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

**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 is currently being used by some election jurisdictions to monitor deployed Direct Record Electronic (DRE) voting systems, also known as electronic 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.

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

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 is the norm in most product development situations due to existing and emergent standards and threats related to Internet-based voting systems. The U.S. Election Assistance Commission (EAC) has published both the 2005 VVSG version 1.0 as well as draft UOCAVA voting systems guidelines.[2] The 2005 VVSG does not specifically address Internet Voting or electronic ballot delivery systems, so its use as a base set of requirements is questionable.

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 its own, often very different, requirements for the conduct of Primary Elections, language display, instructions to the voter, and the presence or absence of voting flavors such as straight party voting. A voting system that is expected to be national in scope must include these requirements no matter how esoteric or benign they may seem in statute, and the developers of that system must reconcile conflicting State requirements. Furthermore, the voting system must be usable by the entire population, taking into full

[2] "High-Level Guidelines for UOCAVA Voting Systems", 2011-06-21 Draft (NIST) - http://www.nist.gov/itl/vote/upload/High-level-Guidelines-Draft-2011-06-21.doc

account the needs of persons with disabilities and persons with literacy and other language challenges.

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

### 2.2 Determining how to develop the system

There are a number of development methods to choose from, each offering advantages and disadvantages to the voting systems development organization. It matters not so much which development method is chosen, be it Waterfall[4], Agile[5], Extreme Programming (XP)[6], or some hybrid approach. Similarly, there are development method independent works describing how to write secure software[7]. What is most important is that the development method is documented, understood by developers and their management, followed 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

---

[3] *"Capability Maturity Model Integration (CMMI)",* Software Engineering Institute, Carnegie Mellon - "http://www.sei.cmu.edu/cmmi/
[4] *"Waterfall Model",* Wikipedia - http://en.wikipedia.org/wiki/Waterfall_model
[5] "Agile Software Development", Wikipedia - http://en.wikipedia.org/wiki/Agile_software_development
[6] "Extreme Programming", Wikipedia - http://en.wikipedia.org/wiki/Extreme_programming
[7] *Writing Secure Code* by Michael Howard and David LeBlanc (Microsoft Press, 2002)

scheme can be utilized for product testers, with additional material regarding test planning, test methods and automation, the formation of test cases, scripts, artifacts, and the current status of product test. Once the development method is chosen and the staff trained, it is not yet time to develop the product, but instead to move into risk management for the forthcoming system.

### 2.3 Risk Management

Bridging "what to develop" and "how to develop it" 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. Without this triad functioning continuously, there can be no secure system development or eventual deployment. There are a number of risk management frameworks, and organizations are developing new frameworks and combining existing frameworks to advance the state of the art[8]. ISO 27001[9] requires that organizations adopt the standard practice of risk management with regard to management of its information security.

### 2.4 How to test the system

Before considering product testing, that is, 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

---

[8] Quality Progress article; Vol 45 number 12 (Jan 2012) pages 16 – 23 *"Safe and Secure: A Case Study"*

[9] ISO 27001, ISO, Switzerland

validation attempts to ensure that the product will work in practice.[10] The process auditor will likewise seek to 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 later in the process demonstrate a level of security.[11]

The stages of testing are well known and will not be detailed here except to provide some additions that one would expect to find in 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. 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, a firewall or router, the 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 custom to the system under development,

---

[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] "Unit Testing", Wikipedia - http://en.wikipedia.org/wiki/Unit_testing

to further ensure that the basics are being covered. By "the basics", this implies 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. System integration follows component testing and answers the question – can you run an election on the system. Voting systems can be developed to the 2005 VVSG, be secure beyond imagination, and yet completely incapable of processing a jurisdiction's election definition and results reporting needs.

Now that there is a 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 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? Is the process 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. In this phase; in contrast, the system undergoes verification as a system in an environment mimicking deployment.

Verification should start to include security testing and other sorts of negative path test cases; however, most of the work at this stage will be “happy path”, looking at such parameters 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 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 test. Validation is the phase of test where real confidence can be gained in the system’s capabilities. That confidence can be lost with repeated failures during validation. Significant validation problems would likely result in significant re-architecture and subsequent re-development of the voting system, or possibly it being scrapped in favor of an entirely new approach.

# 3 Security Testing of Voting Systems Methodology

## 3.1 Information gathering – internal & external process 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. Organizations that clearly follow established internal and external processes

are also easier to third party 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.

A quick review of the AICPA website [13] 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, the ISO27000 and 9000, or the Open Web Application Security Project (OWASP)[14] slows system development, causes redesign, redevelopment, and failure to meet security requirements, and significantly increases the final cost of the delivered system.

Each of these methodologies have defined a process and checklists for capturing and evaluating the internal and external processes that voting system evaluators can use to determine the risks during the software lifecycle. It is essential that the testing effort be continuous, not a point in time analysis of an applications security profile. Security must be integrated early to be the most successful and must be continuous to be relevant to the changing landscape of threats and vulnerabilities. In the end, analyzing internal and external processes and requirements is a simple gap analysis between corporate processes, and industry recognized processes and best practices. Now that we've established what methodologies are best suited for the internal and external process evaluations, let's now look closer at how to identify the high-level components and information flow.

---

[13] American Institute of CPAs - http://www.aicpa.org/Research/Standards/AuditAttest/Pages/SAS.aspx
[14] OWASP, Wikipedia - https://www.owasp.org/index.php/Main_Page

### 3.2 Identification and analysis of the high-level components and information flow

Often “white hat” testing is employed, which involves the open support of the voting system vendor system administrators, network architects, and senior business leaders and data center service providers. 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 asked for by the testers so that an accurate inventory of all components that support the voting system exists.

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, and 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 does 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.

### 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 and simple action sequences. The focus on misuse cases is on the disruption of any one of three primary objectives, the corruption of 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 objectives.

### 3.3 Identification of threats and attack exposures

The threat modeling process can be decomposed into three high-level steps that include decomposing the application, determining and prioritizing threats, and 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, as described

above: 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. We often find that this last part is lacking completely.

In the second phase, identified threats are categorized and ranked using methodology such as 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 or Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of privileges. Another very useful approach is the Application Security Frame (ASF) that defines threat categories such as auditing & logging, authentication, authorization, and configuration management, data protection in storage and transit, data validation, exception management. No matter which one is used, the goal of the threat categorization is to identify threats from both the attacker and the defensive perspective.

Finally, countermeasures and mitigation are 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 the highest to the lowest 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 potential threat sources:

- The Malicious Insider - The Malicious Insider, one with malicious intentions who has been granted direct access to the system. The malicious insider is the most 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 of 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 the other threat model variables.

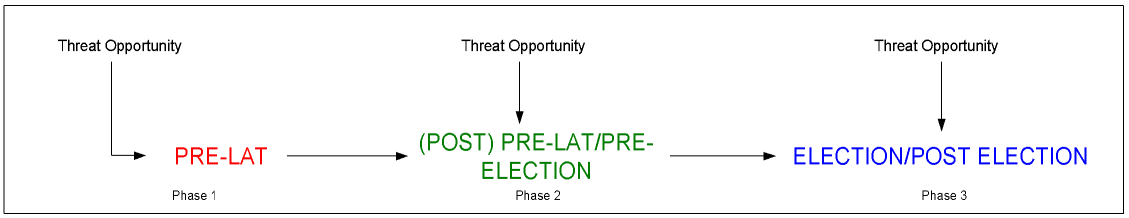
The threat opportunity for the malicious insider is generally at its peak during the phase 1 and phase 2 of an election jurisdiction's election management workflow as shown in figure 1. Generally, in these phases of the election management workflow, the majority of the components of the electronic voting system are being prepared for use in an election and requires 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 it peak during phase 3 of an election jurisdiction's election management workflow (figure 1). Generally, in this phase of the election management workflow, the publically accessible components of the electronic voting system are deployed into the field for use in an election. As a result, 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.[15]

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

**Figure 1. Simplistic Election Management Workflow Threat Model.**




[15] http://www.sos.ca.gov/voting-systems/oversight/top-to-bottom-review.htm

## 4.0 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 a part of overall election system risk management and mitigation strategy.

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 that 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 were successful in negatively impacting the operations of the electronic voting system.

Once the risk assessment has been completed, forensic auditing can be used to examine in detail 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 and clean baseline of the operations of the electronic 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. This 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 such as program or system updates. Therefore, the behavior of static file is limited and can easily be determined.

Dynamic files are designed to change based on the routine system operation that caused it to change. 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, found in VVSG 2005, Volume I, section 7.4.

One threat common in all system models is 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.

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

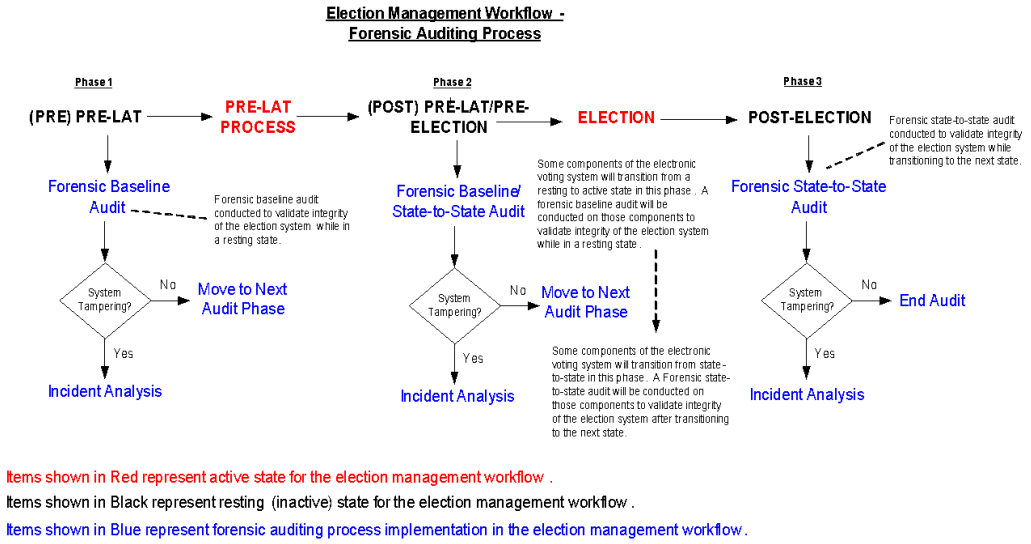
### 4.2 Forensic Auditing Process Implementation

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

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.

.

**Figure 2. Election Management Workflow & Forensic Auditing Process.**




The election management workflow is cyclical, thus the electronic voting system usage is commensurately cyclical. There a number of periods of time where the election system is not used or in a resting state waiting of the next election cycle.

The process of forensic auditing consists 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 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 that have the affect 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 analysis processes made changes to the data sample that could lead to inaccurate results. The goal of the analysis process is to validate that known static files were unchanged and that the changes made to dynamic files were valid and according to forensic audit expectation.

The concept of implementing forensic auditing on DRE-based or Internet-based voting systems is the same. When forensic auditing used and implemented in the manner described previously, it can serve as a detection function to detect of the operational integrity of the electronic voting system have 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 it recurring implementation

## 5 Conclusion

Designing security into the system is extremely important. This methodology includes internal and third party assessment of risk management competency, process documentation, and adherence to that documentation.

Continuous, multi-pronged testing from development through implementation and beyond is critical for any voting system – prior to and after each voting system use - election. Testing must be continuous and third-party testing is crucial to stay on top of new threats and effective mitigation strategies.

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 may have caused.

These efforts are being utilized on electronic voting systems and we assert that this same type of methodology will assist in guarding against and detecting security issues and associated with internet voting systems.

## Bibliography

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

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

Walker, Cyrus J. *"A Case Study of Real-Time Election Forensics"* Data Defenders (2011) - http://www.data-defenders.com/wp-content/uploads/2011/07/A-Case-Study-of-Real-time-Election-Forensics-FINAL.pdf