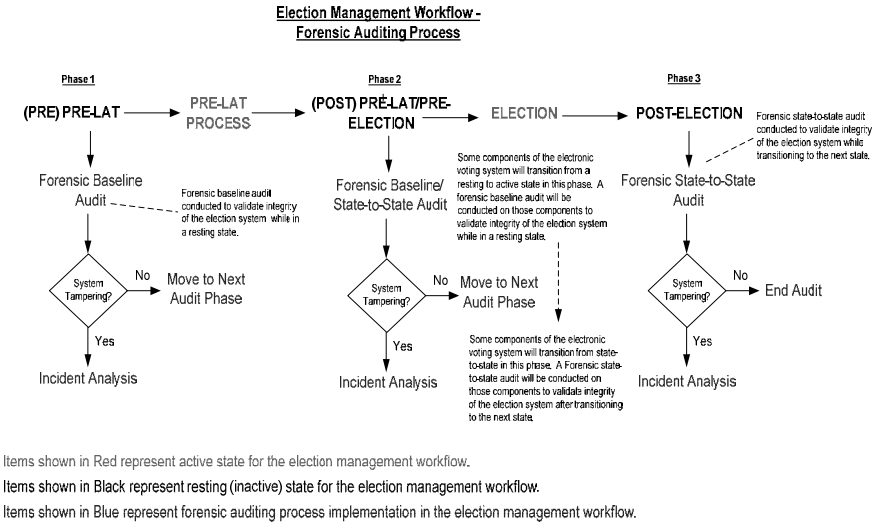

Fig. 4.: Election Management Workflow & Forensic Auditing Process

The election management workflow is cyclical: the electronic voting system usage is commensurately cyclical. There are significant periods of time where the election system is not used and simply awaiting the next election cycle.

The process of forensic auditing consists of taking samples of data from target electronic voting system components at various intervals in the election management process. Each data sample collected is analyzed by comparing that sample of data to a "known good state" of data contained in that sample, in order to identify and validate the integrity of changes made to that data sample as a result of normal, routine system operations or to identify anomalies (unexpected changes) in the data sample made by foreign code or components inserted into the system, which both have the effect of negatively impacting the operational integrity of the electronic voting system.

"Known Good States" are data samples that have been taken from a number of sources including election system manufacturers, Voting System Test Laboratories (VSTLs), and data samples from a clean, unused state of the target system. Each clean sample of data is assembled into a single "Known Good State" baseline for the target device and used to validate the integrity of the data samples taken from that device during a forensic audit. Analyzing each data sample consists of conducting a "Resting State" to "Baseline" comparison or a "State-to-State" comparison to identify and validate changes made in the data sample.

General computer forensic methods such as acquiring data samples and generating hash values for that data, are used to ensure that the integrity of the data sample is maintained and can be validated at any point in the analysis process. This ensures that none of the analytic processes made changes to the data sample, which could lead to inaccurate results. The goal of the analysis is to validate that known static files were unchanged and that the changes made to dynamic files were valid and according to forensic audit expectations.

When forensic auditing is used and implemented in the manner previously described, it can serve as a detection function, detecting if the operational integrity of the electronic voting system has been impacted in any way. Additionally, with the forensic auditing function being regularly executed on the electronic voting system, it serves to deter the malicious insider as a result of its recurring implementation.

5 Conclusion

Designing security into the Internet voting system is extremely important. A suitable methodology includes internal and third party assessment of risk management competency, development and test process documentation, and adherence to that documentation. The development and deployment team for Internet voting must have a superior system for recognizing, assessing, and managing emergent threats to the voting system.

Process and product (voting system) auditing alongside continuous, multi-pronged testing from the development stages through implementation is critical for any voting system – prior to, during, and after each voting system use.

Forensics must be used before and during system deployment to identify intruders, aid in stopping their malicious efforts, and delineating any damage a successful intrusion might have caused.

These efforts, product and process auditing, unit through system testing, and forensic analysis are being utilized on hardware-based electronic voting systems, and we assert that these same methodologies will assist in guarding against and detecting security issues associated with internet voting systems.

Bibliography

[Epp12] Dana Epp, Microsoft MVP Enterprise and Developer Security, "The Evolution of Elevation: Threat Modeling in a Microsoft World", 2012. http://technet.microsoft.com/en-us/security/hh778966.aspx

[Sin05] Sindre, G and Opdahl, A. L., *Eliciting Security Requirements with Misuse Cases*. Requirements Engineering Journal, 10(1):34–44, 2005.

[Wal11] Walker, Cyrus J., *A Case Study of Real-Time Election Forensics,* Data Defenders, 2011. www.data-defenders.com/wp-content/uploads/2011/07/A-Case-Study-of-Real-time-Election-Forensics-FINAL.pdf