Are Your Votes *Really* Counted? Testing the Security of Real-world Electronic Voting Systems

Davide Balzarotti, Greg Banks, Marco Cova, Viktoria Felmetsger,
Richard Kemmerer, William Robertson, Fredrik Valeur, and Giovanni Vigna
Computer Security Group
University of California, Santa Barbara
{balzarot,nomed,marco,rusvika,kemm,wkr,fredrik,vigna}@cs.ucsb.edu

ABSTRACT

Electronic voting systems play a critical role in today's democratic societies, as they are responsible for recording and counting the citizens' votes. Unfortunately, there is an alarming number of reports describing the malfunctioning of these systems, suggesting that their quality is not up to the task. Recently, there has been a focus on the security testing of voting systems to determine if they can be compromised in order to control the results of an election. We have participated in two large-scale projects, sponsored by the Secretaries of State of California and Ohio, whose respective goals were to perform the security testing of the electronic voting systems used in those two states. The testing process identified major flaws in all the systems analyzed, and resulted in substantial changes in the voting procedures of both states. In this paper, we describe the testing methodology that we used in testing two real-world electronic voting systems, the findings of our analysis, and the lessons we learned.

Categories and Subject Descriptors: D.4.6 [OPERATING SYSTEMS]: Security and Protection; D.2.5 [SOFTWARE ENGINEERING]: Testing and Debugging—*Testing tools (e.g., data generators, coverage testing)*; D.2.11 [SOFTWARE ENGINEERING]: Software Architectures—*Domain-specific architectures*

General Terms: Security

Keywords: Voting systems, DREs, Security testing

1. INTRODUCTION

Electronic voting systems have been introduced to improve the voting process. Since their inception, they have been controversial, because both the technologists and the general public realized that they were losing direct control over an important part of the voting process: counting the votes.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ISSTA'08, July 20–24, 2008, Seattle, Washington, USA.
Copyright 2008 ACM 978-1-59593-904-3/08/07 ...$5.00.

A quote attributed to Stalin says: "Those who cast the votes decide nothing. Those who count the votes decide everything." It is clear that voting systems represent a critical component of a democracy. Although the consequences of a malfunctioning electronic voting system are not as readily apparent as those for air traffic control or nuclear power plant control systems, they are just as important, because the well-being of a society depends on them.

While most critical systems are continuously scrutinized and evaluated for safety and correctness, electronic voting systems are not subject to the same level of scrutiny. A number of recent studies have shown that most (if not all) of the electronic voting systems being used today are fatally flawed [18, 22, 34] and that their quality does not match the importance of the task that they are supposed to carry out.

For example, a report published in January 2008 describes the problems encountered in Sarasota County, Florida, when counting the votes in the November 2006 Congressional District 13 election [23]. In this case, 17,846 ballots (14.9% of the total number of votes) cast on electronic voting machines showed no vote for either candidate in the race. In addition, the race was determined by only 369 votes. The report described the system responsible for recording and tallying the votes as a "badly designed, shoddily-built, poorly maintained, aging voting system in a state of critical breakdown."

Up until recently, electronic voting systems have been certified by third-party evaluation companies. Most of the time, these companies test the general functionality of the systems, their usability, and their accessibility. However, no substantial security testing was performed in the past to identify serious, system-wide security flaws.

Recently, a number of states (in particular California, Ohio, and Florida) have commissioned studies to test the security of the electronic voting machines to be used in forthcoming elections.

Our team was involved in the California Top-To-Bottom Review (TTBR) in July 2007 [31] and in Ohio's Evaluation & Validation of Election-Related Equipment, Standards & Testing (EVEREST) in December 2007 [19]. In the former, we evaluated the Sequoia voting system, while, in the latter, we evaluated the ES&S system. Our task was to identify, implement, and execute attacks that could compromise the confidentiality, integrity, and availability of the voting process. As a result of the security testing performed in these studies, the systems used in California were decertified and those used in Ohio were recommended for decertification.

In this paper, we present the methodology that we developed in the security testing of these voting machines, the results of our analysis, and the lessons learned in the process of testing real-world voting systems. Electronic voting systems have peculiar characteristics that make the testing of their security particularly challenging. We believe that our experience can help other researchers in this field, as well as policy makers and certifying organizations, develop more rigorous procedures for the security testing of voting systems.

The rest of this paper is structured as follows. In Section 2, we provide an overview of electronic voting systems, and we describe the challenges that one faces when testing the security of real-world voting systems. In Section 3, we present, in detail, our experience with the security testing of real-world electronic voting systems. In particular, we present our testing methodology, the testing techniques and tools that we developed as part of the process, and the findings of our analysis. Then, in Section 4, we describe the lessons we learned in the process. Finally, Section 5 presents related work, and Section 6 briefly concludes.

2. ELECTRONIC VOTING SYSTEMS

Electronic voting systems are complex distributed systems, whose components range from general-purpose PCs to optical scanners and touch-screen devices, each running some combination of commercial off-the-shelf (COTS) components, proprietary firmware, or full-fledged operating systems. In this section, we present a description of the components that most frequently are part of an electronic voting system. Then, we describe what the peculiarities of these systems are and why testing their security is challenging.

2.1 Components of the system

The components of an electronic voting system are:

- DRE - Direct Recording Electronic voting machine. A device to record the voter's choices. This is usually a touch-screen device where the voter casts his/her vote.
- VVPAT - Voter-Verified Paper Audit Trail. A paper-based record of the choices selected by the voter. The VVPAT printer is hooked to the DRE and the paper record is viewable by the voter, but it is under a transparent cover so that it cannot be modified other than through the normal voting process.
- EMS - Election Management System. The system responsible for the initialization of the components that collect the votes and also for the final tallying of the votes. The EMS is usually located at election central.
- Optical Scanner. An optical reader that counts votes cast on paper ballots. There is usually one scanner at each polling site and one at election central (e.g., for the counting of absentee ballots).
- DTD - Data Transport Device. Storage devices to transfer data between different components of the systems. These devices are used to transport ballot information to the DREs and optical scanners at the polling site and to transport voting results to the EMS.

Prior to the election, ballot information is prepared on the election management system at election central. This information may be directly entered into the DREs and the optical scanners, or it may be written onto DTDs that are sent to the polling places, separate from the DREs and scanners. Paper ballots for each of the polling places are also prepared at election central.

On election day, prior to the start of the voting process, if the DREs and optical scanners were not initialized at the central location, then they are initialized with the appropriate ballot information at the polling site, using the DTDs that were sent to the polling place separate from the DREs and scanners. After the DREs are initialized (or simply powered up, if they were initialized at election central), they are tested with sample votes to see if they record everything accurately. The optical scanners are tested in a similar way. If the DREs and the scanners pass the pre-election testing, then they are ready to be used for voting.

When a voter comes to the polling place, he/she registers at a desk. Then, either the voter is given a token (e.g., a smart card) to insert into the DRE to start voting, or the election official carries the token and inserts it into the DRE on the voter's behalf. In the case of the voter carrying the token, he/she removes the token and returns it to the election official when he/she is finished voting. If the election official initiates the voting session, the token is usually removed before the voter starts to cast his/her ballot. The voter's choices are displayed on the DRE screen and are also printed on the VVPAT.

If paper ballots are used, the voter is given a ballot and a marking device to cast his/her vote. When the voter is through, the ballot is handed to an official who inserts it into the optical scanner to be read and recorded. Some optical scanners will report an undervote (voting for less than n choices when n are supposed to be picked) or an overvote (voting for more than n choices when n is the maximum number that can be marked). If this is the case, the voter is given the opportunity to correct his/her vote.

After the election is closed, the results from each of the DREs and scanners at a polling place are collected on a DTD and returned to election central, where they are read into the election management system to produce a tally for the entire area.

2.2 How voting systems differ from other systems

Electronic voting systems differ from other types of systems in a number of ways. An important difference is that their results are hidden from the user. For instance, if one is interacting with an ATM, the user has the cash disbursed by the machine along with the receipt to verify that the transaction occurred correctly. With an electronic voting machine, the most that voters receive is a receipt indicating that they voted. Of course, the receipt never records a vote, because if the voter were to receive such a receipt indicating who he/she voted for, then votes could be purchased or coerced.

Furthermore, with electronic voting systems, failures are not apparent because the results are hidden from the voter. That is, even if the system were not behaving maliciously, the DRE can make mistakes due to configuration problems, such as an inaccurate touch-screen calibration (i.e., the voter touches near the desired icon, but the vote is given to the candidate or party associated with a neighboring icon). This kind of problems can have an enormous effect on the election results. Of course, these errors would be indicated on the

tally screen and on the VVPAT, but experiments have shown that these summary screens and the VVPAT are seldom carefully reviewed by the voters [4, 29].

To prevent voting systems from being compromised, physical security is of great concern. The systems are locked in warehouses, which often require two-person controls to enter. In addition, every component of a voting system is accompanied by a "chain of custody receipt" in order to generate an audit trail of who had access to which components at what time. Unfortunately, voting systems are often delivered to polling places a week or more ahead of election day, and in the interim are often stored in an insecure environment, such as a school gym or an election official's garage. Such practices represent an obvious "weak link" in the chain of custody.

In addition to physical security, much of the security of the voting process is dependent on the poll workers following explicit procedures. Unfortunately, most of the personnel that are required to carry out these procedures have very limited IT training and are not capable of dealing with problems that may arise when using electronic voting systems. The solution to this often is to have a representative from the voting system vendor on site (or at least on call) to deal with IT problems that may arise.

Since voting machines are so critical to our democracy, there is a strong desire to assure that they perform correctly. Currently the assurance of these systems involves a lengthy certification process. This certification process is a double-edged sword. Because it takes so long for (re)certification, vendors are often slow to apply patches to their systems. The result is that vulnerabilities are not fixed as soon as they should be, and vulnerable systems are widely deployed. In a testimony before the U.S. House of Representatives [32], Wagner presented a number of problems with the certification process: there is a conflict of interest, because the required certification process performed by testing authorities is paid for by the vendors; there is a lack of transparency, since the reports are generally not publicly available; certification does not include the testing and enforcing of required standards; and there is a lack of a clear decertification path for systems that fail testing.

Certification standards have their own set of problems: they lack a clear system and threat model, they often propose seemingly arbitrary specifications, and sometimes they mandate impossible features [2].

2.3 Security testing of voting systems

Security testing is generally an overlooked and under-appreciated part of the testing process as a whole. The reason for this deficiency stems from many factors. First, the majority of software developers are not security experts, or even security-aware. This leads to software with "bolted-on" security, poorly implemented security, or no security whatsoever. Second, software testing engineers and the organizations that employ them are concerned with proper execution in response to use-cases and the advertised functionality of a product. Exceptional and hostile environments are usually not considered in the testing process; therefore, security holes are not discovered. Finally, the security of large systems with many developers is often hard to assess, because it requires knowledgeable individuals who are able to understand how one could leverage the complex interactions between the components to bring the system into an unintended and vulnerable state.

The aforementioned characteristics of voting systems imply three important consequences that necessitate proper design and security evaluation. First, the presence of sensitive election information makes the threat of well-funded and motivated attackers a real concern. Second, the distributed nature, complex design, and reliance on proper execution of operational procedures all serve to create a wide and varied attack surface. Finally, the public's relation to voting systems, both in their use and in their effect on the future direction of society, makes public perception and confidence of primary importance when testing these systems. All of these factors, combined with historical implementation issues, set the testing requirements for electronic voting systems apart from the testing of other systems.

In order to ensure the public's confidence in a voting system, a rigorous, objective, and publicly accessible test procedure and report must be developed. Reassurance by the vendor is necessary, but cannot be considered sufficient in this case. Instead, the system of checks and balances that are important in any public arena should also be applied here.

Although the certification process can help validate the proper functioning of a voting system under ideal conditions, real-world deployments often rely on operational procedures for this assurance. However, these operational procedures cannot be the only measure guaranteeing security. Instead, there should be safeguards built-in at both the software and physical layers for cases in which these procedures are not carried out correctly or in good faith. For these reasons, the relationship between the operational procedures, their effect on information flow, and the overall security of the system must be carefully analyzed.

One of the biggest frustrations to the potential testing of current electronic voting systems is that they are proprietary in both hardware and software. In many cases, this makes obtaining source code, documentation, and build environments very hard. In addition, the technologies that are used can be very old and outdated, making the reproduction of a suitable test environment nearly impossible. For these reasons, the resources that are available to a potential objective tester are usually severely constrained, enabling only black-box testing where white-box or gray-box testing is appropriate.

3. EXPERIENCE IN TESTING REAL-WORLD VOTING SYSTEMS

In our experiments, we tested the security of two electronic voting systems: the Sequoia system for the California TTBR and the ES&S system for the Ohio EVEREST project. The two vendors have their own proprietary name for each component of the system. However, to avoid unnecessary confusion, we will refer to each of them using the general terminology introduced in Section 2.

In the following, we first introduce the general methodology we used in our analysis. Then, we focus on the tools we developed to cope with some of the problems that characterize the testing of electronic voting machines. Finally, we present some examples of the vulnerabilities we found in our experiments.

3.1 Security Testing Methodology

In this section, we present a five-step testing methodology that can help security engineers in designing experiments to evaluate the security of an electronic voting system. This is a high-level approach that focuses on finding bugs and design errors that can potentially be exploited by an attacker to violate the integrity, confidentiality, and availability of the voting process.

Step 1. Information gathering

The first step consists of collecting all the available information on the system under test and preparing the environment in which the testing will be performed. In particular, it is important to obtain the following resources:

- A copy of each of the machines that are part of the voting system. Even though some previous analyses of electronic voting systems [9, 18, 25] were based solely on the source code, the availability of the actual hardware greatly increases one's confidence in the results and allows the tester to actually implement and verify the effectiveness of each attack.
- A copy of both the source code and binaries for each software component installed on the voting machines. This is not strictly required in order to test the voting system, but it can help to reduce the amount of reverse engineering required and simplify the vulnerability analysis.
- A copy of all the available documentation (e.g., software user manuals, hardware schematics, and description of the voting procedures) and the results of past testing experiments (if any) performed by other teams on the same voting system. Many of these documents are publicly available on the Internet.
- Vendor support in terms of the training required to properly operate each hardware or software component. In addition, a step-by-step example of a complete election process can be very useful to quickly understand all the involved procedures and the interaction between the different components. This information can be extracted from the documentation or from the analysis of each module. However, the involvement of the vendor can simplify this task.

In our experiments (and therefore in the rest of the paper), we assume that the testers have full access to all of the aforementioned resources. It is important to note that even though this can greatly improve the quality of the testing, previous studies have shown that an attacker can successfully find exploitable vulnerabilities with very limited access to the hardware/software infrastructure.

Step 2. System analysis and identification of the information flow

The goal of this phase is to model the input/output interface of each hardware and software component.

First of all, it is important to inspect the hardware and list every input/output channel such as serial ports, memory card slots, or wireless interfaces. For example, even though a DRE is usually not equipped with a keyboard, opening its case can reveal an internal keyboard port that can be very useful for debugging and testing.

The best way to reconstruct the information flow between the different components is through a precise analysis of the source code. However, in order to avoid problems in the rest of the experiments, it is a good practice to initially verify that the source code obtained in the previous step corresponds to the actual software installed on the various machines. Unfortunately, the use of proprietary (or no longer available) build environments can complicate this operation, sometimes making a precise verification impossible.

During this phase, the testers must precisely identify which data is exchanged between the different components, what protocol and data format is used in the communication, and which physical medium carries the information (e.g., an Ethernet cable, a phone line, or a compact flash card).

In addition, it is important to understand how each component authenticates and validates the data it receives and how the information is protected from external analysis, eavesdropping, man-in-the-middle attacks, tampering, and replay attacks. For example, it may be possible for an attacker to use the same credentials to vote twice or to sniff the communication containing the voting results.

If the data is encrypted, it is important to understand the way in which the encryption key has been shared between the sender and the receiver. For example, if the key is transmitted on the same medium, the security of the communication can be easily compromised.

Finally, the analysis of the source code can reveal other valuable information, such as undocumented features or the presence of debug functionalities that can be exploited to subvert the voting system.

Step 3. Identification of threats and attack exposures

Once all the details of each component have been collected, the tester can draw a global picture representing the interaction and the information flow between the devices involved in the voting process. Before starting to look for vulnerabilities, it is important to add one last piece to the puzzle: the voting procedures. Voting procedures are a set of rules and best practices that regulate how a real election must be executed, describing, for example, who is in charge of each operation, who is going to operate the voting devices, and how the devices will be operated.

Taking into account the procedures is very important in designing realistic attack scenarios. However, it is also very important to remember that a procedure cannot be the only defense mechanism against an attacker. For example, if there is a button on the side of an electronic voting machine to reset the system, assuming that a poll worker can check during the election that nobody presses that button is not a solution to the problem. For this reason, it is important to also devise experiments to test the cases in which some of the procedural assumptions are violated, intentionally or not.

Finally, at this stage it is important to define a precise threat model, which is a model of the possible attackers, their motivations, capabilities, and goals. For instance, an attacker can be interested in deleting or altering the results, in preventing other people from voting, or in discovering the identity of previous voters. Categorizing the attackers is also very important. Given the critical tasks performed by these devices and the amount of money involved in a real election, insiders, as well as outsiders (e.g., regular voters), can be interested in attacking the system. Poll workers and election officials can be bribed, or they may have personal interests

in affecting the results of an election. Even malicious vendor employees must be taken into consideration, especially given their access to the low-level voting infrastructure.

To summarize, the system analysis describes how the voting system works, the operational procedures describe how humans are supposed to interact with the system, and the threat model describes the possible goals and resources of various attacker categories. Combining these three pieces of information allows the security engineers to identify the possible attack scenarios. For instance, if the component analysis reveals that the smart card tokens are not cryptographically signed and the procedures state that voters are given tokens to manually insert into the DRE, then it is possible to devise an attack scenario in which a user can switch the smart card provided by the poll worker with a malicious one, thus providing erroneous data to the DRE.

One can visualize the voting process as a chain of trust and information that links together all the machines and the people involved in the voting system. At the beginning, the election officials prepare the ballot definition. The definition is saved into some devices that are then used to initialize the electronic voting machines. At the end of the election, the votes stored in the machines are collected and sent back to the election management system to be tallied.

This process has a circular structure, where the input of a step in the process is the output of the previous step. Enumerating all the attack scenarios means enumerating all the possible ways in which an attacker can compromise a component involved in the process and break the circle.

Step 4. Breaking the circle: attacking a component of the voting process

The objective of this phase is twofold. First, the tester must perform a vulnerability analysis to identify any bug or flaw in the system design that can be exploited to realize one of the attack scenarios that has been identified in the previous step.

When a vulnerability is discovered in one of the components, the second step consists of developing an attack that successfully exploits the vulnerability. Compared with other security testing experiments, this task presents interesting and novel difficulties. First, due to the intrinsic characteristics of the voting environment (see Section 2.3 for more details), it is often necessary to develop a number of *ad hoc* tools in order to interact with the voting devices.

Second, the stealthiness of the attack can be a very important point. Even though a simple exploit that crashes a DRE can be an effective denial of service attack, more advanced attacks that aim at affecting the results of the election need to go unnoticed. This is particularly difficult, because most election systems are designed to identify and audit any error and suspicious condition; often, they rely on a Voter-Verified Paper Audit Trail (VVPAT), which can be very difficult to modify.

Examples of tools and techniques that can be used to circumvent these limitations are presented in Section 3.2.

Step 5. Closing the circle: compromising the entire voting system

In the previous step, the testers develop attacks that can be used to compromise a single component of the voting system. In this last phase, the focus shifts from the single component to the entire voting system. In particular, it is now important to evaluate how a compromised component can take advantage of the legitimate information flow to take control of other devices, with the goal of eventually controlling the entire voting infrastructure.

The idea is to use a combination of the attacks developed in the previous step to inject a virus-like malicious software that is programmed to automatically spread to as many voting machines as possible. This can be achieved by copying the virus onto media devices (DTDs) that are later inserted into other components where the malicious data can trigger a local vulnerability. If the virus can reach and infect election central (where components for all of the precincts are initialized and the votes are tallied), the entire voting process can be compromised.

3.2 Techniques and Tools

In this section, we describe some of the techniques and tools we used to apply our methodology. Because electronic voting systems are implemented using specialized hardware, custom tools are often needed. The type of tools required depends on the type of firmware the voting machine utilizes. Therefore, we first review the different firmware types and discuss how they influence the testing process.

Types of Voting Machine Firmware

The firmware of the voting machines we analyzed can be classified in three different types, based on the amount of COTS components they utilize. The first group of voting system firmware utilizes a COTS operating system and all voting-specific code is run as processes within the operating system. The second class of firmware utilizes a COTS BIOS. In this case, the voting system firmware includes functionality normally performed by the operating system, but utilizes the BIOS for most I/O operations and boot time initialization. The third class of voting system firmware does not rely on any third-party components. This type of voting system firmware runs completely standalone.

Depending on the class of firmware, the type of analysis tools needed for the evaluation differs. For systems utilizing a COTS operating system, the operating system tools and services can be leveraged to perform the analysis. For instance, most operating systems include tools to perform file operations (e.g., a command shell) that can be leveraged in order to replace the voting-system-specific code. In addition, many operating systems include functionality to support the debugging of user-level processes. This debugging functionality is very useful when crafting exploits. Finally, an OS provides process isolation; therefore, if an attack causes the voting system process to crash, the operating system would keep running and allow for uninterrupted debugging support.

Systems that rely on a COTS BIOS but do not run in an operating system require radically different analysis methodologies. This class of voting system firmware does not include all the services normally provided by an operating system. None of the systems we analyzed contained functionality for attaching a debugger to the voting application process or for manipulating files. Another challenge with these systems is that they have very limited I/O capabilities. A

common debugging technique is to create a debug program trace by printing to the console as the program execution progresses. This technique is complicated by the fact that none of the voting machines has a built-in console that could be printed to. In addition, the voting systems we analyzed of this type were designed so that if the process running the voting system software crashes, then the whole system halts. This complicated the process of doing a post-mortem analysis of failed exploits.

Voting systems that are completely standalone have all the challenges of OS-free voting systems, in addition to some specific challenges that the lack of a BIOS causes. One of the main tasks of the BIOS is to facilitate the boot process. The normal boot sequence of a system with a BIOS begins with the processor jumping to a specific address within the BIOS where it starts executing. The BIOS initializes some of the hardware and loads the boot block from the boot drive. The boot block in turn loads the operating system from disk. The operating system is often contained in a regular file on the boot file system. In a BIOS-free system, however, the boot process works differently. The processor starts executing at a specific address after a reset. Since there is no BIOS in the system, the voting system code must be located where the BIOS would be in a COTS system. This means that the voting system code cannot be stored in a file on a file system, but, instead, has to be stored on a ROM or EPROM chip. This fact complicates the analysis. In a BIOS-based system, it is easy to read and replace the voting system firmware since it is located in a regular file on a flash card and hardware adapters to access flash cards are readily available. The BIOS-free systems we analyzed all required specialized and less available hardware in order to read and write the EPROM chips. In addition, in one of the systems we analyzed, the EPROM was soldered on the board and it could not be removed without unsoldering it.

Tools

During the testing process, we implemented a number of tools in order to perform the required analysis. First, we developed tools for performing low-level reconnaissance and vulnerability analysis. These tools allowed us to, for instance, extract and replace firmware images, dynamically inspect and modify runtime state, and read or write DTDs. Next, additional tools to facilitate system exploitation were developed; an example presented is that of a firmware patching framework.

Firmware reader/writer. For the voting machines that had firmware stored on an EPROM chip, we needed a way to read and modify the contents of the chip. While commercial EPROM readers are readily available, they did not suit our needs. EPROM readers require the chip to be extracted from the circuit and inserted into the reader. The voting machines we analyzed had the chips soldered onto a circuit board, and removing them would have been cumbersome. Fortunately, the processors used in the voting machines with ROM chips all had built-in JTAG [12] support, which could be leveraged to access the EPROM chips. JTAG is a hardware debugging interface. It allows, among other things, the tester to completely bypass the processor logic and control the logic state of the processor's pins directly. By changing the logic state of the processor's pins in a carefully controlled pattern, the EPROM can be accessed through the JTAG port. Unfortunately, we could not find an affordable JTAG tool that supported the particular processor used by the voting device. We ended up extending an open source JTAG tool designed for ARM processors (OpenOCD [24]) to work with the voting machines' processors.

Debugger. One of the most important tools needed to write a functioning exploit is a debugger. The debugger allows the tester to inspect the memory contents of the voting machines, set breakpoints, and single step through the sections of code that contain vulnerabilities. Since none of the DREs we analyzed had built-in support for debugging, we had to add this functionality. We chose to implement GNU debugger (GDB) support over a serial line. This technique allows the tester to attach a debugging computer to the voting machine using a serial cable. The debugging computer runs the GDB application and provides the tester with full debugging support of the target voting machine. In order to implement this functionality, a debugging stub has to be installed on the voting machine.

We were not able to compile the firmware of any of the voting machines we tested because we were not provided with a functional build system. This forced us to binary-patch in the debugging stub. The debugging stub we used was based on a stub shipped with GDB. We modified the stub in order to make it self-contained, since we could not rely on operating system services to access the serial port. After compiling the stub, we copied the binary stub to an unused area of the voting machine's ROM. For the stub to work, it needs to be hooked into the voting machine's interrupt table. We located the interrupt table of the voting machine by disassembling the binary. The interrupt table can easily be found by looking for the assembly instruction used to load the interrupt table. After identifying the location of the interrupt table, we modified the table to point at our debugging stub. By performing these modifications, the code running on the voting machine could be debugged.

DTD reader/writer. Most voting machines we analyzed utilized some kind of data transport device (DTD) or hardware token for access control and voting machine initialization. The DTDs stored a sizeable amount of data that was read into the voting machine during the authentication and initialization processes. Since the voting machine was reading data from these tokens, they represented an interesting attack vector. In order to explore this vector, we developed tools to perform low-level reads and writes of the data contained in the DTDs. This allowed us to create tokens that contained illegal or unusual data.

Firmware patching framework. After identifying a vulnerability and creating a working exploit, the next step was to modify the firmware and cause it to behave in a malicious way. Since we were not able to compile the source code, we could not just modify the source and compile a malicious firmware version. Instead, we had to modify the firmware by binary-patching in the new functionality. Manually performing the binary-patching can be time consuming and error-prone. Therefore, we designed a patching framework that allowed us to write extensions to the firmware in C and link these extensions to the original firmware. Two types of linking were needed. First, the extensions needed to be able to call functions in the original firmware. Second, the extensions needed to be able to hook themselves into the original firmware so that the original firmware would

call the extensions at an arbitrary location. The framework consisted of two jump tables and a patching script. The two jump tables processed the two kinds of links and allowed the extension to call functions in the original firmware normally. The patching script concatenated the original firmware and the extensions and created a new binary. In addition, the patching script overwrote interesting function calls in the original firmware and diverted the function calls to the extension.

3.3 Findings

Security evaluations of both the Sequoia and ES&S voting systems resulted in the discovery of a number of previously-unknown vulnerabilities. In the following sections, we describe a representative sample of the vulnerabilities found and discuss several possible attack scenarios.

EMS vulnerabilities

Tests of both vendors' election management systems (EMS) revealed numerous flaws. Perhaps the most troubling finding was the presence of exploitable software defects allowing the execution of arbitrary code of an attacker's choosing. For instance, buffer overflows were present throughout the source code for ES&S' EMS, indicating a pervasive ignorance or dismissal of basic security awareness and defensive programming techniques. In one case, a working exploit was developed for a buffer overflow in ES&S' election results processing code that allows an attacker to gain full control of the EMS.

Another area of significant concern was the general lack or misuse of cryptographic techniques to authenticate users of the voting system or to ensure the integrity of critical election data. For instance, asymmetric cryptography was completely eschewed in favor of secret keys, which in many cases were hard-coded into a component's software with no apparent strategy for key revocation in the event of a compromise. Additionally, although election data was in some cases protected by a checksum, these were easily forged and, invariably, no cryptographically-strong signing mechanism was used. These oversights allow an attacker, for instance, to forge authentication tokens and election results, in some cases for entire precincts.

A third area in which vulnerabilities were found is that of incomplete specification of system requirements and misconfiguration of system environments. Both vendors support the option of deploying their EMS on customer-provided hardware; in this case, however, documentation relating to proper system configuration and security hardening is often misleading or incomplete, resulting in potentially serious vulnerabilities. For instance, Sequoia's EMS was configured with the Windows "autorun" feature enabled for removable media, allowing an attacker to compromise the machine via the simple insertion of a flash drive or CD-ROM. The EMS also allowed remote users to not only connect to its back-end database as the database administrator, but also allowed remote users to execute arbitrary commands using database extensions that could have been easily disabled. Finally, ES&S' EMS shipped with a version of the Java Runtime Environment that includes a known vulnerability in its image processing code. Clearly, a coherent, pellucidly articulated set of configuration procedures and system requirements would largely mitigate such vulnerabilities.

DRE vulnerabilities

In our evaluations, the vendors' respective DRE products suffered from classes of vulnerabilities similar to those found in the election management systems. Both DREs contained multiple buffer overflows in their handling of election data, and working exploits were developed for overflows in ballot-loading code for each machine. Generally speaking, each of the vendors' source code bases were written without regard to modern security engineering practices, such as avoiding the usage of unsafe string handling functions or performing rigorous input validation checks.

The design of both DREs also exhibited the same ignorance or misapplication of cryptography as in the case of the EMS, with similar implications. During our evaluations, it was trivial for an attacker to forge authentication tokens as well as modify or simply create election data. This critical lack of cryptographic protection allows a malicious person to impersonate an election official or vendor technician, vote multiple times, perform unauthorized reconfiguration, or introduce exploits into the system.

A particularly disquieting finding was the presence, in both products, of backdoors or expressly-prohibited features in the source code. In the case of Sequoia's DRE, its firmware contained a full-fledged interpreter for a scripting language allowing a user to set the "tamper-proof" protective counter of the machine, set the machine's serial number, overwrite arbitrary files (including election data, the firmware, or audit log) on the internal compact flash drive, and reboot the machine.[1] The source code for ES&S' DRE likewise recognized special "initialization" and "factory" authentication tokens that allow one to, for instance, reset or bypass system passwords and erase election and audit data.

Finally, contrary to the claims of the vendors, the physical seals protecting access to critical components of the DRE were, in almost all cases, not tamper-proof or even tamper-evident. In many cases, the seals could be removed without evidence or bypassed altogether by simply removing a small number of screws and disassembling the chassis of the DRE. The lack of physical security allows an attacker to access sensitive poll worker controls and I/O ports during an election, or to directly access the system firmware, election data, and audit logs.

Optical scanner vulnerabilities

Evaluations of the various optical scanners offered by both vendors followed much the same pattern of the previous voting system components. A patent disregard for cryptographic authentication and integrity checks allows attackers to overwrite a system's firmware with malicious versions and modify or construct election data to be processed by an EMS. Physical security measures were also lacking. In particular, the ES&S scanner lock was easily picked with a paper clip during our tests, while the "unpickable" lock on the Sequoia scanner was bypassed by removing a few screws and pulling out the lock cylinder from the scanner's chassis by hand. In both cases, this allows an attacker to access machine internals to potentially execute arbitrary code.

[1] "*Self-modifying, dynamically loaded or interpreted code is prohibited, except under the security provisions outlined in section 7.4.*" [30, Sec. 5.2.2]

Attack scenarios

The vulnerabilities that pervade each vendor's voting system allow a multitude of serious attacks to be executed under several threat models. Taken in isolation, these vulnerabilities constitute a sobering threat to the successful execution of a fair election. Clearly, the ability to run arbitrary code on the election management system of a county, or directly modify election data from a high-speed optical scanner that processes tens of thousands or more votes during a single election, is a cause for alarm. When these vulnerabilities are considered in the context of the system as a whole, however, even more alarming attack scenarios present themselves. To illustrate this point, we discuss a class of attacks that was successfully demonstrated on both vendors' voting system: a voting system virus.

Sequoia virus attack. In a Sequoia voting system deployment, an attacker first obtains access to a county elections office.[2] The attacker surreptitiously drops a maliciously crafted USB flash drive into the pool of drives used to initialize authentication token programmers. When this drive is inserted into the computer hosting the EMS, Windows autorun automatically executes a Trojan hidden on the drive that contains the virus.

The virus silently installs itself and begins monitoring the host for removable media insertion and removal events. Any flash drive inserted into the EMS is infected with a copy of the virus. In addition, results cartridges are modified to contain an exploit for a buffer overflow in the DRE's ballot-loading code as well as a copy of the virus.

Infected results cartridges are subsequently used to initialize DREs prior to the election. The exploit silently executes during ballot loading and installs a malicious firmware on the DRE. The malicious firmware acts normally during pre-election logic and accuracy testing by taking advantage of existing variables that indicate whether the DRE is being tested.

On election day, the malicious firmware begins to execute various vote stealing attacks. Examples of these attacks include:

- Modifying the ballot such that the favored candidate is voted for even if not selected.
- Modifying uncompleted ballots when the voter has fled[3].
- Printing a ballot summary and indicating that voting is complete, waiting until the voter has left, voiding the requested selections, and then casting a modified version.
- Simply casting a modified ballot that disagrees with the paper audit trail.

Note that in some of these attacks the VVPAT will be consistent with the number of votes recorded internally by the DRE. Therefore, they will pass consistency checks. In addition, attacks that insert additional votes (in both the VVPAT trail and the memory of the DREs) are still feasible because, in most cases, the election officials cannot determine which votes have been forged and they will have to either accept all the votes or throw away the election altogether.

ES&S virus attack. In an ES&S voting system deployment, an attacker with access to a DRE loads a malicious firmware containing the virus into the machine either by exploiting a vulnerability or by directly modifying the on-board flash memory. When a master DTD is inserted into the DRE to initialize it for the election, the malicious firmware installs a copy of the virus on the DTD itself. Subsequent uses of the DTD to initialize other DREs result in those machines being infected through a ballot-loading exploit.

During pre-election logic and accuracy tests, the firmware behaves as expected. During the election, however, the malicious firmware carries out vote stealing attacks similar to those described in the previous scenario.

After the election has ended, a master DTD is used to collect the votes from each DRE. During this operation, the malicious firmware infects the DTD with a copy of the virus, if it is not already infected. The DTD is then transported by an elections official to the county elections office, where the votes are transferred into the EMS. During this operation, a vulnerability in the EMS is exploited such that the virus is installed in the EMS, allowing the possibility of further attacks against the election.

After the tallying and reporting process has completed, the virus remains dormant on the EMS host until the next election. At this time, the virus will infect the master DTD that is programmed to initialize the DREs for that jurisdiction, and the cycle will continue.

Discussion

In both of these scenarios, analysis of the information flow of the voting system along with the capabilities of each individual attack discovered allowed for the identification of large-scale threats against the voting systems that would not have been recognized in a piecewise analysis. Virus-style attacks clearly pose a greater threat to fair elections than attacks against single components, since they are persistent between elections, and a greater number of votes can be affected.

Additionally, the wide variety and large number of vulnerabilities discovered resulted in many vectors for the introduction of such a virus. For instance, in the case of the Sequoia voting system, the virus could be introduced by exploiting a remote vulnerability in the back-end database of the EMS. Similarly, for ES&S, the virus could be introduced into the system by exploiting the EMS during the tallying process for the preceding election.

Finally, we want to stress that these scenarios have been implemented and tested against real, certified voting systems that are in use today. Far from being the purview of the mythical uber-hacker, our findings indicate that large-scale exploitation of electronic voting systems is well within the capabilities of "persons having ordinary skill in the art."

4. LESSONS LEARNED

As we have shown above, the electronic voting systems that we have reviewed are neither secure nor well-designed. In this section, we summarize what we found to be the major pitfalls of both systems.

Poor integration leads to insecurity. One of the problems with most of the electronic voting systems that are being used today is that these systems are put together by

[2]Note that if the attacker is an insider, such as an elections official, they already have access to the election office.

[3]A "fleeing voter" is a voter that has only partially completed the voting process and has left the voting station.

integrating election components created by different companies/groups. As a consequence, often there is no overall system design and no coherent structure. In fact, one of the reviewed systems was a mishmash of legacy software. The EMS was written using at least four different programming languages. To make matters worse, almost every module was using its own database, often storing duplicate data, and had its own authentication system, if any. While each of the languages used unquestionably has its own benefits and can be the more appropriate choice in a particular situation, the use of a variety of languages in a single system component complicates its analysis and creates a larger attack space. In fact, a thorough analysis of such a system, or even of its data flow, is close to impossible. *If reuse of a piece of code is proved to be necessary or helpful, the whole-system design should be taken into account.*

Cryptography is hard to get right. One of the major areas of concern was the use of cryptography. In both systems we found that most uses of cryptographic techniques could be classified into three main categories: naive use, wrong use, or no use at all. For example, in one of the analyzed systems, data on a DTD was protected via encryption using a strong symmetric block cipher algorithm, but the encryption key was stored in the clear on the same media. In other cases, when election data was protected by a checksum, the checksum could be changed to match the maliciously modified data. Even worse, in both systems, no cryptographically-strong signing mechanisms were used to protect the integrity of sensitive data. *A mindful usage of strong encryption algorithms with strong well-protected keys along with data signing are a must for building secure voting systems.*

Unfounded trust assumptions enable compromise. Another major problem with both reviewed systems was a lack of mechanisms allowing one to check the origin of data along with a lack of appropriate input validation. In fact, most of the components that we reviewed assumed that input data came from a specific system component, disregarding the fact that in many cases it could easily be forged. For example, checksums were often taken as proof of data origin. Also, data that was expected to come from other components (for example, data on a DTD that was supposed to be generated by an EMS) was often unchecked for boundary cases. Many such cases resulted in exploitable vulnerabilities. It is well-known in the security community that the lack of input validation is one of the major premises for the existence of vulnerabilities. *One of the main premises for building a secure voting system is the absence of any unfounded assumptions and mindful checks of all inputs.*

Certification and standards that are currently used are not enough for security. Both of the systems analyzed were certified, and their source code was officially compliant with at least one of the standards in use today. Nevertheless, both systems were inherently insecure. The problem is that currently used source code standards are not security-oriented, and even if they were, a simple checklist-based verification would not be enough. For instance, to prevent buffer overflows, a standard could require that any use of a function writing data to a buffer should be preceded with a boundary check of input size against the size of the destination buffer. In this case, while the standard would enforce the usage of checks before each case, it would still fail to guarantee that the checks were correct. In fact, one of the exploitable buffer overflows that we found was a result of a mistake in a similar check. Also, during the review, we found that systems are not as compliant with standards as they claim to be. *A more thorough and security-oriented certification process for evaluating voting systems is needed.*

Logic and accuracy testing gives a false sense of security. One of the selling points of both systems was the fact that they provide a built-in way of testing their systems for accuracy, which can be done right before an election. In practice, from a security perspective we found the tests to be completely useless, since testing was done only while in a special testing mode, which was enabled through a switch in the system's firmware. Clearly, since the system itself is aware of the testing mode, any malicious code that was implanted into the firmware could easily pass the accuracy tests. Interestingly enough, one of the vendors seemed to have a strong belief that their logic and accuracy test is capable of identifying malicious code. *The only way to make logic and accuracy tests realistic is to, at the very least, have the firmware totally unaware of any testing mode.*

COTS components are difficult to configure in a secure way. We found that the use of COTS components in some cases made the voting systems more vulnerable. The main problem is that COTS components often come with a lot of functionality and can be hard to configure in a secure way. For example, the EMS for both systems was based on the Windows operating system, which is a very complex system of its own, with a large number of pre-configured settings. Adequate hardening of such a system requires a high level of expertise. Nevertheless, the systems that were given to us came mostly with default configurations and no specification on how to configure each system's security was given in either case. The "autorun" vulnerability presented in Section 3.3 is one example of this problem. *When COTS components are used, vendors should either provide a detailed specification of how the systems should be configured or provide pre-configured systems.*

Voting procedures underestimate the power of potential adversaries. Another common problem that we found is that the security and integrity of both systems often depend on poll workers following an explicit set of procedures. In fact, we found that the physical security of most components depended more on compliance with a set of procedures than on strong physical guards. Often the seals that were used to protect critical system components could be easily bypassed. Interestingly, vendors seem to fail to realize that procedures do not substitute for built-in system security and can be easily violated, intentionally or not. In fact, one of the vendors' rebuttals to a discovered security problem was that the problem cannot occur because it violates the procedures. *Procedures should never be relied upon as the only guarantee of system security; rather, each component of a system should implement a complete set of security mechanisms necessary for its protection.*

Security training of developers is not sufficient. One of the most frustrating discoveries that we made is the apparent lack of adequate security training of the voting system developers. For example, it was often the case that code written in C consistently used the infamous *strcpy()* function without checking the size of the copied data against the size of the destination buffer, which is one of the most

common causes of buffer overflows. Even more surprisingly, we found cases when the more secure *strncpy()* function was used incorrectly; the size of the input was checked against itself rather than against the destination buffer. *Knowledge of basic security concepts, their application, and defensive programming practices should be prerequisites for the developers of critical systems such as an electronic voting system.*

To summarize, we found that, in both systems, security was not a part of the design and often security features were added to the systems in an *ad hoc* way. In our opinion, the "security through obscurity" principle was often used as one of the main protection mechanisms. While undoubtedly the proprietary nature of the voting software makes it harder for an attacker to develop a working exploit for the system, we have shown that it does not make a system completely secure. Given sufficient time and determination, an attacker can successfully reverse-engineer a system, starting with very little information. In fact, the only way to make electronic voting systems secure is to build security in from the very beginning of the development process.

5. RELATED WORK

The first analysis of a major electronic voting system was performed in 2003, when Bev Harris discovered that the source code repository of the Diebold system, precompiled binaries, and other documents were stored on a publicly accessible FTP server [9]. After downloading the files and testing the system, in particular the EMS, she discovered various ways to change votes, bypass passwords, and alter audit logs [10].

The same Diebold repository was also analyzed by a team of researchers from Johns Hopkins and Rice Universities [18, 25]. Their analysis, which focused on the source code of the voting terminal, found serious and wide-reaching problems and concluded that the system was "far below even the most minimal security standards applicable in other contexts." Subsequent studies taking into account the actual hardware, election procedures, and environment confirmed the main results of the Hopkins-Rice study [27, 33].

The first independent security assessment of a complete voting system, including both the hardware and the software components, is due to Feldman et al. [5]. The Princeton team obtained the system from an undisclosed "private party," proved that malicious code could be easily installed on the machine to undetectably steal votes and modify records and audit logs, and developed a virus with the capability of spreading from one machine to another. They also uncovered several problems with the physical security of the terminal [6].

More recently, as access to electronic voting equipment has increased either through officially sponsored initiatives or unofficial channels [1], assessments of their security features have also become more common. Available studies demonstrate that problems are present in most systems, independent of the specific system's vendor. For example, Gonggrijp and Hengeveld reported a number of weaknesses in the Nedap ES3B system, used in Holland [8]; Proebstel et al. discovered several exploitable issues in Hart's eSlate system [22]; Ryan and Hoke discussed problems with Diebold's GEMS software [26]; Yasinsac et al. discovered several software vulnerabilities in ES&S' iVotronic [34].

Furthermore, problems are not limited to one category of electronic voting technology. While most of the studies have been directed at DRE systems, optical scanners are not free of security-relevant defects [11, 16, 17].

Internet-based voting systems have also received great scrutiny and showed similarly severe security issues. For example, the SERVE system was designed to allow US overseas voters and military personnel to vote in the 2004 primary and general elections, but was canceled after a study uncovered a number of critical flaws [13, 14].

Finally, the flaws discovered through testing and other analysis techniques in current voting systems have stimulated a number of research efforts to improve the design of voting machines [28, 35], to ameliorate security primitives, such as the storage of votes [3, 20] and auditing [7], or to propose novel voting protocols with desirable characteristics, such as verifiable voting in absence of trusted components [15] and privacy against adversaries with unbounded computational resources [21].

This paper differs from the aforementioned studies in several significant aspects. With respect to earlier efforts, the reviews in which we participated had a much broader scope: we had (almost) full access to source code, documentation, actual voting machines, and procedure descriptions used in real elections. We also had the opportunity to test complete voting systems rather than single pieces of equipment. These factors allowed us to test the security of the system more thoroughly (e.g., confirming vulnerabilities detected through source code analysis by developing actual, working exploits) and to assess the security implication of combining several vulnerable components (e.g., showing how a virus could spread from the EMS to the DRE components).

The systems we tested were used in contexts where the act of casting a vote and the transmission of ballots over a network (e.g., the Internet) was prohibited by law. Therefore, we did not have a chance to explore problems arising in this situation. However, some of the components we reviewed could be interconnected by an internal network, physically separated and disconnected from external networks. We investigated this scenario and documented the corresponding threats.

Finally, the contributions of this paper do not consist of a novel voting technique or system. Instead, this paper provides a comprehensive review of the methodology, techniques, and tools we developed and used in our testing experiences, and a summary of the lessons learned during our studies.

6. CONCLUSIONS

Electronic voting systems have been proposed to make the voting process faster and more reliable. Unfortunately, all voting systems recently analyzed by independent security testers have been found to contain fatal security flaws that could compromise the confidentiality, integrity, and availability of the voting process.

In this paper, we presented our experience with the security testing of real-world electronic voting systems. We presented the methodology we used, the tools we developed, and the lessons we learned.

Our experience suggests that there is a need for a drastic change in the way in which electronic systems are designed,

developed, and tested. Researchers, practitioners, and policy makers need to define novel testing approaches that take into account the peculiar information flow of these systems, as well as the combination of computer security mechanisms and physical procedures necessary to provide a high level of assurance.

Unless electronic voting systems are held up to standards that are commensurate with the criticality of the tasks they have to perform, the very core of our democracy is in danger.

Acknowledgments

We want to thank Matt Bishop at UC Davis and David Wagner at UC Berkeley, for having organized the evaluation of electronic voting systems in California. Also, we want to thank Patrick McDaniel at Pennsylvania State University who led the security testing project in Ohio, and Matt Blaze at University of Pennsylvania, who worked side-by-side with us in evaluating systems both in California and Ohio.

References

[1] A. Appel. How I bought used voting machines on the Internet. `http://www.cs.princeton.edu/~appel/avc/`, February 2007.

[2] E. Barr, M. Bishop, and M. Gondree. Fixing Federal E-Voting Standards. *Communications of the ACM*, 50(3):19–24, March 2007.

[3] J. Bethencourt, D. Boneh, and B. Waters. Cryptographic Methods for Storing Ballots on a Voting Machine. In *Proc. of the Network and Distributed System Security Symposium*, 2007.

[4] S. Everett. *The Usability of Electronic Voting Machines and How Votes Can Be Changed Without Detection*. PhD thesis, Rice University, 2007.

[5] A. Feldman, J. Halderman, and E. Felten. Security Analysis of the Diebold AccuVote-TS Voting Machine. In *Proc. of the USENIX/ACCURATE Electronic Voting Technology Workshop*, 2007.

[6] E. Felten. "Hotel Minibar" Keys Open Diebold Voting Machines. `http://www.freedom-to-tinker.com/?p=1064`.

[7] S. Garera and A. Rubin. An Independent Audit Framework for Software Dependent Voting Systems. In *Proc. of the ACM Conference on Computer and Communications Security*, pages 256–265, 2007.

[8] R. Gonggrijp and W. Hengeveld. Studying the Nedap/Groenendaal ES3B Voting Computer: A Computer Security Perspective. In *Proc. of the USENIX/ACCURATE Electronic Voting Technology Workshop*, 2007.

[9] B. Harris. *Black Box Voting: Ballot Tampering in the 21st Century*. Elon House/Plan Nine, 2003.

[10] B. Harris. Inside a U.S. Vote Counting Program. `http://www.scoop.co.nz/stories/HL0307/S00065.htm`, July 2003.

[11] H. Hursti. Critical Security Issues with Diebold Optical Scan Design. Technical report, Black Box Voting Project, July 2005.

[12] Institute of Electrical and Electronics Engineers. *IEEE Std 1149.1-1990 IEEE Standard Test Access Port and Boundary-Scan Architecture*. IEEE, 1990.

[13] D. Jefferson, A. Rubin, B. Simons, and D. Wagner. A Security Analysis of the Secure Electronic Registration and Voting Experiment (SERVE). Technical report, US Department of Defense, 2004.

[14] D. Jefferson, A. Rubin, B. Simons, and D. Wagner. Analyzing Internet Voting Security. *Communications of the ACM*, 47(10):59–64, 2004.

[15] C. Karlof, N. Sastry, and D. Wagner. Cryptographic Voting Protocols: A Systems Perspective. In *Proc. of the USENIX Security Symposium*, pages 33–50, 2005.

[16] A. Kiayias, L. Michel, A. Russell, N. Shashidhar, and A. See. Tampering with Special Purpose Trusted Computing Devices: A Case Study in Optical Scan E-Voting. In *Proc. of the Annual Computer Security Applications Conference*, 2007.

[17] A. Kiayias, L. Michel, A. Russell, N. Shashidhar, A. See, and A. Shvartsman. An Authentication and Ballot Layout Attack against an Optical Scan Voting Terminal. In *Proc. of the USENIX/ACCURATE Electronic Voting Technology Workshop*, 2007.

[18] T. Kohno, A. Stubblefield, A. Rubin, and D. Wallach. Analysis of an Electronic Voting System. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 27–40, 2004.

[19] P. McDaniel, M. Blaze, and G. Vigna. EVEREST: Evaluation and Validation of Election-Related Equipment, Standards and Testing. Ohio Secretary of State's EVEREST Project Report, December 2007.

[20] D. Molnar, T. Kohno, N. Sastry, and D. Wagner. Tamper-Evident, History-Independent, Subliminal-Free Data Structures on PROM Storage-or-How to Store Ballots on a Voting Machine (Extended Abstract). In *Proc. of the IEEE Symposium on Security and Privacy*, pages 365–370, 2006.

[21] T. Moran and M. Naor. Split-Ballot Voting: Everlasting Privacy With Distributed Trust. In *Proc. of the ACM Conference on Computer and Communications Security*, pages 246–255, 2007.

[22] E. Proebstel, S. Riddle, F. Hsu, J. Cummins, F. Oakley, T. Stanionis, and M. Bishop. An Analysis of the Hart Intercivic DAU eSlate. In *Proc. of the USENIX/ACCURATE Electronic Voting Technology Workshop*, 2007.

[23] S. Pynchon and K. Garber. Sarasota's Vanished Votes: An Investigation into the Cause of Uncounted Votes in the 2006 Congressional District 13 Race in Sarasota County, Florida. Florida Fair Elections Center Report, January 2008.

[24] D. Rath. Open On-Chip Debugger. `http://openocd.berlios.de/web`, 2008.

[25] A. Rubin. *Brave New Ballot*. Broadway, 2006.

[26] T. Ryan and C. Hoke. GEMS Tabulation Database Design Issues in Relation to Voting Systems Certification Standards. In *Proc. of the USENIX/ACCURATE Electronic Voting Technology Workshop*, 2007.

[27] SAIC. Risk Assessment Report: Diebold AccuVote-TS Voting System and Processes. Technical report, Science Applications International Corporation, 2003.

[28] N. Sastry and D. Wagner. Designing Voting Machines for Verification. In *Proc. of the USENIX Security Symposium*, pages 321–336, 2006.

[29] T. Selker and S. Cohen. An Active Approach to Voting Verification. Technical Report 28, Caltech/MIT Voting Technology Project, May 2005.

[30] United States Election Assistance Commission. Voluntary Voting System Guidelines. `http://www.eac.gov/votingsystems/voluntary-voting-guidelines/2005-vvsg`, 2005.

[31] G. Vigna, R. Kemmerer, D. Balzarotti, G. Banks, M. Cova, V. Felmetsger, W. Robertson, and F. Valeur. Security Evaluation of the Sequoia Voting System. Top-To-Bottom Review of the California Voting Machines, July 2007.

[32] D. Wagner. Testimony before U.S. House of Representatives at joint hearing of the Committee on Science and Committee on House Administration. `http://www.cs.berkeley.edu/~daw/papers/testimony-house06.pdf`, 2006.

[33] M. Wertheimer. Trusted Agent Report: Diebold AccuVote-TS Voting System. Technical report, RABA Technologies, LLC, 2004.

[34] A. Yasinsac, D. Wagner, M. Bishop, T. Baker, B. de Medeiros, G. Tyson, M. Shamos, and M. Burmester. Software Review and Security Analysis of the ES&S iVotronic 8.0.1.2 Voting Machine Firmware. Technical report, Security and Assurance in Information Technology Laboratory, Florida State University, Tallahassee, FL, 2007.

[35] K.-P. Yee. *Building Reliable Voting Machine Software*. PhD thesis, University of California, Berkeley, 2007.