Source Code Review of the Sequoia Voting System[1]

Matt Blaze[2]
University of Pennsylvania[3]

Arel Cordero
University of California, Berkeley

Sophie Engle
University of California, Davis

Chris Karlof
University of California, Berkeley

Naveen Sastry
University of California, Berkeley

Micah Sherr
University of Pennsylvania

Till Stegers
University of California, Davis

Ka-Ping Yee
University of California, Berkeley

July 20, 2007

[1]This report was prepared at the University of California, Berkeley under contract to the California Secretary of State as part of a "Top-to-Bottom" review of electronic voting systems certified for use in the State of California.

[2]Team leader.

[3]All author affiliations are for identification only.

Executive Summary

This report was prepared at the University of California, Berkeley at the request of the California Secretary of State, as part of a "top-to-bottom" review of the state's electronic voting systems. This document is the final report of the team that examined the Sequoia voting system source code.

We found significant security weaknesses throughout the Sequoia system. The nature of these weaknesses raises serious questions as to whether the Sequoia software can be relied upon to protect the integrity of elections. Every software mechanism for transmitting election results and every software mechanism for updating software lacks reliable measures to detect or prevent tampering. We detail these weaknesses, and their implications, in Chapters 3 and 4.

In certain cases, audit mechanisms may be able to detect and recover from some attacks, depending on county-specific procedures; other attacks may be more difficult to detect after-the-fact even with very rigorous audits.

There were numerous programming, logic, and architectural errors present in the software we reviewed. Some of these errors may be relatively harmless and reflect the large size and heterogeneous nature of the codebase. But other errors we found clearly have serious security implications. Many of the most significant vulnerabilities we found — those likely to be especially useful to an attacker seeking to alter election results — arise from four pervasive structural weaknesses, discussed in detail in Chapter 3:

- **Data Integrity.** The Sequoia system lacks effective safeguards against corrupted or malicious data injected onto removable media, especially for devices entrusted to poll workers and other temporary staff with limited authority. This lack of input validation has potentially serious consequences, including:
  - Precinct election results stored on DRE Results Cartridges and optical scan memory packs are not effectively protected against tampering. A poll worker with physical access to a Results Cartridge or MemoryPack before results are counted (e. g. when returning results to the county elections board) can change recorded votes, and, in some cases, can introduce spurious results for other precincts. Under some conditions, a corrupted Results Cartridge may be able to cause damage to the WinEDS system itself when it is loaded for vote counting.
  - The safeguards against introduction of corrupt firmware into the precinct voting hardware are largely ineffective. An individual with even brief access to polling station hardware can tamper with installed firmware in a way that causes votes and paper trails to be recorded incorrectly, security logs to be corrupted, or ballots to be presented to voters incorrectly. Under some configurations and conditions, corrupt firmware may be able to be spread virally from compromised hardware and may persist across more than one election.
- **Cryptography.** Many of the security features of the Sequoia system, particularly those that protect the integrity of precinct results, employ cryptography. Unfortunately, in every case we examined the cryptography is easily circumvented. Many cryptographic functions are implemented incorrectly, based on weak algorithms with known flaws, or used in an ineffective or insecure manner. Of particular concern is the fact that virtually

all cryptographic key material is permanently hardcoded in the system (and is apparently identical in all Sequoia hardware shipped to different jurisdictions). This means that an individual who gains temporary access to similar hardware (inside California or elsewhere) can extract and obtain the secret cryptographic keys that protect elections in every California county that uses the system.

- **Access Control.** The access control and other computer security mechanisms that protect against unauthorized use of central vote counting computers and polling place equipment are easily circumvented. In particular, the security features and audit logs in the WinEDS back-end system (used for ballot preparation, voting machine configuration, absentee ballot processing, and post-election vote counting) are largely ineffective against tampering by insider attackers who gain access to WinEDS computers or to the network to which the WinEDS computers are attached.

- **Software Engineering.** The software suffers from numerous programming errors, many of which have a high potential to introduce or exacerbate security weaknesses. These include buffer overflows, format string vulnerabilities, and type mismatch errors. In general, the software does not reflect defensive software engineering practices normally associated with high-assurance critical systems. There are many instances of poor or absent error and exception handling, and several cases where the software behavior does not match the comments and documentation. Some of these problems lead to potentially exploitable vulnerabilities that we identified, but even where there may not be an obvious vulnerability identified, the presence of such errors reduces our overall confidence in the soundness of the system as a whole.

Aspects of some of the weaknesses we discovered appear to have been reported in prior studies, most prominently in the 2006 Alameda County report[1], while others appear not to have been discovered (or publicly disclosed) previously.

Whether the vulnerabilities reported here represent practical threats, and whether their exploitation can be prevented or detected, depends heavily on the physical controls and procedures used in individual counties. We did not examine county-specific controls and procedures in our analysis. However, we caution that effective mitigation of many of the vulnerabilities discussed in this report will, at the very minimum, place considerable additional pressure on physical security features (such as locks and seals) and human procedures (such as two-person control by poll workers). Many of the physical security features and procedures used in the Sequoia system appear to have been engineered under the assumption that the underlying software is considerably more secure than it actually is, and thus may not provide sufficient protection in light of the vulnerabilities discussed here.

[1]C. Humphreys and C. Merchant. Pacific Design Engineering. *Sequoia Voting Systems Vulnerability Assessment and Practical Countermeasure Development for Alameda County.* October 4, 2006.

Table of Contents

CHAPTER 1

Introduction

This report was prepared at the University of California, Berkeley at the request of the California Secretary of State, as part of a "top-to-bottom" review of the state's electronic voting systems. This document is the final report of the team that examined the Sequoia voting system source code.

The Sequoia system source code review team was located at UC Berkeley and consisted of the eight authors of this report: Matt Blaze (team leader), Arel Cordero, Sophie Engle, Chris Karlof, Naveen Sastry, Micah Sherr, Till Stegers and Ka-Ping Yee. Contributions to the review were also made by members of the Palo Alto-based source code review team during the first weeks of the project: Seenu Inguva, Eric Rescorla (team leader), Hovav Shacham and Dan Wallach. We frequently consulted with the UC Berkeley-based document review team[1] and the UC Santa Barbara-based "red team"[2]. All opinions expressed in this report, however, are solely those of the authors.

We started work on May 31, 2007 and received the Sequoia system source code on June 7, 2007. Work ended on July 20, 2007 with the delivery of this report, at which time we destroyed all proprietary materials that we received.

1.1 Scope

The Sequoia software we reviewed is part of a comprehensive system that includes touchscreen Direct Recording Electronic (DRE) voting machines and optical scan ballot collection equipment for use at polling places and election definition, management and counting software and hardware for use at a county elections headquarters. The specific system components certified for use in California for which we were asked to review source code were:

- WinEDS version 3.1.012
- Optech 400-C/WinETP version 1.12.4
- MemoryPack Reader[3] (MPR), firmware version 2.15
- AVC Edge Model I, firmware version 5.0.24
- AVC Edge Model II, firmware version 5.0.24
- VeriVote Printer (No version specified in contract; we received firmware version 4.3.)
- Card Activator, firmware version 5.0.21
- HAAT Model 50, firmware version 1.0.69L
- Optech Insight, APX version K2.10, HPX version K1.42
- Optech Insight Plus, APX version K2.10, HPX version K1.42

[1]Aaron Burstein, Nathan Good, Joe Hall and Deirdre Mulligan (team leader)

[2]Davide Balzarotti, Greg Banks, Marco Cova, Viktoria Felmetsger, Richard Kemmerer (team co-leader), William Robertson, Fredrik Valeur and Giovanni Vigna (team co-leader)

[3]Called a "MemoryPack Receiver" in Sequoia documents.

The centralized back-end processing functions (ballot preparation, voting machine configuration, and post-election vote counting) are performed by the *WinEDS* system, a client-server database that runs on a local network of desktop workstations running Microsoft Windows. If optical scan ballots are used, the WinEDS system is augmented with one or more *MemoryPack Receivers* and may also employ one or more *Optech 400-C* batch ballot scanners for absentee and recount ballot processing.

Precinct polling stations can be equipped with touchscreen DRE terminals, optical scan ballot readers, or both. Two models of DRE terminals are used: the *AVC Edge I* and the *AVC Edge II.* The Edge DREs are equipped with *VeriVote printers* that record "voter verified" paper record of each vote cast. Precinct polling stations that use Edge DREs can use a smartcard-based voter authorization system, which requires a *HAAT model 50* or a *Card Activator* at each precinct for use by poll workers and a supply of reusable *voter cards*. Edge configuration data, as well as ballot records, are stored on PCMCIA *Results Cartridges*, which are returned to the county election office after the polls close for vote tallying.

Optical-scan votes are accepted at polling stations by one or more *Optech Insight* or *Optech Insight Plus* ballot readers. Insight configuration data and precinct vote totals are stored on *MemoryPacks*, which are returned to the county election office after the polls close for vote tallying.

For brevity, we will refer here to the collective system as the *Sequoia System*, the back-end system as *WinEDS*, the touchscreen DRE terminals (of either model) as *Edges*, and the precinct optical scan readers (of either model) as *Insights*. When features specific to a particular device model are discussed, it will be unambiguously noted.

A detailed description of the system, its components, and the flow of data before, during and after an election can be found in Chapter 2.

The reviewed software comprises over 800,000 lines of source code written in a variety of programming languages, including C, C#, C++, Z80 assembly, 8051 assembly SQL, PowerBuilder, and Visual Basic. The system runs on several hardware and operating system platforms, including proprietary embedded controllers as well as general-purpose desktop computers running various versions of the Microsoft Windows operating system.

We took special precautions to protect the proprietary Sequoia source code that we analyzed. All source code review work was performed in a secure room (protected by a monitored alarm system, secured with special locks not on the building master key system, and equipped with a safe) on the UC Berkeley campus. Source code was kept on external drives that were stored in the safe whenever no reviewers were present in the room. All reviewing of source code was performed on a local network of computers (supplied by the California Secretary of State) contained entirely within the secure room (with care taken to avoid cross-connection with the external Internet or any local Intranet). Unauthorized persons not part of the project were not permitted in the room at any time once source review work began (including building maintenance and janitorial staff).

1.2 Methodology

Discovery of programming errors is a notoriously difficult problem in computer science, and no general methodology exists that is guaranteed to find all problems in even very small programs. The large size and complexity of the Sequoia system makes a complete review an especially daunting task under even the best conditions, but particularly so here given the limited time available and other constraints imposed on us by the terms of the review.

Our review did not attempt a line-by-line analysis of the source code. Instead, our analysis was largely *ad hoc*, with reviewers examining those architectural features and sections of code likely to have the most impact on security and reliability. Such an analysis is by its nature incomplete, and is particularly unlikely to discover deliberately introduced and obfuscated flaws, such as hidden "back doors" incorporated into the software by a malicious programmer.

Our focus was on whether the software contains effective safeguards against error and abuse, especially altering election results, changing votes, denying service, altering audit logs, and/or compromising voters' ballot secrecy. More broadly, we explored issues related to our confidence in the security and reliability of the architecture and implementation as a whole. In general, in our review we attempted to explore questions of architectural soundness:

- Does the design and implementation follow sound, generally accepted engineering practices? Is code defensively written against bad data, errors in other modules, changes in environment, and so on? Is the implementation designed to be maintained as bugs are found or requirements change?
- Is the cryptography and key management sound? Is cryptography correctly used to protect sensitive data on untrusted media? Does the cryptography employ standard algorithms and protocols? Are keys managed according to good practices?
- What are the trusted components of the system, when are they trusted and for what purposes? What parties are trusted and for what purposes? Can they exceed their authority? Are trusted components (and the implications of their compromise) properly documented?
- Is the system designed in a way that allows meaningful analysis? Is the architecture and code amenable to an external review (such as ours)? Could code analysis tools be usefully applied? Is there evidence of previous testing and other quality control practices?
- Are security failures likely to be detected? Are audit mechanisms reliable and tamper-resistant? Are data that might be subject to tampering properly validated and authenticated?

We used a variety of tools to support our analysis. A *Trac* Wiki and ticketing system on our secure network was used to summarize and track the various software issues under investigation and to provide a common knowledge base among the team members. The *Subversion* revision control system was used to manage source code and to produce this report. We used the *Fortify* static analysis tool to identify potential problem areas in parts of the Sequoia software. (We are grateful to Fortify Software for making the tool available to us.) Various debuggers, program editors, decompilers and other tools were used to experiment with and confirm software structure and behavior.

1.3 Limitations of This Report

Although our review was "top-to-bottom" in the sense of examining both the voting machine software and the vote-counting and election management software that interacts with it, our study by no means represents an exhaustive or definitive search for vulnerabilities or weaknesses.

Our analysis was focused chiefly on the system design and architecture. While the source code was available to us, the large size of the code base and the limited time available precluded a comprehensive review of the source code. Therefore, no assertions can be made about the existence or nonexistence of vulnerabilities in the source. No security analysis can guarantee discovery of all system vulnerabilities and due to the short time frame, this analysis is even more limited. We made our best effort to identify and prioritize security vulnerabilities based on generally known and accepted security principles, but we caution that additional vulnerabilities may exist that are not documented in this report.

The high stakes of many elections can provide rich incentives for illegal abuse. The resources of this study were quite small compared with those that might be available to an organized criminal conspiracy to commit election fraud. In our professional opinion, it is likely that a sophisticated adversary who sought to subvert an election by technical means would be able to duplicate, and perhaps exceed, the attacks reported here, even without open access to source code. Our study should be regarded, at best, as identifying obvious weaknesses that an attacker might attempt to exploit.

We did not attempt to verify that the voting software is completely free of defects or to exhaustively enumerate all defects, as that exceeds what is feasible with the state of art:

- Even a very thorough manual code review misses many problems. People are fallible: a code reviewer might overlook a defect in the code the same way that the developers who produced the code did. A manual review that found half of all problems in the code would be doing very well, by industry standards, but that would still leave many undetected defects. Moreover, manual code review often misses architectural flaws and other issues that do not clearly manifest themselves at the implementation level.
- Manual source code inspection is laborious, time-intensive, and costly. A rough estimate is that a trained software engineer can inspect approximately 100 lines of code per hour, under optimal conditions. If team members did nothing other than read source code for hours on end—something that few developers can sustain for any length of time—then it would have taken us over a year just to read all of the source code. That would have exceeded our budget and the time available to us.
- Automated code analysis tools (such as those that use static analysis techniques) are useful, but their value is limited in *post facto* analysis of large systems with heterogeneous code bases such as this review. These tools produce a high volume of "false positive" output, capture only a small subset of the kinds of errors that have security implications, and do not work with all programming languages and systems. To the extent possible, we used source code analysis tools, and found them helpful, but most of the work of the review still had to be done manually.
- We made no attempt to find all bugs or vulnerabilities in the code. Once we found several related vulnerabilities in the same portion of the code, we stopped looking for other vulnerabilities of the same type. We made no attempt to catalog all bugs that might enable any particular kind of attack. Instead, we structured our analysis as an attempt to find evidence to confirm or refute the hypothesis that the voting system is secure from tampering. Once we found strong evidence that some aspect of the system was likely vulnerable to a certain kind of tampering, we moved on to examine some other aspect of the system. This methodology was selected because of the limited time available for this review and because it seems to make little difference whether an attacker can choose among three or thirty different attack variations if all the variations have the same impact.

As a consequence, the list of issues and defects identified in this report should not be taken as a comprehensive catalog. In our review, we singled out a limited subset of the code as especially critical, based on our architectural analysis, and then subjected that portion of the source code to more intensive code inspection. Since we did find security vulnerabilities in the source code that was subject to inspection, one might anticipate that the rest of the code that we did not have time to inspect might also contain other vulnerabilities as well. For this reason, it is likely that this report underestimates the true number of vulnerabilities in the code, and it is possible that the issues identified in this report might represent only the "tip of the iceberg." However, because we were only able to inspect a small fraction of the code, we simply do not know whether the rest of the code contains security vulnerabilities as well. In summary, it seems reasonable to expect that there may be many other latent vulnerabilities that we are not aware of and that were not discovered during this review, but it is impossible to know for certain.

The analysis contained in this document is based on data gathered from documentation and source code provided by Sequoia, the State of California, and the ITAs (Independent Testing Authorities). We made no attempt to validate the materials provided to us or to confirm that the source code that we reviewed matches what is in the products used in California. Our conclusions depend on interpretation of those documents and source code. To the extent to which those materials are incomplete, inaccurate, or do not reflect the systems and practices currently in use in California, this may lead to material inaccuracies in this report.

We made no attempt to verify that the source code provided to us matches the binary code that is executed on election day in the Sequoia voting equipment. That was beyond the scope of this review. Also, while we were provided with binary executables for some of this software, we were not provided with a full build environment that would enable us to compile the source code for ourselves and verify that the results were identical to the binary executables provided to us.

Our team had only very limited access to sample voting machines and WinEDS installations (which were located in Sacramento and used primarily by the Red Team) and lacked of a complete build environment that would have allowed us to recompile the complete system from the source code and conduct more extensive experiments with system behavior.

We did not attempt to search for configuration problems or to analyze whether the voting system, as installed in California counties, is configured correctly. This was out of the scope of this work and we did not have access to counties' software installations in any case.

We did not attempt to analyze the procedures or processes or practices used by local election officials for potential security problems. We were given operator's manuals for the Sequoia equipment, and we frequently used them to gain insight how the system might be used in practice. However, we were not provided with detailed information about typical county-level practices and we were told it was beyond the scope of our review to examine how the voting system is used in the field.

The security of a voting system depends upon both the technology (e.g., the software) as well as upon how it is used (e.g., the processes and procedures in place). We were asked to focus in this review primarily on the software and accompanying documentation, not on how it is used. Therefore, any potential issues we identify in this report might or might not be relevant to any particular user of the equipment, depending upon the practices in place in that jurisdiction. It would require a separate, follow-on study to evaluate the impact of these issues on individual California counties and other users of this voting system.

The scope of this work was limited to analysis of the Sequoia voting system, not that of California's entire election system. For instance, voter registration systems, county practices, and election law were outside the scope of this study.

Our analysis was limited to a particular version of the Sequoia voting system. The report is not intended as an endorsement or repudiation of electronic voting in general.

1.4 Summary of Findings

We found significant security weaknesses throughout the Sequoia system. The nature of these weaknesses raises serious questions as to whether the Sequoia software can be relied upon to protect the integrity of elections. Every software mechanism for transmitting election results and every software mechanism for updating software lacks reliable measures to detect or prevent tampering. We detail these weaknesses, and their implications, in Chapters 3 and 4.

In certain cases, audit mechanisms may be able to detect and recover from some attacks, depending on county-specific procedures; other attacks may be more difficult to detect after-the-fact even with very rigorous audits.

There were numerous programming, logic, and architectural errors present in the software we reviewed. Some of these errors may be relatively harmless and reflect the large size and heterogeneous nature of the codebase. But other errors we found clearly have serious security implications. Many of the most significant vulnerabilities we found — those likely to be especially useful to an attacker seeking to alter election results — arise from four pervasive structural weaknesses, discussed in detail in Chapter 3:

- **Data Integrity.** The Sequoia system lacks effective safeguards against corrupted or malicious data injected onto removable media, especially for devices entrusted to poll workers and other temporary staff with limited authority. This lack of input validation has potentially serious consequences, including:
  - Precinct election results stored on DRE Results Cartridges and optical scan memory packs are not effectively protected against tampering. A poll worker with physical access to a Results Cartridge or MemoryPack before results are counted (e. g. when returning results to the county elections board) can change recorded votes, and, in some cases,

can introduce spurious results for other precincts. Under some conditions, a corrupted Results Cartridge may be able to cause damage to the WinEDS system itself when it is loaded for vote counting.

  – The safeguards against introduction of corrupt firmware into the precinct voting hardware are largely ineffective. An individual with even brief access to polling station hardware can tamper with installed firmware in a way that causes votes and paper trails to be recorded incorrectly, security logs to be corrupted, or ballots to be presented to voters incorrectly. Under some configurations and conditions, corrupt firmware may be able to be spread virally from compromised hardware and may persist across more than one election.

- **Cryptography.** Many of the security features of the Sequoia system, particularly those that protect the integrity of precinct results, employ cryptography. Unfortunately, in every case we examined the cryptography is easily circumvented. Many cryptographic functions are implemented incorrectly, based on weak algorithms with known flaws, or used in an ineffective or insecure manner. Of particular concern is the fact that virtually all cryptographic key material is permanently hardcoded in the system (and is apparently identical in all Sequoia hardware shipped to different jurisdictions). This means that an individual who gains temporary access to similar hardware (inside California or elsewhere) can extract and obtain the secret cryptographic keys that protect elections in every California county that uses the system.

- **Access Control.** The access control and other computer security mechanisms that protect against unauthorized use of central vote counting computers and polling place equipment are easily circumvented. In particular, the security features and audit logs in the WinEDS back-end system (used for ballot preparation, voting machine configuration, absentee ballot processing, and post-election vote counting) are largely ineffective against tampering by insider attackers who gain access to WinEDS computers or to the network to which the WinEDS computers are attached.

- **Software Engineering.** The software suffers from numerous programming errors, many of which have a high potential to introduce or exacerbate security weaknesses. These include buffer overflows, format string vulnerabilities, and type mismatch errors. In general, the software does not reflect defensive software engineering practices normally associated with high-assurance critical systems. There are many instances of poor or absent error and exception handling, and several cases where the software behavior does not match the comments and documentation. Some of these problems lead to potentially exploitable vulnerabilities that we identified, but even where there may not be an obvious vulnerability identified, the presence of such errors reduces our overall confidence in the soundness of the system as a whole.

Aspects of some of the weaknesses we discovered appear to have been reported in prior studies, most prominently in the 2006 Alameda County report[4], while others appear not to have been discovered (or publicly disclosed) previously.

Whether the vulnerabilities reported here represent practical threats, and whether their exploitation can be prevented or detected, depends heavily on the physical controls and procedures used in individual counties. We did not examine county-specific controls and procedures in our analysis. However, we caution that effective mitigation of many of the vulnerabilities discussed in this report will, at the very minimum, place considerable additional pressure on physical security features (such as locks and seals) and human procedures (such as two-person control by poll workers). Many of the physical security features and procedures used in the Sequoia system appear to have been engineered under the assumption that the underlying software is considerably more secure than it actually is, and thus may not provide sufficient protection in light of the vulnerabilities discussed here.

[4]C. Humphreys and C. Merchant. Pacific Design Engineering. *Sequoia Voting Systems Vulnerability Assessment and Practical Countermeasure Development for Alameda County.* October 4, 2006.

1.5 Report Organization

The rest of this report is organized as follows: In Chapter 2, we describe the Sequoia system components, the data flows and interactions among them, and the overall structure of the software that runs on them. In Chapter 3, we identify systemic issues that arose throughout the system's architecture and implementation. In Chapter 4, we detail specific vulnerabilities in the source code that give rise to the weaknesses discussed in Chapter 3. In Chapter 5 we discuss the implications of attack scenarios that exploit several vulnerabilities in combination with one another. Chapter 6 concludes with a brief discussion of the difficulty of mitigating the various vulnerabilities. In Appendix A, we give the reference threat model for an election, on which we based some of our analysis.

Throughout this report, markers of this form: ^{*1} refer to proprietary details of the Sequoia source code such as file names, function names, and line numbers. These references are given in a separate proprietary annex to this report.

CHAPTER 2

System Overview

The Sequoia voting system collects votes in three ways: touchscreen machines, paper ballots scanned at polling places, and paper ballots scanned at election offices. The following diagram shows how the parts of the system work together for these three methods of voting.

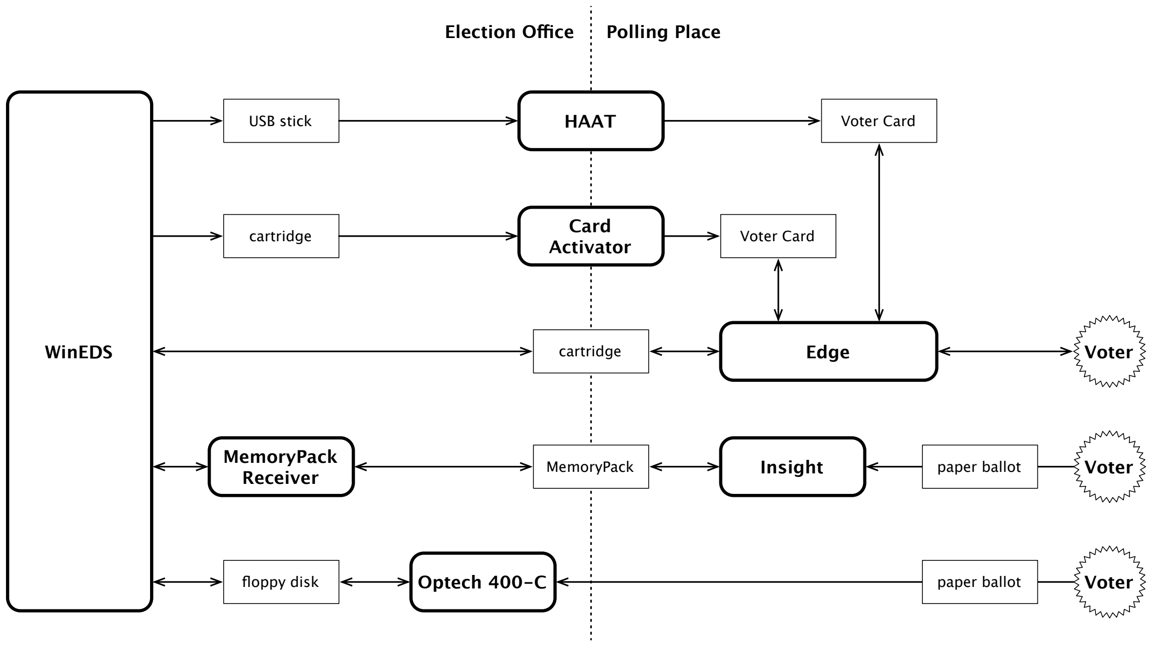

Figure 2.1: Simplified overview of voting system components. Each part in bold is a computer running software we reviewed; the small boxes are storage media. The arrows indicate the main information flows during typical use. HAATs, Card Activators, Edge cartridges, and MemoryPacks are prepared at election offices and then used in polling places.

WinEDS is the Election Database System. It collects and maintains all the information about the election, including offices, candidates, ballots, voting machines, polling places, and results.

The **Edge** is a touchscreen voting machine. To vote, a voter inserts a **Voter Card**. The **HAAT** and **Card Activator** are devices used in polling places to prepare a Voter Card for each voter. **Cartridges** are used to carry election information and cast ballot records between WinEDS and the Edges.

The **Insight** is a machine for scanning paper ballots one by one at a polling place. **MemoryPacks** are used to carry ballot information and vote counts between WinEDS and the Insights. The **MemoryPack Receiver** is a device for reading and writing MemoryPacks.

The **Optech 400-C** is a machine for quickly scanning large stacks of paper ballots at an election office. The software that controls the 400-C is called **WinETP**. Floppy disks are used to carry ballot information and vote counts between WinEDS and the Optech 400-Cs.

2.1 System Components

This section describes each component in more detail.

Many of these components are described as having "firmware" in their manuals and other technical documentation. The word "firmware" typically suggests something intermediate between "hardware" and "software" — easier to change than hardware, but harder to change than software. However, most of these components are computers, and their so-called "firmware" is simply an application program residing in storage that can be overwritten by programs running on the computer. In that respect, it is no different from software installed on the hard disk of a PC. To avoid giving the misleading impression that such software is more permanent than it actually is, we will often use the term "software" to refer to these programs.

2.1.1 WinEDS

WinEDS is a software program that runs on Windows PCs for entering, editing, collecting, and reporting on election information stored in a Microsoft SQL Server database. WinEDS is the central means by which election information is distributed to all types of voting equipment in polling places, and also by which cast votes are collected from all types of voting equipment. In its intended configuration, multiple computers running WinEDS all access a common database over a network on a computer running Microsoft SQL Server.

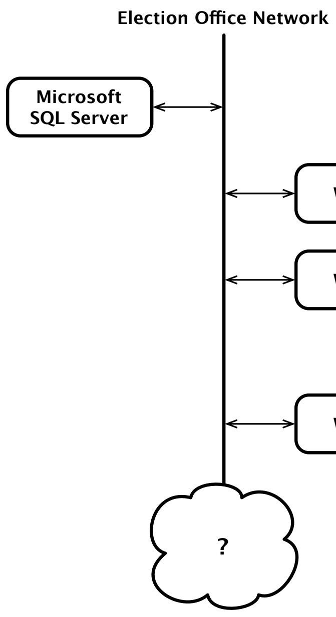

Figure 2.2: WinEDS on a network.

Operators can use WinEDS to enter and edit election information such as:

- offices and their associated political districts
- precincts and their assignment to political districts
- voting machines, their types, their serial numbers, and their assignment to precincts
- candidates and their parties
- the layout of paper ballots
- the layout of electronic ballots
- images and audio recordings for electronic ballots

Results are also stored in the database, such as:

- totals and records of cast votes
- provisional and write-in votes
- election statistics (such as turnout)

Operators can also manually enter results.

WinEDS includes an access control system based on user accounts and roles. Each user account has its own password and can be assigned one or more roles. Each role can be assigned a particular combination of access rights to different parts of the system (some examples of access rights are the ability to create new user accounts, the ability to edit candidates, or the ability to view tallied votes). In addition, each workstation on which WinEDS is installed can be further restricted to a particular set of users and roles.

2.1.2 HAAT

The *Hybrid Activator, Accumulator & Transmitter* (HAAT) is a portable, shoe-box sized device, used primarily to *activate* Voter Cards used by the Edge DRE.[1] The HAAT-100 model, *not* used in California, may also *consolidate* votes cast on Edges or Insights after an election, and *transmit* the unofficial tally of a precinct over a cellular network [2] Although the hardware differs between the two versions of the HAAT, the source code appears to be nearly identical. The source code is written primarily in C#, runs on a standard 486 processor, and is built on top of Windows CE 5.0.

Preparing the HAAT for an Election

Before a HAAT is used, it must be *prepared* or configured for a particular election and polling place, to work with any Edges in that precinct. This is done by copying files from a USB stick called a *Preparation Cartridge* created by WinEDS onto the HAAT. This switches the mode of the HAAT into *prepared* mode.

The HAAT has two modes of operation: **prepared** and **unprepared** mode. Menu options and access controls vary depending on the mode of the HAAT (see Section 3.3.2).

Usage at the Polling Place

A poll worker will use the HAAT, after verifying the eligibility of a voter, to *activate* a Voter Card with the type of ballot the Edge should give the voter (e.g., a Democratic Provisional Audio ballot). Once the voter casts her ballot, the *used* Voter Card is returned to the poll worker, who may eventually reprogram it for another voter. At any time, the current status of a Voter Card may be verified on the HAAT (to determine whether a Voter Card has been used, for example).

Usage During Early Voting

If a county uses Edges for Early Voting, a HAAT may be used to create cards that lock and unlock the Edge. These *lock/unlock cards* are materially identical to the Voter Cards used during an election except for the data they contain. Lock/unlock cards are "stamped" with an expiration time, after which an Edge will reject them as invalid. A lock/unlock card also maintains a record of the Edges it has been used with.

Upgrading the HAAT

The HAAT has a feature to make it easy to upgrade its software when new certified software becomes available. A password-protected menu option allows a Warehouse Technician to replace software on the HAAT with files on a USB stick called an *Upgrade Cartridge*.

[1]Note, a HAAT (or Card Activator) is not required to activate Edges. The Edge may also be used in *Manual Activation* mode.

[2]The HAAT-100 is designed to transmit the unofficial results over Verizon's CDMA 1X network.

Audit Logs Within the HAAT

The results of actions taken by the poll worker are are logged on the HAAT's internal removable Compact Flash memory. At any time, the log may be viewed through the small LCD screen of the HAAT, or uploaded to a USB stick.

Backup Cartridges

The HAAT is capable of creating a backup of parts of its internal memory. A backup operation will copy all logs and configuration files onto a USB stick called a *Backup Cartridge*. This feature is not password protected and may be performed at any time (note Section 3.3.2).

2.1.3 Card Activator

In some jurisdictions, a *Card Activator* (CA) is used in place of the HAAT. The Card Activator is similar in size and shape to the HAAT. Its software is primarily written in C, and runs on top of DOS on a 486 processor. Like the HAAT, it is used to *activate* Voter Cards used by the Edge DRE. The Card Activator also has source code for consolidating Results Cartridges and transmitting the results over a modem, but these features are disabled in this version certified for use in California.

Usage

The usage of the Card Activator is basically the same as for the HAAT (See Section 2.1.2). The Card Activator programs Voter Cards that determine which ballot the Edge presents to a voter. The Card Activator is also capable of creating lock/unlock cards for use in Early Voting, though this feature is not used on the system we evaluated (see Section 4.3.2).

Preparing the Card Activator

The Card Activator, unlike the HAAT has only one mode of operation.[3] All administrative features are available from the menu before, after and during an election. Typically, Warehouse Technicians prepare the Card Activator using a PCMCIA Preparation Cartridge created by WinEDS. Preparation Cartridges are materially identical to the Results Cartridges used by the Edge.

Upgrading the Card Activator

The software on the Card Activator may be updated as new certified software becomes available by using a PCMCIA Upgrade Cartridge[4] (note Section 3.3.2). The Upgrade Cartridge is typically provided by Sequoia. This Upgrade Cartridge contains a file with a particular name that gets copied onto the Card Activator, replacing the existing software.

Differences Between the Card Activator and the HAAT

- *No audit log.* The Card Activator does not keep any logs. The source code makes calls to a logging function, but the function is vacuous and discards all information passed to it (See Section 4.3.9).
- *No backup cartridges.* Unlike the HAAT, the Card Activator does not have a *backup* option.
- *Non-removable Flash memory.* Unlike the HAAT, the internal Flash memory of the Card Activator (the disk that stores the operating system, application and configuration files) is integrated onto the board, rather than being a removable Compact Flash card.

[3]The HAAT has two modes of operation: prepared and unprepared mode. Features and password settings vary depending on the mode.

[4]Unlike the HAAT, this menu option is not password-protected.

- *Limited use of passwords.* The Card Activator does not password-protect most of its features. Passwords are not used by default on the Card Activator (see Section 4.3.2).
- *Software is procedurally written.* Whereas the HAAT is object-oriented, reflecting modern programming practices, the Card Activator is written in a less-structured manner.

2.1.4 Edge

The Edge is a stand-alone Direct Recording Electronic (DRE) touch screen voting machine, accompanied by a Voter-Verified Paper Audit Trail (VVPAT) printer which provides a paper record of the vote for review by the voter.

The Edge system is a special purpose computer, including both proprietary hardware and software. The self-contained unit weighs approximately 40 pounds, and is designed to be easily transported between the county warehouse and polling location.

Edge Hardware

The Edge hardware includes a special purpose CPU board which includes a LCD and touchscreen controller, a Compact Flash device, three serial EEPROMs, two Type II PCMCIA slots, a Smartcard slot and Ejector, a parallel printer port, audio port, and various other components. We discuss the function of only a few of these components.

The LCD **Touchscreen** is the primary interface between the user and the Edge. Poll workers may use the touchscreen to configure the Edge and load new elections. Voters use the touchscreen to view the ballot and make selections.

The **Audit Trail** is stored on a Compact Flash device, and is the primary storage device for the Edge. In addition to the executable code for the Edge, the Audit Trail stores a chronological **Event Log** which records major system activity since the last machine reset. The Audit Trail also stores a redundant copy of the randomized record of votes cast during an election.

The Edge uses the three serial EEPROMs for the Configuration ROM, Protective Counter, and Public Counter. The **Configuration ROM** stores the default Edge configuration. The **Protective Counter** stores a counter which is incremented every time a vote is cast. This counter provides the number of votes cast during the lifetime of the machine. The **Public Counter** is also incremented every time a vote is cast, but is reset every election. Different public counters are kept for the different election modes.

Since the Edge is a stand-alone unit with no network connectivity, it must use special cartridges (see 2.1.6) for communication with WinEDS. For this purpose, the Edge has two type II PCMCIA slots referred to as the **Results Port** and the **Auxiliary Port**. The slots are located on the back of the unit, and are covered with security seals to prevent tampering[5].

The Smartcard slot and Ejector are located at the front of the machine, and allow polling places to use special Voter Cards (see section 2.1.5) to activate the Edge machine for voting.

There are several components at the back of the Edge designed for use only by the poll workers. This includes a small LCD display, called the **Pollworker LCD**, which displays brief messages for the poll worker regarding the status of the machine. In addition to the Pollworker LCD, the back of the Edge has a yellow **Activate Button**, which activates the Edge for voting if manual mode is enabled. This button also allows a poll worker to force the Edge into Maintenance Mode, and to perform a system reset. The poll worker also has access to two switches at the back of the machine. The **Power Switch** allows the poll worker to power on and off the machine. The switch is a soft switch, meaning it sends a power off signal to the Edge. The **Polls Open/Closed Switch** allows the poll worker to open and close the polls.

[5]Unfortunately, these seals are ineffective. See the UCSB red team report for details.

VeriVote Printer

The printer port allows for an optional **VeriVote Printer** to be connected to the Edge machine[6]. The printer is mounted on the side of the machine, allowing voters to see its display area. The printer uses thermal paper up to 300 feet in length, allowing for up to 150 votes per roll. When a printer runs out of paper during an election, a new printer is connected to the Edge.

The VeriVote provides a paper record of the vote for verification by the voter, and often referred to as the VVPAT printer. If the vote is correct, an accepted string is printed on the receipt and the vote is recorded on the Audit Trail and Results Cartridge. The printer then scrolls the last vote off the display area to protect voter privacy. Poll workers may also print various reports, such as event log and cartridge reports.

Edge Software

The Edge is written primarily in approximately 124,000 lines of C code. The executable software run by the Edge is stored on the Audit Trail.

The primary function of the Edge is to record votes cast during an official election. This includes loading ballots, opening the polls, closing the polls, and optionally consolidating the votes.

The Edge software provides a wide range of additional functionality. Numerous tests are allowed from the maintenance diagnostics screen. This includes a LCD, printer, internal RAM, audit trail memory, auxiliary cartridge memory, smartcard memory and EEPROM memory tests. The Edge also provides various reports after an election, such as results, zero proof, consolidation, event log, audit trail, early voting, and ballot image reports.

The Edge also supports both pre-election logic and accuracy testing (Pre-LAT) and post election logic and accuracy testing (Post-LAT). To automate this testing process, the Edge also supports vote simulation using special Simulation Scripts (see refoverview-interpreted) loaded from Simulation Cartridges (see 2.1.6).

Updating Edge Software

The Edge software provides “firmware update” functionality to upgrade the executable software on the Audit Trail without needing to access or update any of the internal hardware components. When an Edge is powered on, it checks for an Update Script (see section 2.1.11) on the Audit Trail. If found, the Edge automatically runs this script without requiring any user interaction.

Otherwise, the Edge checks for a special Update Cartridge (see section 2.1.11) in the Results Port. To protect the data integrity of these cartridges, a manifest file is included on the Update Cartridge. This file contains a list of file names to be copied onto the Edge. Each file name, including the manifest file, is followed by a “keyed cryptographic signature.” The Edge calculates the signatures for each of these files, and compares it with the signature stored in the manifest. If there is a discrepancy between these signatures, the update process fails.

To further protect this process, a password is required to enable the update. This password is encrypted with a separate key and stored on the Update Cartridge. Theoretically, only someone with an authentic firmware cartridge, physical access to an Edge, and the correct password may complete the “firmware update” process.

Outside of entering the password, the update process requires little interaction. First, the Edge must be powered off. The Update Cartridge is placed in the Results Port of the Edge, and the machine is powered back on. The Edge detects the Update Cartridge, and prompts the user for a password. If correct, the Edge uses the manifest file to validate the cartridge. Once verification passes, a firmware update event is added to the event log and the listed files are copied onto the audit trail. The new executable code is run the next time the machine is powered on.

[6]This printer is often referred to as the VVPAT, since it provides the voter verified paper audit trail.

Operating Modes and States

Portions of the Edge are designed as a state machine, performing different tasks based on the current state or operating mode. There are five primary modes defined.

Maintenance Diagnostics mode allows the user to perform numerous diagnostic tests. This is the default mode entered when the system is reset. Once a ballot is loaded on the system, the Edge enters Pre-LAT Mode.

Pre-LAT Mode allows pre-election logic and accuracy testing, both manually and from a Simulation Cartridge (see 2.1.6). For testing to start, the Poll Open/Close Switch must be set to "Open." When testing is complete, the switch is moved back to "Closed." This will then allow the Edge to enter Official Election Mode. **Post-LAT Mode** allows for logic and accuracy testing after the completion of an election.

Early Voting Mode allows for multiple early voting sessions before the official polls are open. Normal early voting is also available from the Official Election Mode. **Official Election Mode** allows for official votes to be cast on election day. This is done using a special election state machine, which performs different tasks based on the current election state.

There are six different states used by the election state machine. The **Waiting State** indicates the system is waiting for the polls to be opened by switching the Poll Open/Close Switch to "Open." The **Polls Open State** indicates the polls have been opened, and the **Polls Closed State** indicates the polls have been closed. Both of these operations occur by moving the Poll Open/Close Switch on the back of the Edge. The **Zero Proof State** causes a zero proof report to be printed on the VeriVote printer, and the **Print Results State** causes a results report to be printed. The **Process Voters State** indicates the polls are open, and the system is ready to begin processing voters. This causes the voting state machine to be entered.

There are four states used by the voting state machine. The **Start Voting State** is the initial state of the machine, and handles setting up the screen for the first voter. Once complete, it enters the **Waiting For Voter State** which waits for the system to be activated for voting (see section 2.2.1). When activated, the system enters the **Voter Active State** or **Audio Active State** to indicate a voter is now able to vote on the Edge.

2.1.5 Smartcards

Voter Cards serve the purpose of controlling access to the Edge voting machines. The HAAT is capable of creating two kinds of Voter Cards:

- *Auto Activation* cards. These cards are given to voters, and allow each to select and cast his or her vote.[7]
- *Early Voting* cards. These cards are created for elections officials to lock and unlock Edges during the early voting phase. Early Voting cards may be used multiple times and on multiple Edges during an election.

Often, smartcards contain a microprocessor capable of performing its own computations. The smartcards used in the Sequoia System, however, do not utilize any processing features on their smartcards. Smartcards, as used by the Sequoia System, are simply storage devices.

2.1.6 Cartridges

WinEDS and the Edge use **cartridges** as a primary means of communication. These cartridges are specially formatted PCMCIA cards provided by Sequoia. Due to their small size, the cartridges are equipped with custom jackets to make the cards easier to handle for poll workers.

The WinEDS system uses a standard PCMCIA reader with an extender to accommodate the custom jacket. The Edge has two type II PCMCIA slots, referred to as the **Results Port** and the

[7]The Edge can also be activated manually, without any Voter Card. Hence, the Voter Card is known as an auto activation card.

Auxiliary Port. The slots are located on the back of the unit, and are covered with security seals to prevent tampering.

According to the source code, there are 10 different types of cartridges, as discussed below. Of these, WinEDS supports formatting of 6 cartridge types: Results, Consolidation, Simulation, Early Voting, Audit Trail Transfer, and Technician Cartridges. We chose to focus our source code analysis on just 4 types of cartridges: Results, Consolidation, Master Ballot, and Update Cartridges.

Results Cartridge

A **Results Cartridge** serves two primary functions. Before an Edge may be used in an election, it must be configured using a Results Cartridge placed in the Results Port of the Edge. During the election, the Edge uses this cartridge to record all votes cast on that machine.

One Results Cartridge for every Edge machine must be created by WinEDS. After the polls close, WinEDS reads each of these cartridges to perform the final vote tally. More information on the life cycle of these cartridges may be found in section 2.2.1.

Consolidation Cartridge

Depending on the election configuration, votes across multiple Edge machines may be consolidated onto a special **Consolidation Cartridge** created by WinEDS. After the polls close, poll workers place the Consolidation Cartridge in the Auxiliary Port of each Edge and perform the vote consolidation. This allows polling locations with multiple Edge machines to return the single Consolidation Cartridge for tallying in WinEDS.

Simulation Cartridge

Every Edge must go through pre-election logic and accuracy testing (Pre-LAT) before entering official voting mode. This process attempts to verify the Edge is correctly counting votes. To automate this process, a special **Simulation Cartridge** may be placed in the Auxiliary Port of the Edge. This cartridge, created by WinEDS, contains a script to cast numerous votes automatically. Once complete, the county election officials can verify the returned results match their expectations without having to cast any votes manually.

Early Voting Cartridge

Early voting is a mode supported by the Edge which allows individuals to vote before the official election start date. The early votes may be stored on a Results Cartridge, or optionally on a separate **Early Voting Cartridge** placed in the Auxiliary Port of the Edge. The Early Voting Cartridge is returned with the Results Cartridge for tallying in WinEDS.

Audit Trail Transfer Cartridge

The Edge records election results on both the internal Audit Trail and on the inserted Results Cartridge. Should the Results Cartridge become damaged or lost, the election results may be recovered from the internal Audit Trail. This is done through a special **Audit Trail Transfer Cartridge** placed in the Auxiliary Port of the Edge while the polls are closed. WinEDS uses this cartridge to recover the lost election results.

Master Ballot Cartridge

Normally, WinEDS must create one Results Cartridge for every Edge being used in an election. This process may be simplified by using a single **Master Ballot Cartridge** to configure multiple Edge machines. The Master Ballot Cartridge is placed in the Auxiliary Port of the Edge while an empty Results Cartridge is placed in the Results Port. The appropriate files and settings are copied onto the Results Cartridge from the Master Ballot Cartridge, and the machine is configured from the

Results Cartridge as before. When complete, the Master Ballot Cartridge may be used to configure another Edge machine.

Update Cartridge

Occasionally, it may be necessary to update the software running on an Edge machine. The software is stored on **Audit Trail** storage, which is a Compact Flash device on the CPU board. To avoid having to open the protective casing and access the compact flash device directly, the Edge provides "firmware update" functionality through an **Update Cartridge** provided by the vendor. This procedure allows new executable code to be installed, along with other files such as new bitmap images or font files.

Audio Cartridge

This cartridge type is defined in the Edge source code. However, it is unclear from the documentation what specific purpose these cartridges serve.

Technician Cartridge

This cartridge type is defined in both the WinEDS and Edge, and may be formatted through the WinEDS interface. However, it is unclear from the documentation what specific purpose these cartridges serve.

Internal Audit Trail Cartridge

This cartridge type is defined both in the WinEDS and Edge source code. However, it is unclear from the documentation what specific purpose these cartridges serve or how they are created.

2.1.7 Insight and Insight Plus

The Insight and Insight Plus are precinct-based optical scanners installed on top of a ballot box at a polling places. The scanner constitutes the top cover of the ballot box, so that ballots must be inserted through the scanner to enter the ballot box. Messages and vote counts are printed on a paper tape that comes out of the top of the scanner.

The Insight and Insight Plus are nearly identical machines; they appear to differ only in that the Insight Plus has a 2-line-by-24-character LCD on the front for displaying messages to voters. We were provided with just one set of source code for both, so we assume they run the same software, and will use "Insight" to refer to both machines.

Voting

A voter at an optical-scan polling place would insert a marked paper ballot into the slot at the front of the Insight; the Insight feeds the ballot through while scanning the marks on the paper. Depending on various criteria (for example, whether the ballot has a write-in, whether the ballot is overvoted, and so on), the Insight then directs the ballot into a front bin or a rear bin in the ballot box or back out the front slot to the voter. A four-digit LED on the front of the Insight (separate from the LCD on the Insight Plus) displays a counter that increments with each accepted ballot.

Administration

There is a locked door on the back of the Insight that provides access to a numeric keypad, the paper tape spool, and a slot for a MemoryPack. A MemoryPack must be present for the Insight to operate.

The operator can use the keypad to open and close the polls, print results on the paper tape, and override settings on the Insight (for example, forcing it to accept a spoiled or overvoted ballot). In addition to printing results, the paper tape is used for communicating with the operator. It prints out messages to the operator and records various events throughout the day (for example, the time that polls were opened).

Software

The Insight contains a small computer based on a Z80 chip. The software that controls the Insight is divided into two parts: the HPX, which resides on an EPROM chip in the Insight, and the APX, which resides on the MemoryPack in the Insight. Every MemoryPack is initialized (by the vendor, we assume) with a copy of the APX software in part of its flash memory.

When the Insight is turned on, it begins running the HPX. After the HPX does a memory test and a few hardware checks, it transfers full control to the APX software, which is responsible for scanning ballots, counting votes, directing received ballots to the appropriate bins, printing results on the paper tape, and all administrative interaction with poll workers and technicians via the keypad and paper tape printer.

2.1.8 MemoryPack

A MemoryPack is a box about the size of a paperback book containing memory chips and a battery. The memory is divided into three areas:

- Flash memory containing the APX software for the Insight.
- Flash memory containing the description of the ballot.
- Battery-backed RAM containing the counts of recorded votes.

Flash memory retains its stored information even when there is no power, but must be erased entirely or in large blocks (which takes a few seconds) before new information can be written to it. On the other hand, RAM can be rewritten at arbitrary locations very quickly, but its stored information vanishes when the power is turned off; hence each MemoryPack has a dedicated battery to keep its RAM alive.

The flash memory containing the ballot description and the RAM for the vote counts are both divided into **pages** (equal-sized sections of memory, where one page at a time is selected as the active page for reading and writing data). A single MemoryPack can store ballot descriptions and vote counts for multiple precincts, with each precinct corresponding to a particular page number. Each ballot can have a *header code* printed on it to indicate its precinct. An Insight can accept ballots for many different precincts by scanning the header code on each ballot and activating the corresponding memory page on its MemoryPack for reading the ballot description and updating the vote counts.

2.1.9 MemoryPack Receiver (MPR)

The MemoryPack Receiver (MPR) is a device for reading and writing data on MemoryPacks. It is a box about the size of a VCR with a slot on the front for a MemoryPack and a 25-pin serial port on the back.

Usage

To use the MPR, the operator connects the serial port on the back of the MPR to a PC using a serial cable. WinEDS, running on the PC, sends commands over this serial connection to read data from the MemoryPack's flash memory and RAM, to write data into the MemoryPack's flash memory, and also to write data into the MPR's own memory.

Software

The MPR contains its own small computer based on a Z80 chip, running software that controls how it communicates to the PC and how it reads and writes data on the MemoryPack. This software is divided into two parts: built-in software that resides permanently on the MPR and temporary software that is uploaded from the PC to the MPR. The temporary software disappears from the MPR's memory when the MPR is turned off. Each time the MPR is turned on and used, WinEDS re-uploads a copy of the temporary software to the MPR before sending other commands.

2.1.10 Optech 400-C

The Optech 400-C is a high-capacity, high-speed optical scanner for paper ballots intended for use at a central election office. It consists of a Windows PC with a display, mouse, and keyboard, attached to a large chassis containing ballot bins, a mechanical ballot feeding apparatus, and optical sensing equipment.

Usage

To begin using the 400-C, the operator loads an election file (from a floppy disk or over a network) that describes the types of ballots to be scanned and their layout. The operator then stacks ballots in the input hopper and clicks buttons in the user interface on the Windows PC to start and control the scanning process.

Software

WinETP is the software application installed on the included Windows PC that interacts with the operator and controls the feeding and sensing electronics in the 400-C. WinETP can be set to run in *standalone mode* or in *counting station mode*.

2.1.11 Interpreted Code

In addition to all the software mentioned so far, whose source code we were given to review, we found six different kinds of *interpreted code* that are also executed in the Sequoia voting system. By **interpreted code**, we mean a sequence of instructions that is executed not by the computer hardware but by an **interpreter**—a software program that reads each instruction, determines its meaning, and carries out the instruction. The Sequoia documentation refers to the six kinds of interpreted code as **update scripts**, **simulation scripts**, **pack scripts**, **B-Code**, **E-Code**, and **R-Code**. For all of these languages:

- The interpreter is part of the Sequoia software we were given to review.
- The language was invented by Sequoia programmers.
- The language was created for a specific, limited purpose.

We do not consider the mere presence of interpreted code to be grounds for a positive or negative opinion about system security. We discuss the significance of interpreted code in Section 3.4.2.

Update Scripts

Update scripts are used in the Edge "firmware update" process. An update script contains a sequence of textual commands to replace, move, or delete files on the internal disk of the Edge. There are also commands to set the Edge's serial number and protective counter. The Update Cartridges and scripts are created and provided by Sequoia.

Update scripts can contain looping and goto commands, but do not have variables, expressions, or subroutine calls. Some commands (such as a command that checks the CRC on a file) report success or failure; failure aborts the script.

Simulation Scripts

Simulation scripts are used to automate pre-election logic and accuracy testing (Pre-LAT) on the Edge. WinEDS has a Script Editor feature for helping users compose these scripts. These scripts contain textual commands that include activating the Edge for voting, changing the selected language, and casting specific votes (including write-in votes). After a script is composed, WinEDS can write it to a Simulation Cartridge (see section 2.1.6) for use in an Edge.

Simulation scripts have a simple looping feature for repeating commands a fixed number of times. Simulation scripts do not contain variables, expressions, branching, or subroutine calls.

Pack Scripts

Pack scripts are both generated and executed by WinEDS during the preparation of MemoryPacks before election day. A pack script contains a sequence of textual commands to read, write, and erase sections of memory in a MemoryPack. These commands are carried out one after another immediately after the script is generated. (The information loaded onto a MemoryPack by a pack script includes B-Code.)

Pack scripts do not contain variables, expressions, branching, looping, or subroutine calls.

B-Code

B-Code is a binary instruction format used to control how the Insight counts ballots and prints reports. The ballot definition on each MemoryPack consists of B-Code generated by WinEDS according to the ballot style or styles that a particular Insight is expected to accept. B-Code instructions specify configuration settings for the Insight, text (such as candidate names) to be printed on the paper tape when reporting results, the locations of marks on the ballot for each candidate, and details of counting for special cases such as primaries, recall elections, and straight party voting.

B-Code allows simple conditional branching, but does not have variables, expressions, looping, or subroutine calls. The Insight APX software makes multiple passes through the sequence of B-Code instructions on the MemoryPack each time a ballot is scanned, and another pass when printing out results on the paper tape. Each instruction may have a different effect depending on which pass is being made. For example, an instruction giving the name of a candidate has no effect during a scanning pass, but will print out the name on the paper tape during a reporting pass.

E-Code

E-Code is a binary instruction format used to control how the Optech 400-C counts ballots. E-Code instructions manipulate an array of counters based upon an array of detected ballot marks. WinETP (the software that runs on the 400-C) contains a virtual machine that executes E-Code whenever a ballot is scanned, as well as a compiler that generates E-Code based on an election description. The 400-C also saves and loads election files containing compiled E-Code.

E-Code is designed in the style of a machine instruction set, with variable-length opcodes, registers, arithmetic operations, jump instructions, conditional branching, and subroutine calls.

R-Code

R-Code is a binary instruction format used to control how the Optech 400-C produces reports. It consists of a sequence of instructions that specify text and counters to print, and some simple arithmetic operations on counters.

R-Code does not contain variables, expressions, branching, looping, or subroutine calls.

2.2 Election Procedures

This section describes our understanding of how the above components are used in an election. Our findings in this report assume that elections follow these general procedures, though detailed procedures may differ from county to county.

2.2.1 Touchscreen (DRE) Voting

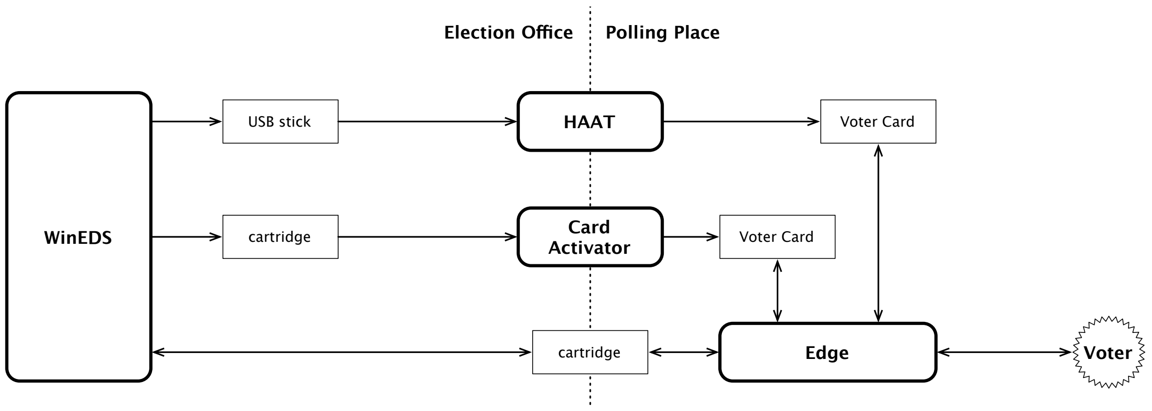

Figure 2.3: Components involved in DRE (touchscreen) voting.

The election process for DRE voting involves four primary systems: WinEDS, Edges, Card Activators, and HAATs.

Preparation

The election process begins by creating an election in WinEDS. This is an involved process, requiring an expert user. The election must be setup with the correct configuration, candidate and contest information must be added, machines must be assigned to polling locations, and multiple language files must be created and assigned.

Once the election is prepared in WinEDS, the other components may be configured. For how to prepare the HAAT or Card Activator for an election, please see sections 2.1.2 and 2.1.3.

An election administrator must create one Results Cartridge using WinEDS for every Edge machine. Once created, the cartridge contains the Edge serial number assigned to that cartridge and the ballot description. The Results Cartridge is then loaded onto the Edge at the county warehouse. Once successfully loaded, the administrator must open the polls and perform pre-election logic and accuracy testing (Pre-LAT) on every Edge. This process may be automated through the use of Simulation Scripts (see refoverview-interpreted) and Simulation Cartridges (see 2.1.6). When complete, the polls are closed and the Pre-LAT results are evaluated. If the Pre-LAT tests reveal no problems, the Edge is powered off and shipped to the polling location.

Election Day

On election day, the poll workers unpack the Edges and power them on. When it is time to begin voting, the poll workers move the Poll Open/Close Switch to “Open” and then start processing voters.

In order to vote on election day, a voter must first obtain a Voter Card from a poll worker. The Voter Card is activated by the poll worker using either a Card Activator or HAAT. Inserting the Voter Card into an Edge activates the Edge to allow the voter to begin voting. Before the vote is recorded, the voter is given the chance to review his or her vote on the VeriVote Printer. Once the vote is confirmed, the string “Accepted” is printed on the voter-verified paper audit trail (VVPAT), and the Edge stores a record on the Results Cartridge and Audit Trail for each voter's selections. When this process is complete, the Voter Card is deactivated and ejected. The voter then returns the Voter Card to the poll worker before leaving the polling location.

Vote Tallying

When voting is complete, the poll worker turns the Poll Open/Close Switch to "Close" and prints out the appropriate reports. The Results Cartridges are collected and brought back to the WinEDS computer for final tallying. The cartridges are inserted and processed by WinEDS one by one. Once the tally is complete, the election administrator can use WinEDS to declare winners.

2.2.2 Precinct-Based Optical Scan Voting

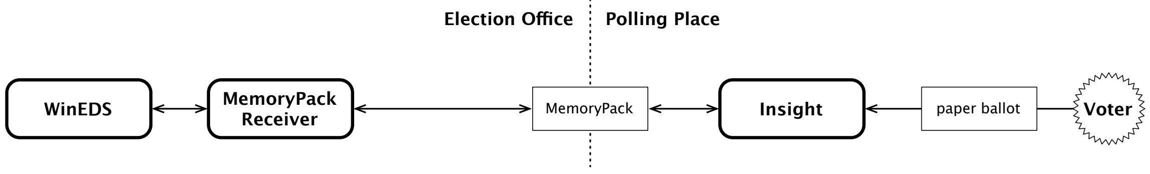

Figure 2.4: Components involved in precinct-based optical scan voting.

There is one MemoryPack for each Insight machine. The MemoryPack carries descriptions of the ballots to be scanned from WinEDS to the Insight, and carries the counts of scanned votes from the Insight to WinEDS.

An election administrator uses WinEDS to prepare the MemoryPacks prior to election day. The MPR is connected to the WinEDS computer with a serial cable. The WinEDS database maintains a list of serial numbers of Insights[8] and the precincts to which they are assigned. The operator selects each machine, one by one, and inserts a MemoryPack into the MPR; WinEDS writes the chosen serial number and the associated precinct's ballot description to the MemoryPack. The prepared MemoryPacks are then placed into Insights.

On election day, poll workers turn on the Insights and voters insert ballots. Each time an Insight scans a ballot, it increments the counters on its MemoryPack according to the ballot description on the MemoryPack. The ballot description indicates the locations of candidates on the ballot and how they should be counted. For example, it might indicate, "Abraham Lincoln is the third candidate in the second column; each time a mark is detected in this location, increment counter number 15."

After election day, the MemoryPacks are collected and brought back to the WinEDS computer. Again they are inserted into the MPR one by one; WinEDS reads the counters from each MemoryPack and then marks that MemoryPack's serial number as "processed." WinEDS will refuse to load data from a MemoryPack with that serial number again, unless the operator manually deletes the existing record of that serial number.

WinEDS uses its knowledge of the MemoryPacks it prepared to convert the counter data into vote tallies. For example, upon determining that the inserted MemoryPack has serial number 5217, WinEDS would look up in its database the precinct associated with Insight number 5217. It would then look up the ballot style corresponding to that precinct to determine that counter number 15 counts votes for Abraham Lincoln. These totals are then loaded into the database, along with a record of which MemoryPack they came from.

[8]In fact, these are serial numbers of MemoryPacks, even though WinEDS shows them as Insight serial numbers. The "Insight serial number" that the Insight prints on its paper tape is just the serial number that was written to the MemoryPack. As far as we could tell, any MemoryPack can be placed in any Insight; Insights have no identity other than the identity of the MemoryPack inside.

2.2.3 Centrally Counted Optical Scan Voting

When an election uses paper ballots that are centrally scanned, the paper ballots are all collected and brought to Optech 400-C machines at the county-level election office.

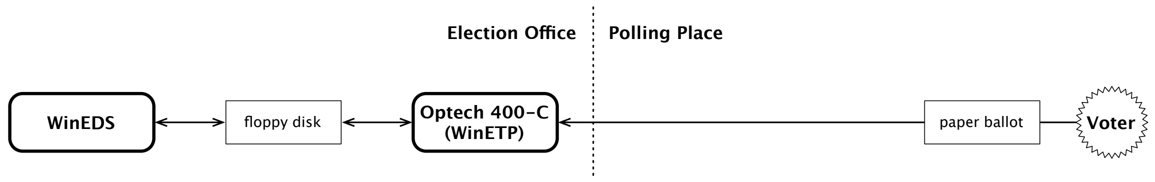

Figure 2.5: Standalone centrally counted optical scan voting.

When WinETP, the software that runs the 400-C, is configured in *standalone mode*, the operator loads an election description using a floppy disk prepared by WinEDS. This election description can contain the layouts for many different ballot styles. The operator can manually choose a precinct and feed in a stack of ballots for just that precinct; or, the operator can feed in a stack of mixed ballots and ask the 400-C to determine the precinct (and hence the layout) for each ballot based on a code printed on the ballot. If needed, the operator can manually delete the results for a batch of scanned ballots and rescan the batch.

During or at the end of the scanning process, the election totals can be saved to a floppy disk, and these totals can then be loaded into WinEDS. The operator can also ask the 400-C to generate various reports of the election results to be printed or saved to the floppy disk.

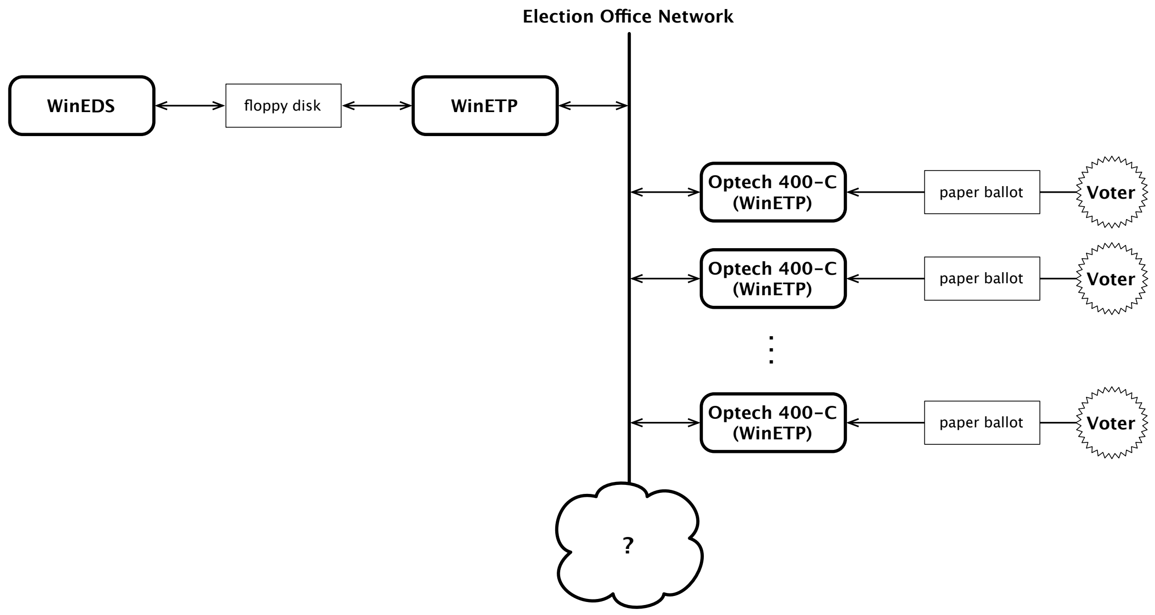

Figure 2.6: Networked centrally counted optical scan voting.

WinETP can also be configured in *counting station mode*. A 400-C machine in this mode will load election information over a network from a *master system* and report its results back to the master system. The master system is another computer running WinETP in *master system* mode.

CHAPTER 3

Systemic and Architectural Issues

We found significant security weaknesses throughout the Sequoia software. They raise serious questions as to whether the Sequoia software can be relied upon to protect the integrity of elections.

There were numerous programming, logic, and architectural errors present in the software we reviewed. Some of these errors may be relatively harmless, simply reflecting the large size and heterogeneous nature of the codebase. But other errors we found have potentially serious security implications. Individual software problems are detailed in Chapter 4. Some of these problems are quite serious by themselves. Taken together and under certain circumstances, however, these weaknesses may have the capacity to enable devastating security breaches that alter the results of elections.

The most significant security weaknesses we identified — those likely to be especially useful to an attacker seeking to alter election results — arise from four pervasive structural weaknesses and engineering failures, discussed in detail in the sections that follow.

Throughout this report, markers of this form: ^{*1} refer to proprietary details of the Sequoia source code such as file names, function names, and line numbers. These references are given in a separate proprietary annex to this report.

3.1 Data Integrity

Some of the most serious weaknesses in the Sequoia system relate to ineffective mechanisms for assuring the integrity of election data and software.

In the Sequoia system, much of the data that determines the outcome of an election — precinct results, voting machine configuration files, ballot definitions, and so on — reside on removable media that may pass through several sets of hands. Precinct results, voting machine configurations, and ballot definitions are stored on small, removable memory units that are handled by poll workers and possibly others during the normal course of an election. At the same time, Edge DREs, HAATs, Card Activators and Insight optical scan readers are themselves configurable devices, controlled by software that can be loaded through easily accessible external interfaces. The integrity of elections therefore depends critically on the integrity of removable media and of the software that controls polling station equipment.

Unfortunately, the software mechanisms that safeguard these critical election components are largely ineffective or absent from the Sequoia system, making the system completely dependent on the physical security of its components and on the rigorous control of digital media used in and produced by precincts. It is a relatively simple matter to place counterfeit precinct results on Edge Results Cartridges or Insight MemoryPacks such that these results will be accepted as valid by WinEDS. Malicious software that records votes incorrectly can be loaded easily into precinct hardware by anyone with brief physical access to the equipment.

The standard audit mechanisms used in California (such as 1% recounts and parallel testing) may be only marginally effective in detecting and recovering from the attacks against data integrity we discuss in this section.

3.1.1 Integrity of Precinct Results

Precinct election results stored on Edge Results Cartridges and Insight optical scan memory packs are not effectively protected against tampering. A poll worker or other person with physical access to a Results Cartridge or memory pack before results are counted (e. g. when returning results to the county elections board) can change recorded votes, and, in some cases, can introduce spurious results for other precincts. Under some conditions a corrupted Results Cartridge may be able to cause damage to the WinEDS system itself when it is loaded for vote counting.

Insight MemoryPacks

Precinct results collected by Optech Insight optical scan precinct ballot readers are stored on special *MemoryPacks* that are typically removed from the precinct polling station and returned to the county elections headquarters at the close of the election day for tallying through WinEDS.

MemoryPacks are not protected from unauthorized reading or tampering by cryptography or any other software-based mechanism[1]. WinEDS cannot reliably detect if a MemoryPack has been altered or if a MemoryPack purporting to be associated with a particular Insight from a particular polling station indeed originated there. Anyone with sufficiently unrestricted physical access to a MemoryPack on its way from a polling station to the county headquarters can alter precinct results, given the proper read/write hardware. See Sections 4.1.28 and 4.1.29.

The MemoryPack physical interface for reading and writing data is not based on any widely available computer industry standard (it appears to be proprietary to Sequoia, although we did not conclusively verify this). Any attacker who wants to alter a MemoryPack would have to construct or obtain the specialized hardware required to communicate with it. However, the proprietary nature of the interface is not likely to provide substantial protection here. Used optical scan hardware (including memory packs and Optech ballot scanners) is openly available on the surplus market, and could likely be adapted by an attacker for this purpose.

Edge Results Cartridges

Precinct results collected by Edge DRE machines at precincts are stored on industry-standard PCMCIA *Results Cartridge* memory modules that, like Insight MemoryPacks, are typically removed at the close of the election day for tallying at the county elections office through WinEDS.

Results Cartridges are supposed to be protected by cryptography to detect tampering and counterfeiting. Unfortunately, as noted in Section 3.2, design and implementation errors render the cryptography ineffective.

The consequences of this ineffective protection are serious. As with Insight MemoryPacks, WinEDS cannot determine reliably whether the ballots recorded on a Results Cartridge have been altered, added or deleted or whether a Results Cartridge actually came from the Edge and precinct from which it purports to have originated. Anyone with physical access to a Results Cartridge on its way from a polling station to the county headquarters can alter precinct results. See Section 3.2.

Because the Results Cartridges are off-the-shelf, industry standard PCMCIA memory cards, readily available computer equipment (including most laptop and many palmtop computers) is sufficient for carrying out Results Cartridge attacks. No special hardware or reverse-engineering of proprietary interfaces is involved.

Besides altering the results for the precinct, additional attacks are also possible. Counterfeit extra returns for precincts other than the one from which a cartridge originated can be added to a Results Cartridge. These extra results are added to the other precincts' totals when read into WinEDS but are not listed in several of the tally audit reports that WinEDS generates, making the attack difficult to detect and isolate. See Section 4.1.23.

If a counterfeit Results Cartridge can be injected into the WinEDS tally processing workflow (or if a legitimate cartridge is replaced with a specially constructed Results Cartridge that emulates a succession of multiple cartridge insertions when it is loaded into WinEDS), any genuine Results

[1]MemoryPack data is protected against hardware error by a checksum scheme, but this does not protect against deliberate forgery.

Cartridges associated with the precincts on the forged cartridge will subsequently be rejected by WinEDS. If the total number of votes recorded on the counterfeit cartridge matches the actual number of votes cast at the precinct, this attack will not be detected when reconciling vote totals during the tallying process. See Section 4.1.23.

Finally, there are potential avenues for corrupt Results Cartridges to attack the WinEDS system itself. First, depending on the configuration of the Microsoft Windows system on which WinEDS runs, a corrupted Results Cartridge could be configured to load software that takes control over the entire WinEDS system when the cartridge is loaded for tallying. This attack can occur if the "Autorun" mechanism is enabled on the WinEDS system's PCMCIA interface. On the system we examined, Autorun was disabled (thus preventing the attack), but because the Windows configuration is not precisely specified by Sequoia (see Section 4.9.2), we cannot rule it out entirely. Second, there are buffer overflows in the WinEDS Results Cartridge processing (see Sections 4.1.24 and 4.1.25) that may be exploitable by a corrupted Results Cartridge.

3.1.2 Integrity of Precinct Voting Firmware and Software

The previous section noted ways in which MemoryPack or Results Cartridge data can be subverted *after* the polls close to introduce false precinct returns into the WinEDS system. In this section, we discuss in which precinct equipment can be vulnerable to tampering *before* the polls open.

In particular, the safeguards against the introduction of unauthorized or corrupt firmware[2] into Sequoia precinct voting hardware are largely ineffective. An individual with even brief access to polling station hardware can tamper with installed firmware in a way that causes votes and paper trails to be recorded incorrectly, security logs to be corrupted, or ballots to be presented to voters incorrectly. Under some configurations and conditions, corrupt firmware may be able to be spread virally from compromised hardware and may persist across more than one election.

The consequences of such attacks can be quite severe and far-reaching; they are largely difficult to detect and sometimes impossible to recover from even if they are detected.

Insight

The behavior of Insight optical ballot scanners is controlled by firmware and ballot definition data stored on the same removable MemoryPacks that hold the machines' vote tallies. The firmware and ballot definition data can be altered or tampered with by anyone with physical access to the Insight or its MemoryPack. See Sections 4.6.2, 4.6.3, 4.6.5 and 4.6.6.

Malicious firmware for an Insight can be configured to record and report votes incorrectly, reject valid ballots (requiring manual override), accept invalid ballots, print inaccurate log data, or cause the device to fail outright during an election.

Corrupt Insight firmware may be difficult to detect and may leave no telltale signs, especially if it is designed to report the correct total number of accepted ballots. However, because the Insight hardware has no software-controlled ability to mark or destroy the paper ballots themselves, a manual recount of the collect paper ballots can recover the true precinct results.

Edge

Edge touchscreen DREs are controlled by firmware and ballot definition files contained on (or loaded through) removable PCMCIA cartridges. Although there are cryptographic and other software mechanisms that are supposed to protect the firmware and ballot definition data against unauthorized tampering, these mechanisms are ineffective. We found many ways to load unauthorized firmware and election parameter data into Edges. See Sections 4.4.2, 4.4.7, 4.4.9, 4.4.14, 4.4.3, 4.4.5, 4.4.13, 4.4.12, and 4.4.15.

Malicious Edge firmware can be configured to record and report incorrect vote data on the Results Cartridge and on the internal Audit Trail. However, the nature of records produced by the

[2]For the purposes of this discussion, *firmware* is the software that controls specialized embedded devices such as Edge DREs and other precinct equipment.

Edge DRE workflow makes the consequences of firmware attacks much more difficult to detect and recover from. It also allows for several additional attack scenarios.

The Edge firmware and ballot definition controls every aspect of the user interface presented to voters, including the ballot choices themselves. (In an Insight optical scan system, on the other hand, the ballot is pre-printed on paper and is not under control of the firmware). Corrupt Edge firmware can therefore not only record incorrect votes, it can present incorrect choices to the voter (such as eliminating entire races, adding to or eliminating candidates from races, refusing to accept votes for particular candidates, or presenting the ballot in a confusing or misleading manner). Such attacks cannot be recovered from by a post-election audit recount of VVPAT or Edge Audit Trail records because it alters the ballots from which voters selected their choices in the first place.

Finally, while the VVPAT may provide a hand re-countable record in the event of post-election Results Cartridge tampering, it provides less protection against the firmware-based attacks we discuss here. Several aspects of the VVPAT behavior, including the speed at which it scrolls, the density of the print, and so on, are under the control of the Edge firmware and thus reduce the value of the VVPAT record in the presence of malicious Edge firmware. Corrupted firmware could be configured in a way that prints misleading information on the VVPAT printout or that prevents the voter from clearly seeing how their vote is recorded on the VVPAT printout[3]. See Section 4.4.18.

HAAT and Card Activator

Like the Edge, the HAAT and Card Activator have programmable firmware and are subject to malicious injection of firmware by anyone with physical access. Although neither the HAAT nor the Card Activator interact directly with stored ballots or voters, the consequences of corrupt firmware here can be quite serious. In particular, the USB- and PCMCIA-based configuration mechanisms used on the HAAT and Card Activator provide a rich vector for *viral propagation* of malicious firmware, and potentially for corruption of the Edge and WinEDS.

See Sections 4.2.4, 4.2.8, 4.2.5, 4.2.6, 4.2.9, 4.3.2, 4.3.2, 4.3.4, 4.3.5, 4.3.6, 4.3.7, 4.9.1 and 5.

3.1.3 Detection and Auditing of Corrupted Firmware

Many devices that support loadable firmware (such as the Edge, HAAT, etc.) report the current firmware version on their display. While this may accurately indicate the firmware version for properly operating Sequoia-supplied versions of firmware, the displayed version gives no indication of whether corrupted firmware has been installed.

There are no interfaces provided to election officials that can extract and verify the actual firmware present on any of the Sequoia devices. This means that there is no way for election officials to confirm or refute whether suspected devices have been subverted with corrupted firmware, and no standard procedure to capture forensic evidence if corrupted firmware is discovered.

Two audit protocols are routinely employed in California elections to detect corrupted voting machine firmware: *Parallel Testing* and *1% Recount Audits*. Unfortunately, neither protocol is likely to detect the precinct-based firmware corruption attacks against Edge DREs discussed in this report.

1% Recount Audits

In California, the results for at least 1% of precincts in each race are manually recounted against the paper ballots. This protocol is intended to detect discrepancies between the votes cast and the votes recorded and tabulated, including errors in scanning or votes mis-recorded by software. (This protocol predates electronic voting.)

[3]With DRE machines such as the Edge, the voter's check of the VVPAT record is the only confirmation that the electronic and paper records match. Relatively little is known about how carefully voters check VVPAT records when the machines are operating correctly. The software control of the VVPAT behavior makes it possible for an Edge with corrupt firmware to make it far more difficult for even a very diligent voter to confirm the accuracy of the VVPAT record.

As noted above, the recount procedure may not reliably detect Edge DREs whose results were produced by corrupted firmware, even if the precinct is subject to a 1% recount. As noted in the previous section, because the VVPAT printer is under the Edge's control, it may be possible to construct Edge firmware that prints incorrect VVPAT records (matching the incorrectly recorded electronic records) but in a way that escapes the notice of voters when they cast their ballots.

Parallel Testing

In the Parallel Testing protocol, a random sample of county voting machines are selected and not used in an actual precinct on election day. Instead, the machines are subject to testing to verify their correct behavior.

While this procedure may detect certain kinds of unreliable or corrupt firmware delivered from the vendor, it will not detect the precinct-based firmware corruption attacks discussed in this chapter. In these attacks, the corrupt firmware is loaded after the machines have been delivered to the precincts, and so escape the possibility of being subject to parallel tests.

3.1.4 Use of Tamper-Evident Seals

As with many election systems, the Sequoia System makes broad use of tamper-evident seals for protecting equipment and data. It is important to understand that *tamper-evident seals do not prevent tampering*; at best, they can detect that tampering has occurred.[4] We do not have a clear understanding of the procedures followed in practice when seals are found broken. If equipment is *not* decommissioned when seals are found broken, the election may be susceptible to tampering. If equipment *is* decommissioned when seals are found broken, then the seals can provide a quick way to disable many voting machines and potentially disenfranchise voters.

3.2 Cryptography

Many of the security features of the Sequoia software, particularly those that verify the integrity of precinct results and that safeguard against the introduction of malicious software and firmware, employ cryptography. Unfortunately, in every case we examined the cryptography provides only ineffective protection. This is a pervasive failure, with fundamental cryptographic errors throughout the software. We could not find a single instance of correctly used cryptography that successfully accomplished the security purposes for which it was apparently intended.

The cryptographic protection in the Sequoia system was effectively neutralized by several poor design decisions compounded by a range of implementation errors.

3.2.1 Hardcoded Keys

Cryptographic keys are the secret parameters used for encrypting and authenticating data. The secrecy of these keys is essential for preventing unauthorized parties from reading or counterfeiting cryptographically protected data.

Virtually all cryptographic key material in the Sequoia system is permanently **hardcoded**[5] into the software source code (and is apparently identical in all hardware shipped to different jurisdictions).

The use of hardcoded keys runs counter to widely accepted security practice. Hardcoded keys are especially likely to be exposed to adversaries and are difficult or impossible to change in the event they are compromised. They increase the necessity that a software supplier (and all of its employees with access to system code) be completely trustworthy and that the secrecy of software be protected diligently. Perhaps even more seriously, hardcoded keys can be relatively easily

[4]In some cases, seals can be broken and closed again, or the item that the seal is intended to protect can be accessed without breaking the seal. See the Red Team's report for their findings on tamper-evident seals.

[5]*Hardcoded* in this context means that the numeric values of the keys are permanently incorporated into the software, with the effect that every copy of the software uses the same keys.

extracted from the software and firmware shipped in products. Rather than hardcoding keys, the usual security practice in well-designed systems is to provide customers with the ability to generate the keys they use themselves (often based on strong random bitstream generators supplied by the vendor) and with the ability to re-generate and replace keys at intervals or as needed. No such provisions are made in the Sequoia system.

The global nature of the hardcoded keys in the Sequoia system facilitates a number of serious (and plausible) attack scenarios. The keys simply cannot be relied upon to remain secret. A malicious individual who gains temporary access to Sequoia hardware (inside California or elsewhere) can extract and copy all of the the secret cryptographic keys that protect elections in every California county that uses the Sequoia system.

Some of the hardcoded keys are used to authenticate data stored on media that pass between system components (and for which there is a risk of tampering). These keys are especially exposed, since they typically reside in the software or firmware of at least two different system components; see Figure 3.1.

Name	Cryptosystem	Component(s)	Usage
`UPDT_CRYP`	SHA	Edge HAAT, Card Activator	Validation of upgrade cartridges; also used (ineffectively) for smartcard authentication
`UPDT_ENCR`	DES	Edge	Encrypted password for loading Edge firmware upgrade file
`EE_JURD_CRYPT`	SHA	Edge	Stub for jurisdiction-specific smartcard and Results Cartridge authentication key (apparently not used; not present in WinEDS)
`EE_MACH_CRYPT`	SHA	Edge	Stub for machine-specific Results Cartridge authentication key (apparently not used; not present in WinEDS)
`AAKey`	DES (ECB mode)	Edge, HAAT, Card Activator	Encryption (and rudimentary authentication) of voter smartcard
`key` (1)	SHA	Optech 400-C	Authenticator for 400-C election coding file; implementation does not use
`key` (2)	SHA	Optech 400-C	Hashing key for passwords restricting access to reporting features; implementation does not use
`key` (3)	3-DES (ECB)	WinEDS (early voting phase), WinEDS (vote counting phase)	Textual 3-DES key used to encrypt early vote ballots; cipher used is actually single DES

Figure 3.1: Hardcoded cryptographic keys used in Sequoia system

Several places in the Edge source code suggest the availability of jurisdiction-specific and machine-specific authentication keys for authenticating Results Cartridge data. These keys are apparently intended to be stored on the Edges' internal EPROMs. Unfortunately, these keys are not actually used (and no corresponding keys exist in WinEDS). Even if they were used, the intent in the code appears to be that they would be supplied by Sequoia rather than be generated by the end-user county. This would leave many of the same vendor trust and re-keying problems even if the use of these keys were enabled.

3.2.2 Poor Algorithms and Protocols

As important as the secrecy of the keys in a cryptographic system is the soundness of the encryption algorithms and protocols that use them. Designing cipher algorithms and protocols is notoriously difficult. *Ad hoc* designs are almost always proven to be weak or insecure. In general, a cryptographic algorithm can be considered to be secure only after it has received careful analysis from a broad range of expert specialists. Standard practice in cryptography is to employ whenever possible standard algorithms and protocols that have survived the scrutiny of the academic, industry and government cryptologic communities. Fortunately, there are now standard, widely trusted algorithms and protocols for most conventional cryptographic applications, including encrypting and authenticating data.

Unfortunately, every one of the cryptographic algorithms used in the Sequoia system is either obsolete and known to be weak, is an *ad hoc* design with obvious flaws, or is used in a manner that does not provide the stated or required security properties.

The following weak or incorrectly used algorithms are relied upon in the Sequoia system:

- **SHA used as a Message Authentication Code:** The US government *Secure Hash Algorithm* is used for several purposes, primarily as a **message authentication code** (MAC) — that is, a way of establishing the origin and validity of data. The manner in which SHA is used is known not to be secure as a MAC. Also, while the current version of SHA is widely trusted, the Sequoia system uses an older version that is now known to have certain flaws. Finally, the implementation has a flaw in which the secret keys are not actually used when authenticating the data, thereby eliminating the (already reduced) protection that the function might have provided (see Section 3.2.3).

- **CRC used as a MAC:** *Cyclic Redundancy Check (CRC)* algorithms were apparently used as message authentication codes on Results Cartridges produced by older versions of the Edge firmware. Such algorithms are appropriate only for detecting non-malicious errors; they provide no defense against intentional data tampering. Although these CRCs are not produced by the current Edge software, WinEDS retains backward compatibility with it, and so an attacker who forges a Results Cartridge can force the use of the insecure algorithm.

- **DES used in ECB mode:** Voter smartcard data and early voting ballots are encrypted with the now-obsolete *Data Encryption Standard (DES)* algorithm (under hardcoded keys; see Section 3.2.1). Because of its relatively small key length, the more than 30 year old DES algorithm is no longer considered adequate against a well-funded adversary with modern computers. (In one instance, the more secure *3-DES* algorithm is mentioned in comments, but the implementation actually uses only DES.) Worse, the manner in which DES is used, called *ECB mode,* is vulnerable to "cut and paste" attacks that do not require significant computing resources to carry out.

3.2.3 Flawed Cryptographic Implementation

Much of the cryptography is implemented incorrectly, in a way that effectively eliminates whatever protection the static keys and poor choices of algorithms might have provided. This is a pervasive problem throughout the software.

The most serious incorrect implementation issues concern the authentication of Edge Results Cartridges. The SHA authentication code implementation appears to ignore the secret key parameter, such that the key is not actually used when producing or verifying the cartridge. Furthermore, under some circumstances the authenticator is not actually verified by WinEDS. The effect is that an attacker does not need to discover the authentication key in order to produce altered or counterfeit Edge Results Cartridges that will be accepted as valid by WinEDS.

Most of the cryptographic functions appear to have been implemented from scratch[6] and

[6]One exception is the DES code, which was apparently derived from a public domain implementation written by Phil Karn in the 1980's. Unfortunately, although the public domain implementation on which the code is based is sound, it is used incorrectly for 3-DES.

never tested against widely available standard reference implementations or test vectors for the algorithms.

These implementation problems suggest a general lack of cryptographic module testing; see Section 3.4.

3.2.4 Attacks Facilitated by Bad Cryptography

The weaknesses in the Sequoia cryptography architecture and implementation gives rise to many security issues. Many of the most serious potential attacks we identified against the system are enabled or exacerbated by cryptographic flaws.

Weaknesses and attacks involving improperly authenticated Edge results cartridges are discussed in Sections 4.1.23, 4.1.24, 4.1.25, and 4.4.5.

Attacks against the Edge firmware that involve cryptography are discussed in Sections 4.4.2, 4.4.3, 4.4.5, 4.4.9 and 4.4.7.

Cryptographic attacks against ballot secrecy are discussed in Sections 4.4.8 and 4.4.17.

Cryptographic attacks against the Optech 400C are discussed in Sections 4.8.2, 4.8.3, 4.8.4, 4.8.5 and 4.8.6.

3.3 Access Control

Access control consists of restricting access to sensitive data and capabilities and managing the set of access rights possessed by individual users or programs. In an election system, different kinds of users require different levels of access. For example, an eligible voter needs some access in order to cast a vote, but must be denied access to cast more than one. Poll workers need access to perform duties at their polling place, but should not have access to alter the database that tallies the official results. Some other examples of capabilities requiring access control are:

- Changing voting machine configurations.
- Modifying the ballot definition.
- Modifying software.
- Changing the internal state of a machine, such as its date, time, or protective counter.
- Reading private information.

The Sequoia system provides various mechanisms for restricting access to its system components. Unfortunately, several of these mechanisms have serious flaws that allow them to be bypassed.

3.3.1 Access Control in Central Elections Software

Of the components we reviewed, WinEDS has the most extensive and complex access control system. The WinEDS access control system is based on user accounts and roles. Each user account has its own password and can be given one or more roles. Each role can be assigned a number of *access rights* that correspond to commands and buttons in the WinEDS user interface. A user with more than one role receives the combined access rights for each role. If a user lacks the right to access a particular command, the command will be disabled in the user interface.

The access rights that can be assigned to roles are organized in a table. Each access right provides a particular action on a particular component. The Role Maintenance window in WinEDS lists 111 components, with up to 6 checkboxes next to each one, in columns labeled New, Edit, Remove, Query, Admin, and All. We counted a total of 615 checkboxes in this window. For example, one of the components is named Precinct; any user with the New Precinct right can add new precinct records to the database. As another example, the Role Maintenance window is itself subject to access control; there is a component named Role, and any user with the Edit Role right can change the assignment of rights to roles.

Thick Client Security Architecture

Every WinEDS user account corresponds to an account in the MS SQL database with database administrator (DBA) privileges. When a user enters his username and password into WinEDS, the application logs into the database using the provided username and a password derived from the password entered by the user (see Section 4.1.1). If the login is successful, WinEDS queries a table in the database to discern the permissions for that user. Application features that fall outside the granted permissions are disabled by the client.

Hence, WinEDS constitutes a **thick client** in which access control is provided by the application running on the client's machine. This is in stark contrast to common practice. More often, users operate **thin clients** in which access control is imposed by the back-end system. For example, web browsers function as thin clients when connected to online banking sites. Web browsers do not control which parts of a banking site the user has permissions to use; rather, access restrictions are enforced by a system under the bank's control.

Because the Sequoia design enforces access control only in the client, the extensive access control system in WinEDS can be completely circumvented by communicating directly with the database (see Section 4.1.1). Further weaknesses in WinEDS's handling of passwords (see Sections 4.1.2, 4.1.5, 4.1.6, 4.1.7, and 4.1.8) facilitate the discovery of database passwords, any one of which is sufficient to log in to the database directly and gain control of all its contents.

Data Wizard

WinEDS includes a feature called the Data Wizard, described in the reference manual as "a utility used to import data into WinEDS and export WinEDS data from in [*sic*] WinEDS tables." The manual calls the Data Wizard an "advanced feature" and warns that it "allows access at the table level." The Data Wizard appears to list all the database tables used by WinEDS as tables available for exporting and importing. If it worked as intended it would give users the most wide-ranging access of all, as it could be used to modify any of the election information or any user's access rights (since access rights are also stored in the database). But the Data Wizard feature is not subject to access control. Any WinEDS user, even a user with all 615 access rights turned off, can use the Data Wizard (see Sections 4.1.10 and 4.1.11). The Data Wizard is incorrectly implemented (see Section 4.1.12); it fails to import data. However, it can erase entire tables.

Platform Access Control

WinEDS and WinETP clients run on Microsoft Windows 98, Me, 2000, or XP. These operating systems provide mechanisms for user access control. For example, an administrator account is capable of installing new programs, but a guest account is not. However, we found the platform-based access control on the WinEDS client and the Optech 400-C provided to the Red Team by Sequoia to be overly permissive.

The Optech 400C system was set up with only one user account, WinETP, with administrator access and no password. On the laptop running the WinEDS client and the SQL server, the user Tally running WinEDS had sufficient permissions to install additional software [7]. These privileges allow any WinEDS or WinETP user logged on to the respective system to have full control over these machines, regardless of their privileges in WinEDS or WinETP.

The Optech 400-C Security Specification (page 3-2) notes that "The vendor shall provide the jurisdiction with a list of all software needed by the election management software. All other Third-party software must NOT be installed, that has not been previously approved for use by authorized personnel, to prevent the introduction of software that may damage the Optech 400-C." We would interpret this to mean that only the minimum set of software needed for election purposes should be installed on the machine, which is a good security practice. However, the Optech 400-C machine as configured and provided by Sequoia came with programs such as Google Desktop Search,

[7] The WinEDS laptop was equipped with a CD-ROM driven and the personal computer in the Optech 400-C — though stowed in a locked cabinet — can accessed with little effort (see the Red Team Report). This raises the possibility of introducing malicious software on these systems.

Internet Explorer, Windows Movie Maker, MSN Gaming Zone, NetMeeting, Outlook Express, Minesweeper, and 3-D Pinball.

3.3.2 Access Control in Polling Station Equipment

The HAAT and the Card Activator are portable devices and may find themselves in the hands of many people. However, unregulated access to either device significantly weakens the security of the Sequoia System (see Sections 4.2 and 4.3). To prevent unauthorized access, the HAAT and Card Activator have both software and hardware access control measures.

Software-Based Access Control

The HAAT and Card Activator use numerical passwords to control access to different functions. However, critical flaws in how passwords are handled negate their effectiveness as access control measures:

- Passwords do not protect the right features.
  - In the Card Activator, passwords are only used in two places: during the boot sequence, and in Early Voting. Particularly, a password is *never* required to replace the firmware with an arbitrary file on a generic PCMCIA cartridge (see Section 4.3.2). The file to be copied must have a particular name, but if the firmware update feature is accidently invoked, a message like "ABC123.EXE not found" is displayed, revealing the necessary filename (see Section 4.3.6).
  - The HAAT protects more features including changing the time and preparing, upgrading or reseting the HAAT. However, the backup feature is not protected by a password. This feature copies all the configuration and log files onto a generic USB stick. Among the files copied is an XML file with the HAAT passwords listed in plain text (see Section 4.2.7).
- The default behavior is insecure.
  - The Card Activator proceeds without requiring any passwords if the password files are empty, and WinEDS appears to always configure it with an empty password file (see Section 4.3.2).
  - When the HAAT is not prepared for an election all features are protected by a single permanently hardcoded password. Sequoia's *HAAT Security Specification* asserts this password is the same on all HAAT units. This password may be easily learned by an adversary, and can be used to modify the firmware of the HAAT (see Section 4.2.2).

The consequence of the above is that the software on the HAAT or Card Activator will not prevent someone with minimal access from replacing the firmware of the device. Such replacement can be performed without any need to physically access its internals. In other words, tamper-evident seals are insufficient to protect the integrity of these devices.

Hardware-Based Access Control

Neither the HAAT nor Card Activator has any locking mechanism or appreciable physical access control. Both are secured by screws, and make no evident attempt at detecting the opening of the case, for instance with sensors or alarms. Therefore, tamper-evidence relies entirely on the correct choice and use of seals over the screws. Seals do not *prevent* access, and would not, for example, prevent someone from learning Sequoia's proprietary secret keys and fabricating counterfeit Voter Cards (see Section 4.5.2). Additionally, if the hardware is accessed, all bets are off on software-based access control from that point on: the software can be modified arbitrarily.

3.4 Defensive Software Engineering

Complex software systems are especially susceptible to bugs or errors that cause the system to behave in an unintended manner. When one of these bugs is encountered, the effect can vary from causing minor instability to enabling devastating security exploits.

Defensive software engineering, as used in this report refers to software development practices that attempt to prevent software bugs likely to lead to security problems *before* they are incorporated into the finished product. In security-sensitive contexts such as elections, each software component should be designed and written defensively with conservative assumptions about the rest of the election system, in order to limit the damage that can be caused through failures or weaknesses in hardware components, software components, and human procedures. Relying on everything else to be perfect makes a component fragile and vulnerable.

During our analysis of the Sequoia source code, we saw artificats consistent with several potentially dangerous development practices. This section reviews these practices, and their relationship to defensive software engineering.

3.4.1 Complexity

One of the fundamental defensive software engineering practices is to keep the design and source code as simple and small as possible. The very act of programming inevitably introduces software errors into the code. Limiting the complexity of the code allows frequent and thorough analysis to find and correct these errors.

Code Size

The Sequoia system consists of approximately 800,000 lines of code in 10 different languages, including 4 different assembly languages (see Figure 3.2). In addition, the system makes use of at least 6 interpreted languages (see Section 2.1.11). The sheer size of the Sequoia code base and the breadth of languages make understanding and analysis difficult in any time frame.

Programming Languages

The type and number of programming languages used contributed greatly to the complexity of analysis required, and limiting the amount of static analysis we were able to perform.

Approximately 55% of the Sequoia source code is written in C, C++, or assembly — all of which are not memory-safe languages. These three programming languages are known to be prone to several common types of security vulnerabilities, including buffer overruns, format string vulnerabilities, and integer overflows (see Section 3.4.3 for more on these vulnerabilities). We found instances of all of those vulnerabilities.

Many security engineers recommend use of memory-safe, type-safe programming languages, because those languages have inherent resistance to several of the most common types of security vulnerabilities. For instance, buffer overrun vulnerabilities have been called the #1 cause of security holes in software for many years in a row (although they may have recently dropped to the #2 or #3 cause).

Additionally, the WinEDS GUI was created using PowerBuilder — a development system for creating database applications. Many of the PowerBuilder source files are machine-generated and made only to be understood by the PowerBuilder application, which we were not provided. This made analysis of the PowerBuilder code considerably more difficult.

Legacy Code

The incorporation of legacy code, while not a bad practice in and of itself, adds to the complexity and heterogeneity of the code base. Some code components licensed and used by Sequoia date back to the mid-1980s, and some revision comments date back to 1996, indicating the code base has been growing for more than 10 years.

Component	Language	Code Only	Code and Comments
WinEDS 3.1	C	1 038	1 594
	C++	121 640	228 765
	PowerBuilder	230 027	355 502
	SQL	86 222	114 249
	Visual Basic	10 260	16 772
Edge (AVC Edge 5.0.24)	C	124 043	212 731
	x86 assembly	99 521	124 657
VeriVote (VVPAT 4.3)	PIC assembly	245	353
ADA Audio Board 5.0	C	1 328	1 956
Card Activator (Card Activator 5.0)	C	8 907	14 238
HAAT 50 (HAAT 1.0.69L)	8051 assembly	5 368	5 891
	C	535	963
	C++	2 886	5 640
	C#	38 648	120 246
Insight (HPX 1.42, APX 2.10)	Z80 assembly	24 405	46 452
MemoryPack Receiver (MPR 2.15)	Z80 assembly	5 679	9 714
Optech 400-C (WinETP 1.12.4)	C	561	1 007
	C++	45 361	83 229
	x86 assembly	273	612
Total:		806 947	1 344 571

Figure 3.2: Lines of code and languages included in the Sequoia source code review. Blank lines were not included in the count.

Configurability

The high configurability of some of these systems also adds to their complexity. Some amount of configurability is necessary to accommodate the rapidly changing landscape of voting and certification requirements. However, each possible setting quickly explodes the number of possible configurations any component may be in at any given time. For example, the Edge machine has 76 total configuration settings. The large number of possible combinations makes testing and analysis of every configuration infeasible.

3.4.2 Interpreted Code

Designing and using special-purpose interpreted languages is a technique that can improve or impair the robustness, security, and auditability of a system, depending on the design of the language and the way it is used. We are aware of the prohibition in Section 4.2.2 of the 2002 Voting System Standards (emphasis ours):

> Self-modifying, dynamically loaded, or interpreted code is prohibited, except under the security provisions outlined in section 6.4.e. **This prohibition is to ensure that the software tested and approved during the qualification process remains unchanged and retains its integrity.** External modification of code during execution shall be prohibited.

However, the mere fact that interpreted code is used does not, in and of itself, imply anything good or bad about its reliability. The text of the requirement makes clear that the intent is to support the approval process and the integrity of the approved software. In our opinion, what actually matters is **whether the code is reviewed**, **whether the code can change after it is reviewed**, and **whether the code can be effectively reviewed**, not whether the code is interpreted.

From a reviewer's perspective, a well-designed or well-chosen interpreted language can make software much easier to understand, leading to a more effective review; using a complex or inappropriate language can obscure the code and make review much more difficult. From a security perspective, it is important to consider **how powerful these languages are**. Replacing a simple component with a highly flexible language that has broad access can increase the risk of error or vulnerability; on the other hand, using a language with only a limited set of capabilities or a language in which some unsafe operations are inexpressible can significantly reduce risk.

The following paragraphs evaluate the six types of interpreted code we found in the Sequoia system (update scripts, simulation scripts, pack scripts, B-Code, E-Code, R-Code) along the aforementioned criteria (above, in bold).

Exposure of Interpreted Code to Review

Update scripts are provided by Sequoia after system deployment. Simulation scripts are intended to be written by users of WinEDS. The remaining kinds of interpreted code (pack scripts, B-Code, E-Code, and R-Code) are generated by the software itself. Consequently, none of this interpreted code is reviewed as part of the normal certification process. Since none of this interpreted code is reviewed, it is pointless to consider whether it can be modified after review.

The lack of review is cause for concern, especially since some of the interpreted code has the capability to replace firmware or affect election results.

Effect on System Complexity and Reviewability

Update scripts and simulation scripts are straightforward sequences of textual commands; their execution is easy to understand.

Pack scripts are also straightforward sequences of textual commands. However, they are generated programmatically and then immediately executed. Their purpose is to send data to a MemoryPack, which would be achieved more simply by just sending the data. Instead, WinEDS constructs a sequence of instructions describing what to send, writes the instructions out to a temporary file, then reads them back from the temporary file, parses the meaning of the instructions, and carries them out — by sending the data. The extra code for generating the pack script, parsing the script, and carrying it out is unnecessary; it needlessly complicates the procedure and any attempt to analyze its correctness.

B-Code and R-Code are more difficult to decipher because each instruction is a parameterized data structure rather than a single textual command. We did not examine these languages closely enough to determine whether their functionality could be achieved without the need for invented languages.

E-Code is the most complex interpreted language in the Sequoia system, with approximately the full power of a machine instruction set. For example, it is possible to carry out arbitrary arithmetic in E-Code and control program flow using the results of that arithmetic. This level of complexity is almost certainly in excess of what is needed for counting votes. For instance, note that the Insight and the Optech 400-C scan the same kinds of ballots, but whereas the Insight uses B-Code to control ballot counting, the Optech 400-C uses the far more complicated E-code for the same purpose.

B-Code is generated by a corresponding B-Code compiler in WinEDS, and R-Code and E-Code are generated by a corresponding R-Code compiler and E-Code compiler in WinETP. Determining whether the interpreted code in these languages functions correctly requires analysis of both the interpreter and the compiler. Attempting to establish the correctness of just the E-Code compiler and virtual machine, for example, would have far exceeded the time we had available to perform this review. Consequently, we are unable to conclude anything from the source code alone about whether the 400-C counts ballots correctly. We did, however, notice that the E-Code interpreter is not written defensively, which makes it a possible avenue for attack (see Section 4.8.8).

Power and Capabilities

Despite their simplicity, update scripts are very powerful in that they can rearrange or replace any files on the internal disk of an Edge, and can thereby modify all the software that runs on the Edge.

Pack scripts have the power to erase and write data to a MemoryPack, though they only have limited capability to determine the data content since a pack script can only name which files are to be transferred to the MemoryPack.

Simulation scripts have the power to control the Edge user interface, but only in simulation mode, which is explicitly initiated by the operator.

B-Code has the power to determine election results as well as Insight configuration parameters. An incorrect B-Code program can miscount votes, misdirect ballots, or alter Insight settings.

It would appear that E-Code is intended to have only the power to calculate election results. However, because of a weakness in the E-Code interpreter, E-Code programs may have the ability to take control of WinETP and the Optech 400-C (see Section 4.8.8).

R-Code has the power to format election reports. An incorrect R-Code program can generate incorrect or misleading reports.

3.4.3 Input Validation

When implementing election software — or any critical software system that is required to be reliable and robust against failure — it is important to detect invalid input and respond accordingly. Input validation is even more critical when the software must defend against determined attempts to subvert the outcome. **Input validation** is the act of checking whether data used in a program complies with an expected format, is consistent, and thus is unlikely to cause the program to behave in ways unintended by the developer. For instance, when reading an integer from a file, one should check that the number is neither too big nor too small, if a number outside the expected range might cause an error in a later part of the program. Furthermore, data passed from one part of a software system to another should be validated by the receiving component so that if the sending component is behaving erratically or has been compromised by an attacker, the receiving system detects the abnormality. The receiving component can then respond to it gracefully (see Section 3.4.5) and prevent the error or attack from spreading.

Input validation differs from data integrity protection in that it does not attempt to detect whether a given input coming from an authorized source or not. Validation of input is a necessary practice to build reliable and secure systems, independent of cryptographic integrity mechanisms such as cryptographic hashes, signatures, or message authentication codes. We now briefly describe several types of errors that can be caused by ineffective input validation. This is not intended to be an exhaustive list.

Buffer overflows are common software errors than frequently lead to serious vulnerabilities in software systems. A buffer overflow happens when more data is written to a buffer in memory than space has been reserved for the buffer, causing data outside of the buffer to be overwritten. If an attacker can control the data written to the buffer, she can often engineer an exploit string that, when written to the buffer, overwrites memory outside the buffer in a way that will execute code of her choice on the victim's computer.

Integer overflows are a related software error. Most numeric variables in computer programs can only represent numbers in a fixed range. If an arithmetic operation such as addition yields a result that exceeds this range, then this is considered an arithmetic overflow (similarly, if the result of the operation is too small, we speak of an underflow). For instance, if a variable x can only hold integers from 0 to 99, but as a result of an arithmetic operation is assigned the value 102, then, typically, the number wraps around and causes the erroneous value 2 to be stored in x. Some common ranges for integer variables used in software systems are -32768 to $+32767$ (for two-byte short integers) and $-2,147,483,648$ to $+2,147,483,647$ (for four-byte integers). Undetected integer overflows or underflows are likely to cause the software to crash or behave in unexpected ways, and often trigger other errors, such as buffer overflows.

Another common flaw is insufficient validation of **format strings**. Format strings are character strings whose main purpose is to tell the computer how to format or interpret data as text, for instance when writing to a file. Usually, format strings are interpreted by library code supplied by the operating system. With many libraries, format strings can be used to read and write memory out of bounds, and thus may cause the executing program to behave in unexpected ways, crash, or even execute code specified by the attacker. Therefore, format strings from untrusted sources should preferably not be used, or at a minimum be sanitized.

Due to time constraints, we concentrated our search for buffer overflows, integer overflows, and format string vulnerabilities on three components: WinEDS, WinETP, and the Edge. We chose these components because they are written largely in programming languages susceptible to these problems and because the other systems have vulnerabilities that appear to provide an easier avenue for unauthorized persons to gain control.

In all three of these components, we found that weak input validation was widespread. We discovered multiple vulnerabilities that are likely to provide significant assistance to unauthorized persons in gaining control over these systems to alter the outcome of an election or disrupt an ongoing or future election. We also found that format strings handled in the back-end systems, WinEDS and WinETP, were not sanitized despite using dangerous library functions. Format strings are handled somewhat better in the Edge code, which uses custom implementations that seem not implement all dangerous format modifiers.

The poor input validation in many parts of the code we examined is a serious concern, as this indicates that this fundamental defense against common, and potentially very serious, attacks is weak. Weak input validation poses an increased risk for attacks on one component to affect another, not dissimilar to flimsy compartment walls in a submarine. Furthermore, even in the complete absence of malicious intent, the weak input validation decreases the reliability of the voting system, and potentially increases the effect of any component malfunctions.

We found multiple instances in the design, setup and implementation of the WinEDS client where data from removable media and the database is assumed to be valid.

- *USB sticks:* The Sequoia system uses USB sticks to program HAAT devices for an election. The WinEDS client appears not to check whether a USB stick plugged into the computer running WinEDS is authorized. This is particularly dangerous when a USB stick serves to program multiple HAAT devices or is reused across elections. Under standard configurations of Microsoft Windows, the Autorun feature will automatically run programs on some USB sticks. If such a stick is used in a HAAT that has been compromised by an attacker (see Section 4.2.8), or an attacker can provide a maliciously modified USB stick in place of a legitimate one, the attacker could surreptitiously take complete control over the WinEDS client. From there, it could spread malicious software to all subsequently programmed HAATs and to the Edge, again through weak input validation (see Section 4.4.13).

- *Results Cartridges:* We discovered several buffer overflows in the WinEDS code that reads from Results Cartridges (see Sections 4.1.22, 4.1.24, and 4.1.25), which appear to have the potential to give full control over WinEDS to an attacker who can cause a specially crafted Results Cartridge to be processed. One of these bugs that would appear to be particularly easy to exploit (see Section 4.1.22) seems to be dead code.

- *MemoryPacks:* WinEDS seems not to check whether vote counts stored on MemoryPacks received from the MPR are consistent with the number of voters (4.1.28), so an erroneous MemoryPack will corrupt the final tally instead of being detected.

- *Database:* Furthermore, WinEDS does not sufficiently validate data from the database which makes clients more vulnerable to a compromised database server and any compromised or malfunctioning WinEDS clients. For instance, WinEDS contains a format string vulnerability when reading from the database (see 4.1.13). Also vote totals read from the database are not checked to be non-negative, which can cause negative totals to be printed in reports.

Furthermore, WinEDS contains a host of buffer overflows, which is an indication of fragile coding practices (see Section 4.1.19). These are not necessarily exploitable, but likely to contribute to the unreliability of WinEDS and further reduce our confidence in the program.

Similar to WinEDS, input validation on the Edge is weak and gives rise to dangerous vulnerabilities. For instance, font files on the Results Cartridge are not sufficiently checked for invalid contents. This has the potential to allow anyone who can rewrite a cartridge to take complete control of any Edge that the cartridge is inserted into (see Section 4.4.13). In another case, a file on the Results Cartridge specifies names of files that are copied from the cartridge to the internal drive. The file is intended to contain filenames, not relative paths, but no check is made to verify this. By specifying relative paths, an attacker can overwrite the Edge's firmware with her own executable (see Section 4.4.12). When reading candidate endorsements from the Results Cartridge, the Edge does not check the file it reads for data consistency, and can be made to run in an infinite loop during report generation (see Section 4.4.10). Another example representative of the insufficient validation performed in the Edge software is an unsafe memory allocation routine (see Section 4.4.16) that is used pervasively.

Due to time constraints, we were only able to spend a relatively short examining the source code for WinETP, the main application running on the Optech 400C. Still, we were able to quickly identify that the interpreters for the languages used for ballot counting and report generation, respectively, do not include sufficient memory protection. In particular, we found scripts in the E-code language are trusted to write to unchecked memory addresses, which is likely to provide an attack vector powerful enough to compromise the machine.

3.4.4 Implicit Assumptions

Another way to view the lack of input validation is as an implicit assumption of trust: If one component processes data from a second component without validating it, then the second component is trusted to be working correctly and have not been subject to an attack.

Furthermore, we found that the use of non-trivial string values and numbers is pervasive in all components with the possible exception of the HAAT. For instance, the Card Activator contains hardcoded shell commands that explicitly contain file names. Should the file name ever change for reasons completely unrelated to the place in the source code where the shell command is executed, the string specifying the shell command would have to be changed. Even when data like file names and paths are collected in one place, there is often no attempt to minimize dependencies. For example, in one instance, a set of constants defining paths to files in the same directory each hardcoded the path to the directory. Should the directory name ever be changed, then all constant definitions would have to be changed consistently or errors would result.

Introducing implicit — and often undocumented — dependencies like these in a complex software system is likely to reduce reliability and introduce bugs as the software evolves.

3.4.5 Error and Exception Handling

Error and exception handling is essential to the reliability of a system, and to the recoverability of the system when problems occur. While examining the Sequoia system, we discovered numerous cases where errors are handled ungracefully. Specifically, we encountered situations where errors were silently ignored or caused WinEDS to crash. This creates an expectation of unreliability, as discussed in Section 3.4.6.

Some SQL stored procedures used by WinEDS fail to properly check return codes for errors (see Section 4.1.18). Additionally, there were some situations in which a problem caused several error messages, but then the operation would be reported as completing successfully. For example, WinEDS would list the result of processing a cartridge as "Success" immediately after complaining of errors on the cartridge. This leaves the user with no assurance that the operation was actually successful or that the database was left in a consistent state.

Critical systems such as voting software should be written to prevent and detect errors so that an adequate response can be taken. However, the weak input validation (as described in Section 3.4.3) is often insufficient in preventing errors produced in one component from cascading to another. This silent cascading of errors makes problems harder to track, and may cause the software to crash or fail in unexpected and unsafe ways — possibly resulting in irrecoverable data loss or corruption.

3.4.6 System Unreliability

Since we did not have build environments or consistent access to equipment, we did not specifically test the reliability of Sequoia systems. However, we did have access to an executable of WinEDS running on Windows XP Professional SP2.

We found using WinEDS, even in a normal context, unreliable. Performing normal actions would cause WinEDS to crash, lock up, or corrupt the system at unpredictable times. For example, Sections 4.1.3, 4.1.4 and 4.1.9 give just a few examples where normal actions affected system reliability. The weaknesses in Sections 4.1.11, 4.1.13, 4.1.18, 4.1.19, 4.1.22, 4.1.24, 4.1.25, and 4.1.26 also have consequences for system reliability. We also experienced several other cases of unexplained system crashes while using the WinEDS system which we do not document in this report.

Expectation of Unreliability

Creating an expectation of unreliability gives an advantage to attackers. It is generally difficult to write attack code in a way that avoids triggering errors or causing a system crash. In a reliable system, these errors and crashes may raise suspicion and alert administrators to take a closer look at the system. If users have an expectation of unreliability, an attacker does not need to be concerned about writing exploit code with sophisticated cleanup routines.

Importance of Reliability

Conducting an election is both time-intensive and time-critical. It takes considerable time to prepare for an election, and considerable time to tally and verify the results — including any necessary hand recounts of the paper trail. After election day, election administrators only have 28 days to announce the official results. For some counties, this involves working 18-hour days to complete the tally on time.

In addition, elections are data-intensive and data-critical. Votes need to be accurately stored and counted for *every* voter, and protected against both accidental corruption and intentional manipulation. Every time a system crashes, there is a possibility that the data has become corrupted or inconsistent. This causes a lack of confidence in the election results, and can be costly and time consuming to recover from.

This is not a process which can accommodate an unreliable election management system.

3.4.7 Documentation Inconsistency

We had access to numerous documents from Sequoia for each of the system components, which we used to aid our understanding of the system. Due to our limited scope and time constraints, we did not exhaustively examine the documentation and source code comments for correctness and completeness. However, we did encounter several discrepancies between the documentation, comments, and source code. Specifically, we found instances where the documentation and comments were insufficient, inconsistent, and that did not represent the actual functionality of the source code.

Insufficient Documentation

On several occasions, we could not find documentation for functionality found in the source code. We cannot conclusively say that there is no documentation on these features, only that we could not find any such documentation.

One example comes from the equipment provided to the UCSB-based Red Team. The Red Team was provided a single laptop for both the WinEDS client and SQL database server. The laptop was equipped with a built in PCMCIA slot and a special extender to accommodate the custom jacket placed on cartridges. We were unable to find in the documentation any specification for the PCMCIA reader and necessary extender.

We also encountered incomplete documentation concerning cartridges. WinEDS and the Edge define 10 different cartridge types in the source code (see Section 2.1.6). Both systems define a Technician Cartridge and Internal Audit Trail Cartridge type, and the Edge defines an additional Audio Cartridge type. We were unable to determine from the comments and documentation what function these cartridges serve.

Inconsistency Across Systems

The Edge systems require WinEDS for configuration and tallying of votes. However, we found examples of functionality documented in the Edge, but not in WinEDS.

For example, the Edge documentation discusses how to use Consolidation Cartridges, Simulation Cartridges, and Audit Trail Transfer Cartridges. However, the WinEDS documentation does not describe how to create these cartridges.[9]

In another example, the Master Ballot Cartridge (see Section 2.1.6) feature is documented and functional[10]on the Edge, but does not show up in the WinEDS documentation and does not seem to be implemented in the WinEDS source code (see Section 3.4.8 for more discussion).

This inconsistency across systems seems indicative that the systems are at different stages of development, with potentially untested and undocumented features.

Examples of Substantive Inaccuracy

One example of substantive inaccuracy involve the mentions of secure cryptography in the documentation. For example, the AVC Edge 5.0 System Overview claims:

> Calculation or validation of these signatures can only be done with prior knowledge of the seed value…

However, as discussed in Section 4.4.5, these seed values, or keys, are not used by the actual functions. Also, the WinEDS Software Specification claims that the system uses Triple DES encryption for the early vote collection process, when in fact the implementation in source is only single DES. Section 3.2 discusses these problems in more detail, and how the expectation of secure cryptography practice does not match the actual implementation.

The Edge documentation also makes a broad claim (in the AVC Edge 5.0 Security Overview) that malicious software cannot be introduced into the Edge. See Sections 4.4.2, 4.4.3, 4.4.12, 4.4.13, and 4.4.15 for indications otherwise.

The Optech 400-C Security Specification (page 3-1) claims that "The Optech 400-C does not have source code, assembler, compiler or linker files resident on the computer." However, the Optech 400-C software (WinETP) itself contains two compilers (one for R-Code and one for E-Code).

We also found instances of comments in the source code that do not reflect what functions actually do. For example, WinEDS has a function that appends a fixed suffix to passwords. The function's comments claim that this concatenated string is actually an "encrypted string" — where this is clearly not the case (see the observations in Section 4.1.1 for more detail).

These discrepancies in the documentation suggest a misunderstanding of basic system functionality, which can lead to dangerous and inaccurate security decisions (by both the vendor and the end-user).

3.4.8 Unused Code

Unused features and dead code traditionally receive less scrutiny, and may provide additional avenues of attack in a system. Removing unnecessary code from the system removes such possible avenues of attack, as well as reducing the code and configuration complexity of the system (see Section 3.4.1).

[9]See the Sequoia Document Review Team's report for more detail on these undocumented features.

[10]We were able to confirm that the Edge machine provided to the Red Team does recognize, accept, and load Master Ballot Cartridges.

For example, the Card Activator contains an undocumented feature to switch its mode of operation for use in Basingstoke, UK. This happens if a particular set of files is present on the Preparation Cartridge (a standard PCMCIA cartridge). Even though WinEDS will not deliberately create these files for an election in California, it is possible to manually create these files and cause the Card Activator to run in "Basingstoke mode." The consequence is that potentially untested code paths (and any vulnerabilities on those paths) will be executed[11] (see Section 4.3.7). A similar feature exists on the Edge as the "customer specific" flag. If set to "UK/England" the Edge performs different validation checks of the Results Cartridge. Such features are unnecessary to include in a U. S. build of the source code.

Discrepancies between the Edge and WinEDS documentation seem to indicate that there are extra features in the Edge that are currently unused and unsupported by WinEDS (see Section 3.4.7). For example, the Edge may be vulnerable to a buffer overrun when handling of Master Ballot Cartridges (see Section 4.4.14). Since this cartridge type is not supported by WinEDS, this functionality in the Edge would seem to serve no purpose other than providing an avenue for attack.

The Edge also includes examples of dead code, identifiable by the lack of any "callers" listed in a function's comments. For example, the function for running script files is never called and has no callers in its comments (see Section 4.4.20). The Card Activator also includes an example of inert, if not dead, code. Throughout the Card Activator source code, events appear to be logged through a special function. However, the function just returns `true` without doing or logging anything (see Section 4.3.9). This gives the false impression that events are tracked in the Card Activator.

WinEDS contains another example of possibly inert and dangerous code. During preferential tallying, malicious data on a Results Cartridge can cause an integer overflow in WinEDS, which in turn may enable an attacker to execute arbitrary code on the WinEDS client. However, a boolean flag seems to prevent the code from ever executing. For more details, see Section 4.1.22.

3.4.9 Software Testing

Due to the scope of this project, we did not exhaustively examine the test reports provided by Sequoia or the ITAs. However, the nature and volume of security-related software bugs we found are strongly suggestive of insufficient software testing, especially *unit* and *regression* testing. Unit testing looks at smallest testable software units, and tests them separately for proper functionality and adequate error handling. Regression testing ensures that as the software changes and grows, previously working components still function properly and previously fixed bugs have not been reintroduced.

The test specifications we were provided seem to focus on getting the overall correct result given correct input. However, exploiting the behavior of a system when given incorrect or malicious input is the primary attack vector used by malicious users. Thus, testing with *incorrect* input is essential. Even in the complete absence of malicious intent, insufficient error detection and response is likely to exacerbate the consequences of software bugs and will be detrimental to the system's reliability and usability.

The materials provide for our review did not include internal testing artifacts documenting unit or regression testing. However, the quality of the software suggests that any such testing was likely insufficient in breadth and depth. We encountered software bugs that indicate a broad quality assurance failure on three levels.

- **Advertised functionality that does not work:** Several cryptographic routines in different parts of the system purport to use key materials, but in fact do not. Had any of these routines been tested even once against standard test vectors or a reference implementation, the problem would have been discovered.

  The Change Password feature does not work properly for the system administrator (Section 4.1.9). The import function of the Data Wizard tool in WinEDS does not work

[11]Granted, the Card Activator has much more severe problems since the *entire* code can be replaced with an arbitrary file.

(Section 4.1.12). Due to clear errors we found in the source code, we believe these two functions could not possibly work properly under any circumstances. Changing a username locks the user out of the system (Section 4.1.4). The PIN entry function in the Card Activator seems clearly intended to stop allowing further attempts after two failed attempts to enter the PIN code; but due to a logic error it only stops beeping for incorrect PIN codes after the second attempt (Section 4.3.3). Had any of these features been tested even once, each problem would have been discovered.

- **Poor handling of non-malicious errors:** The system does not handle errors introduced by users or malfunctioning components well (see Section 3.4.5). For instance, we observed that a single invalid database entry can cause many error messages in WinEDS. Many times, WinEDS does not check whether reading, writing,[*2] copying[*3] or deleting[*3] a file actually worked. We found an integer-to-string conversion routine[*4] that causes a small buffer overrun when passed very small negative numbers. All these errors would have been caught in thorough and well-written unit tests. See also Section 3.4.3.

- **Vulnerabilities to malicious input:** Some examples of weak defenses against malicious input are detailed in Sections 4.1.13, 4.1.22, 4.1.23, 4.1.24, 4.4.14, 4.4.13, 4.4.12, 4.4.15, and Section 3.4.3.

The prevalence and severity of problems present in the Sequoia system suggest that the independent testing process failed to detect or address these issues as well.

3.4.10 Source Code Audits

One of the most important defensive software engineering practices is to routinely audit the source code not just for functionality, but also for security. Reviews such as this one are critical for identifying security weaknesses in the system, but these such reviews also need to be initiated by the manufacturer so that changes and improvements may be integrated into the code during development. There is evidence in the documentation of vendor-initiated source code reviews as part of the independent testing process, however the broken cryptography architecture indicates that more audits by qualified security experts are required.

CHAPTER 4

Specific Weaknesses and their Implications

We identify here the weaknesses we found in the components we reviewed. For each, we state the facts we found, their implications for system security, and the observations that led us to our findings. Since we made no attempt to give equal attention to all components of the system, the distribution of weaknesses among components in this list should not be considered representative of the actual number of weaknesses that might exist in each component. Also described here are a few particular non-software weaknesses that have implications in combination with other software weaknesses.

This weaknesses identified here are considered primarily from the perspective of the *security* of the Sequoia System. That is, we focused on how a malicious adversary might exploit a flaw to harm an election.

Also of concern, however, is the effect of flaws on the *reliability* of the system. That is, even without a malicious adversary exploiting them, program flaws can cause the system to operate incorrectly, possibly with with the effect of altering election results. Reliability and security are therefore related, and many of the weaknesses enumerated in this section also have a strong potential impact on reliability (see also Section 3.4.6).

Throughout this report, markers of this form: *1 refer to proprietary details of the Sequoia source code such as file names, function names, and line numbers. These references are given in a separate proprietary annex to this report.

4.1 WinEDS

4.1.1 WinEDS creates administrator-level database accounts for all users.

Finding: For every WinEDS user account, WinEDS creates a corresponding account on the database server with administrator privileges, using the WinEDS account's username as the database account username and the WinEDS account's password, plus a short suffix,*5 as the database account password. The suffix is the same for all accounts on all installations of WinEDS.

Implication: Knowledge of the short fixed suffix is sufficient for any WinEDS user (even with minimal WinEDS privileges) to log directly into the database using a SQL client program, bypassing all WinEDS access restrictions and gaining complete control over all data. Since Microsoft SQL Server supports `EXEC` statements, this also permits the execution of arbitrary system commands on the server.

The fixed suffix is short enough to be guessed by exhaustive search. Because it is fixed in the code, it can also be found in the WinEDS software files installed on every WinEDS client workstation. Other weaknesses in WinEDS (Sections 4.1.7, 4.1.8) further facilitate the discovery of the fixed suffix.

Observations: We examined the source code for WinEDS and found that the fixed suffix is used by a function whose name suggests that it performs encryption. This function[*6] merely appends the fixed suffix to its input; yet the comments for the function describe the input as "A string to be encrypted" and the return value as the "encrypted string." This function is called during the login process[*7] on the user's password to yield the database login password.

Using the WinEDS system installed by Sequoia in the office of the Secretary of State, we confirmed that we could connect directly to the database using the username of a WinEDS account and appending the short fixed suffix to the WinEDS account's password.

On our own installation of WinEDS, we used a database administration tool to observe that the accounts created by WinEDS are members of the `sysadmin` server role.

4.1.2 WinEDS does not encrypt or authenticate database communication.

Finding: Database queries and responses are transmitted in the clear (i. e. without encryption) between WinEDS and the Microsoft SQL database. Also, communication with the database is not authenticated, so — if the database and the WinEDS client are running on different computers — it can be intercepted and altered in transit by anyone who has gained access to the local network.

Implication: If the database and the WinEDS client reside on different computers, anyone who has gained access to the network can eavesdrop on election data travelling between WinEDS and the database, or make arbitrary changes to the database by modifying database requests in transit. Additionally, by eavesdropping while any user changes his or her password, anyone with access to the network can discover the user's password (see Section 4.1.5) and use the password to access the database with unlimited privileges (see Section 4.1.1).

Observations: We installed a WinEDS client and Microsoft SQL Server 2005 on two separate computers on the isolated local network in our source code review room. Using the Wireshark network protocol analyzer[1], we observed that messages between WinEDS clients and the database server were not encrypted; SQL statements and responses were easily discernible.

On a third computer, we installed Ettercap NG 0.7.3[2], and compiled a simple Ettercap filter to replace the word "orange" with the word "purple". We then started Ettercap and told it to conduct a man-in-the-middle attack (using ARP poisoning) between WinEDS and the SQL Server, using the filter we compiled. In the WinEDS client, we opened a contest and double-clicked on a candidate to edit it. We entered the last name "orange" for the candidate and applied the changes. When we double-clicked on the candidate again, we observed that the candidate now had the last name "purple". We examined the candidate record in the database and confirmed that the candidate's last name had indeed been changed to "purple". We did not find any evidence of the attack in the WinEDS error log file.

We have not attempted to verify whether eavesdropping or traffic injection are possible if the database server and WinEDS are running on the same machine.

It appears likely that WinEDS is not designed to support encrypted communication with the database. We could not find any mention of encrypted database connections, secure database connections, or SSL in the WinEDS 3.1 Installation Guide, Reference Guide, or Operators Manual. We did not find any options for encryption offered during the software installation process, and the login window provides no database properties page where encryption could be an option. Finally, there is no place in WinEDS to manage certificates.

Even if WinEDS does have an undocumented ability to use encryption when communicating with a database, we observed that WinEDS provides no notification that encryption is disabled.

4.1.3 WinEDS does not remove database access for deactivated users.

Finding: After deactivating a user in WinEDS, the corresponding user account for the database is still active.

[1]Wireshark is available at `http://www.wireshark.org/`.
[2]Ettercap is available at `http://ettercap.sf.net/`.

Implication: User accounts that have been deactivated in WinEDS, such as those for former employees, can still be used to log into the database to obtain full control of all data (see Section 4.1.1).

Observations: We created a user account in WinEDS and confirmed that we could log into the account. We then logged in using the administrator account `sa` and deactivated the user in the Edit User dialog (WinEDS does not appear to have a command to delete user accounts). Afterwards, the user disappears from the WinEDS user list and cannot be used to log into WinEDS anymore. However, we were still able to use a SQL client to log into the database, using the supposedly "deactivated" account's username and password with the short password suffix (see Section 4.1.1). The database account was still a member of the `sysadmin` server role.

4.1.4 WinEDS changes account usernames incorrectly.

Finding: When the username of a user is changed in WinEDS, WinEDS changes its own record for the user but does not create a database account for the new username. This leaves the user unable to log in.

Implication: Users can be locked out of WinEDS after having their account name changed.

Observations: We created a new user, confirmed that this user can log into WinEDS, and then renamed the user by clicking on its name in the dialog Configuration → Security → Users. We then tried to log into WinEDS using the old and new usernames, and tried the password with and without the password suffix described in Section 4.1.1. All login attempts failed. When we examined the database, we found that there was no account corresponding to the new username, and the account with the old username still existed. We confirmed that we were still able to log into the database using the old username and the password extended by the short suffix mentioned in Section 4.1.1.

4.1.5 WinEDS does not encrypt password change requests.

Finding: When a user changes her WinEDS password (as is required when she uses WinEDS for the first time, or by selecting Tools → Change Password...), and the database is running on a different computer, the old and new passwords are passed in the clear over the network (with the password suffix attached) as parameters to a stored procedure in the database.

Implication: Since the network connection with the database is not encrypted (see Section 4.1.2), anyone who can observe network traffic during the password change request can discover the user's new password with the password suffix, and then use this password to connect to the database with administrator privileges (see Section 4.1.1).

Observations: We installed a WinEDS client and Microsoft SQL Server 2005 on two different machines on the isolated local network in our source code review room. As the WinEDS administrator, we created a new user account. Then we logged out of WinEDS and logged in with the new username and the default password. WinEDS immediately requested that we change the password for the new account, and we entered `applepie` as the new password. Using the Wireshark network protocol analyzer, we observed the default password plus the short password suffix and `applepie` plus the short password suffix travelling in the clear in network traffic.

We repeated these steps using the Tools → Change Password... command in WinEDS, and again observed that both the old and new passwords were easily discernible.

4.1.6 WinEDS retrieves the default password in the clear on every login.

Finding: WinEDS assigns a default password to new user accounts. This default password is stored in the database and may be modified by an administrator. When a user logs in, WinEDS checks whether the user's password matches the default password; if so, the user is required to select a new password. In order to make this check, WinEDS retrieves the default password with a database request after every successful login.

Implication: Since communication between WinEDS and the database server is not encrypted (Section 4.1.2), a person with access to the local network over which WinEDS connects to the database server — if the database and the WinEDS client are running on different computers — can learn the default password by examining network traffic.

Observations: We examined the database and found the default password stored in a particular field of a database table.*[8] We also observed in the WinEDS source code*[9] that this field is retrieved after every login to compare it with the user's entered password.

We installed a WinEDS client and Microsoft SQL Server 2005 on two computers on the isolated local network in our source code review room. Using the Wireshark network protocol analyzer, we observed the default password transmitted in cleartext over the network whenever a user logs in using WinEDS.

4.1.7 WinEDS places the password suffix in a password entry field.

Finding: When the user's password matches the default password, WinEDS presents a Password Expired window immediately upon login and forces the user to choose a new password. The Current Password field of this window contains the default password plus the password suffix (even though it is displayed as a string of asterisks).

Implication: The user can easily discover the password suffix using a tool that reveals the contents of password entry fields. (Many such tools are available for free download, such as X-Pass and 123 Password Recovery.) Having done so, the user can then use the default password with suffix attached to gain complete control of the database (Section 4.1.1).

Observations: On our installation of WinEDS, we created a new user account, then logged in as this user using the default password. We used X-Pass to reveal the contents of the Current Password field, and observed that it contained the default password with suffix attached. We confirmed that we could use this password with suffix to log into the database directly and obtain full access.

4.1.8 WinEDS displays the password suffix when resetting passwords.

Finding: WinEDS provides a Reset Password feature that allows the WinEDS administrator to reset any user's password to the default password. The fixed password suffix is displayed in the WinEDS user interface whenever this feature is used.

Implication: A user who asks the administrator to reset a forgotten password may be able to see the fixed suffix appear on the administrator's screen. Having discovered the password suffix, the user can then gain complete control of the database (Section 4.1.1).

Observations: We examined the source code that handles the Reset Password feature and found that it appends the suffix to the default password, changes the database password, then displays a message containing the database password in the status bar.*[10]

On our installation of WinEDS, we logged in as the administrator, opened the Users tab under Configuration → Security..., and double-clicked on a username in the list. Clicking the Reset Password button yielded the message

> Successfully reset password to '*password*'

in the status bar of the main window, where *password* is the default password plus the short suffix.

4.1.9 WinEDS changes the password for the administrator incorrectly.

Finding: For the account named `sa` (which is the default username for the system administrator account), WinEDS uses the entered password as the password for logging into the database, without adding the password suffix. However, the Change Password feature of WinEDS adds the password suffix to the newly chosen password.

Implication: If the system administrator's account is named `sa`, then after changing his or her password with the Change Password command, the administrator will not be able to log in using the new password he or she selected. The administrator will no longer be able to use WinEDS unless he or she knows and appends the password suffix to the newly chosen password.

Observations: We examined the source code for logins and for changing passwords. The routine for handling a login compares the username to `'SA'`*11 to decide whether to add the password suffix. But the routine for changing a password compares the *password* to `'SA'`*12 to decide whether to add the password suffix.

On our installation of WinEDS, we logged into WinEDS using the administrator account with username `sa` and selected Tools → Change Password..., then entered `cheezburger` as the new password. WinEDS responded "Password changed successfully" and instructed us to restart WinEDS and log in with the new password. We then tried to log in with the password `cheezburger` and the login failed with an "Incorrect login" message. Only by adding the suffix to the end of `cheezburger` were we able to log in.

4.1.10 The WinEDS Data Wizard lets any user export data from the database.

Finding: Using the Data Wizard tool in WinEDS, any WinEDS user can export almost any table in the Profile database. Any user with permission to open an election can export almost any table in the Election database as well. (The exceptions are tables containing binary columns, which the Data Wizard does not support.)

Implication: Any user of WinEDS with access to an election can read nearly all the information in the election database, regardless of the access rights they have been given. Since all of the predefined roles have permission to open elections, it is probable that nearly all users can access this information.

Observations: WinEDS provides a tool called the *Data Wizard* that allows users to export and import database tables to and from tab-delimited text files. On our own installation of WinEDS, we created a number of WinEDS users with varying levels of access rights and found that the Data Wizard was available to all of them, regardless of their assigned permissions.

In particular, we created a user with no access rights at all by assigning the user only one role, a role with every access right deactivated. We confirmed that this user could run the Data Wizard and use it to export tables from the Profile database. Adding a single access right (the Query right on the Open Election component) was sufficient to enable this user to open an election. We confirmed that after opening an election, the user could then export tables from the Election database. (Attempting to export tables containing binary columns yielded an error message saying that a column "requires the use of an embedded SQL statement.")

4.1.11 The WinEDS Data Wizard lets any user erase any table in the database.

Finding: Using the Data Wizard tool in WinEDS, any WinEDS user can erase any table in the Profile database. Any WinEDS user that can open an election can erase any table in that election's Election database.

Implication: Using the Data Wizard, any WinEDS user can erase tables in the Profile database to render WinEDS inoperable. After such erasure, no one (not even an administrator) can log into WinEDS from any workstation; the database must be repaired separately.

Any user that has access to an election can destroy all the election information in the database, or destroy some tables but not others, leaving the election results in an inconsistent state. Since all of the predefined roles have permission to open elections, it is probable that nearly all users can cause this damage.

Observations: On our own installation of WinEDS, we created a user with no access rights (Section 4.1.10). After logging in as this user, we started the Data Wizard. The Data Wizard offers a choice of four actions: Append, Replace, Update, and Export. We clicked Replace and selected a particular table,*13 then clicked Next. The Data Wizard displayed an error message about incorrect syntax; we clicked OK to dismiss the message, then clicked Next and selected an empty text file from the desktop. The Data Wizard displayed another error message about a missing column; we clicked OK to dismiss the message, then clicked Finish. The Data Wizard displayed a third error message about the lack of an "UPDATE capability" and we clicked OK again.

We examined the database using a database administration tool and found that the table we had selected was now empty. All WinEDS logins failed after this point.

We then repaired the database using the database administration tool and gave this user a single access right (the Query right on Open Election). This enabled the user to open election tables in the Data Wizard. Repeating the procedure above for a particular election table*14 erased the election returns. We observed that election returns that were previously visible in the Declare Winners window no longer appeared in that window.

4.1.12 The WinEDS Data Wizard's import function does not work.

Finding: The WinEDS Data Wizard is documented and intended to be used for importing data into database tables, but this function does not work.

Implication: Remarkably, this implementation error has the positive security implication that one avenue of unauthorized database access turns out to be defunct. (Many other avenues still exist, described in Sections 4.1.1, 4.1.2, 4.1.5, 4.1.10, and 4.1.11). However, this is documented as an intended function of the Data Wizard, so the fact that it is written in a way that can never work suggests that it was not tested prior to release.

Observations: We tried to use the Data Wizard to Append or Update data in tables, and received an error message about invalid SQL syntax. We tried a number of different database tables, which all yielded the same result.

We examined the source code for the Data Wizard's import function,*15 which constructs a SQL statement to obtain information about the table into which data is to be imported. We found that the function fails to place the table name in the SQL command, causing a SQL syntax error. There is a line that attempts to append the table name to the SQL command,*16 but the variable that is supposed to contain the table name*17 is never assigned any value.

4.1.13 WinEDS does not validate a format string read from the database.

Finding: WinEDS interprets a particular field*18 stored in the database as a format string. The string can be entered in the WinEDS user interface or placed directly in the database. Four of the user roles supplied with WinEDS (Clerk, Administrator, PHASE I – Election Data, and PHASE II – Ballots) have sufficient permissions to enter the text string, but users in other roles may escalate their privileges to be able to enter the string as well (see Section 4.1.15).

Implication: A person that gains access to the WinEDS database, for instance as a legitimate WinEDS user or by sniffing a password on the network (see Sections 4.1.1, 4.1.2), may be able to take control of any WinEDS client that executes the vulnerable function in WinEDS.

Since anyone on the network can modify communication in transit between a WinEDS client and a database server running on distinct computers (see Section 4.1.2), an attacker can also inject an invalid format string into WinEDS over the network, and may be able to take control of WinEDS in this fashion.

Observations: We entered a format string containing format codes such as `%s` and `%n` into a particular dialog in WinEDS.*[19] We found that these strings were stored unsanitized in the database. A WinEDS function*[20] that reads this string passes it on to a second function*[21] that interprets it as a format string modifier and calls the library function `vsprintf`, passing on the unsanitized string and the next word on the stack. When we selected the WinEDS function*[22] in the GUI that reads this string, the program crashed with an error message if the string contained format modifiers such as those mentioned above, but seemed to work as intended when entering only letters. The length of the format string is limited to 60 characters before being passed to `vsprintf`.

4.1.14 WinEDS accepts negative vote totals from the database.

Finding: When the official total for an election is calculated using Post Election → Declare Winner, WinEDS does not check whether any vote totals stored in the database are negative. Any negative vote totals are included in the canvass report.

Implication: Since the database is protected using poor access control (Sections 4.1.1, 4.1.2, 4.1.10), a WinEDS user or an intruder listening in on communication between WinEDS clients and the database server may be able to edit the database and enter negative vote totals. Undetected by WinEDS, these are printed in canvass reports.

Observations: We created votes in the database and generated a canvass report using the Declare Winner function. No negative totals were displayed. After we modified vote totals in the database, the canvass report included the negative vote totals.

4.1.15 Some WinEDS default user roles allow users to gain administrative privileges.

Finding: WinEDS comes with ten default user roles (see Section 2.1.1). Besides users in the Administrator role, who have unlimited access to WinEDS, users with one of the roles PHASE I – Election Data, PHASE II – Ballots, PHASE III – Machine Programming, PHASE IV – Tally, PHASE V – Post Election, and PHASE VI – Archived have permission to assign roles to users, and can thus circumvent the WinEDS access control by assigning themselves the Administrator role.

Implication: Users in the six roles mentioned above have complete control over WinEDS, including all election data and other user's accounts.

Observations: We created a new user `foo` in WinEDS, and assigned it the role PHASE I – Election Data. We logged in with username `foo`, changed the account password and logged in again. Using the dialog Configuration → Security → Users, we added the role Administrator to the list of roles assigned to `foo`, saved our changes, and quit WinEDS. We then logged into WinEDS using the administrator account and removed the Administrator role from the user `foo`. We repeated these steps for the roles PHASE II – Election Data through PHASE VI – Archived.

4.1.16 WinEDS access rights are poorly described and documented.

Finding: The WinEDS documentation describes user access capabilities in insufficient detail and overly broad terms.

Implication: The poor usability of the WinEDS access control system may lead to configuration mistakes and frustrated administrators. These problems might in turn result in overly broad access permissions.

Observations: The Configuration → Security dialog consists of a matrix of 111 components with 6 types of access to each component (New, Edit, Remove, Query, Admin, All). It contains 111 rows of checkboxes, with up to six checkboxes in each row (we counted a total of 615 checkboxes).

The documentation[3] gives a one-line explanation for each of the six types. We did not find definitions for the 111 components in the documentation. Furthermore, the per-component permissions are not independent, since enabling some access capabilities is ineffective unless certain others are also enabled. We did not find a documentation of these dependencies.

4.1.17 WinEDS access rights are difficult to minimize.

Finding: The user interface for assigning access rights in WinEDS consists of 111 rows of checkboxes, with up to six checkboxes in each row. An administrator attempting to configure access rights must individually turn on or off these checkboxes. In particular, there is no easy way to clear all the checkboxes or manipulate them in groups.

Implication: Administrators might be willing to trade off the need to minimize access permissions and the effort required to assign these permissions. This might result in overly broad permissions.

Observations: When creating a new role, 111 of these checkboxes (all the checkboxes in the "Query" column) are turned on by default. To start from a clean slate, the administrator must individually click off all 111 checkboxes by hand.

4.1.18 WinEDS fails to check some function return codes.

Finding: We found several examples of code where WinEDS does not check the return code of a function for errors.

Implication: As discussed in Section 3.4.5, failing to check for and handle errors affects the reliability of a system, and to the recoverability of the system when problems occur.

Observations: WinEDS utilizes BCP (Bulk Copy Program), a utility included with Microsoft SQL Server, to process cartridge data from database blobs, to temporary files, and back into the database again. In numerous instances,[*23] the result of the SQL `EXEC` call is not checked. This is just one of many[*3] examples.

4.1.19 WinEDS contains many small buffer overflows.

Finding: On many occasions, WinEDS performs unchecked memory accesses which could potentially violate memory safety.

Implication: If an attacker has control over the data written beyond the bounds of the buffer and the memory layout is favorable to her, she may be able to change the behavior of WinEDS, or even execute arbitrary code. Errors like these are indicative of fragile programming practices, which might lead to vulnerabilities. This is particularly true in large and highly configurable systems.

Observations: We give some examples of unsafe practices in the WinEDS code base that indicate fragile programming practices prone to buffer overflow vulnerabilities. These are not necessarily exploitable, but indicate poor code quality.

- We found at least 14 instances of code that causes a two-byte overflow, most often when using the function `sscanf` incorrectly by writing a four-byte integer to a memory slot reserved for a two-byte short integer.[*21]
- WinEDS uses a homemade function for converting 8-bit to 16-bit characters which does not take into account the length of the buffer that the resulting string is written to, and is thus inherently unsafe. On one occasion, a language name obtained from the database is converted[*24] and written to the heap. The length of the language name is restricted by both the PowerBuilder code querying the database and by the database layout.

[3] *WinEDS 3.1 Reference Guide, document version 6.02*

- On another occasion, when writing a string containing the name of a language to memory, WinEDS checks the bounds of the source, not the destination.*25 In fact, a comment next to the write operation explicitly warns of a buffer overflow, and still an erroneous bounds check is used, possibly because the countermeasure was not tested to be effective.
- A routine*2 that converts an integer ranging between $-2,147,483,648$ and $2,147,483,647$ to a null-terminated string allocates only only a 10-byte buffer for the result, potentially causing a two-byte buffer overrun.

4.1.20 Integrity checking of Results Cartridges by WinEDS is not adequate to detect tampering.

Finding: WinEDS does perform a check to detect whether a the contents of a Results Cartridge has been tampered with. When writing to the Results Cartridge, the Edge adds a cryptographic message authentication code. However, apparently to provide backward compatibility, it is possible to construct an code that is accepted by WinEDS without access to any key material or other secrets.

Implication: An attacker can modify the votes on a Results Cartridge and WinEDS will accept and tally them.

Observations: We used our local installation of WinEDS to prepare Results Cartridges for a sample election. (Since we did not have PC cards to use as Results Cartridges, we used floppy disks. Results Cartridges inserted into the WinEDS computer supplied by Sequoia appear as normal disks with files and directories.) Using tools we created for this purpose, we added some vote records to this floppy disk. We found WinEDS would accept and tally the results on the floppy disk. We confirmed that we did not have to change any information on the disk other than the vote records themselves in order to change the votes tallied by WinEDS.

4.1.21 WinEDS fails to check the integrity of election results

Finding: WinEDS does not always check whether a cartridge has been tampered with. If certain checksums on a Results Cartridge are zero, or the Edge firmware version on an Audit Trail Cartridge is too small, or the cartridge is an Early Voting Cartridge, the checks are omitted.

Implication: Somebody who gains control of a cartridge can modify the election returns on it and cause WinEDS not to check for tampering when the cartridge is tallied. Please see Section 3.2.2 for weaknesses of the SHA and CRC integrity check mechanisms.

Observations: The function*26 in the WinEDS source that determines whether the integrity checks should be performed checks for the conditions mentioned above. These weaknesses are also described in source code comments.

4.1.22 WinEDS does not prevent an integer overflow during preferential vote tallying.

Finding: When WinEDS computes tallies for preferential contests from votes read from a Results Cartridge, malicious data on the cartridge can cause an integer overflow in WinEDS, which can be used by an attacker to write to memory addresses of her choice. We have no evidence that the code containing the weakness is ever executed.

Implication: If the code for tallying preferential contests was used or could be activated by an attacker, a maliciously prepared Results Cartridge could likely cause the execution of arbitrary code on the WinEDS client.

Observations: When preferential ballots from a maliciously prepared cartridge are tallied by WinEDS, the software calculates the size of a buffer*27 using an arithmetic operation depending on the number of candidates in preferential contests and the maximum number of candidates in a preferential contest. Both values are completely determined by a file*[28] on the Results Cartridge, and not checked for overflows. An attacker can cause the arithmetic operation used to calculate the size of the buffer to overflow, so that the amount of memory allocated on the heap is smaller than expected.

When tallying, WinEDS writes to the buffer.*[29] For each candidate, it writes a 32-bit candidate ID, followed by a number of 32-bit vote counters incremented corresponding to how often the candidate was ranked first, second, and so on. Both the candidate ID and the votes are read from the cartridge and are thus controlled by the attacker. In particular, using the candidate ID, an attacker can write 256 KB of evenly spaced 32-bit words to memory, with the spacing under her control. She can also use the vote counters to write up to 4 GB of data, with the additional constraint that these writes must be performed as integer increments.

We found that the vulnerable code is only called*[30] if a certain boolean flag is set to `false`. There is a function exposed through an ActiveX interface*[31] which sets this flag to `true`, but were unable to find a call to this function. Furthermore, preferential votes are only tallied depending on a flag in the database,*[31] which is was not set in any of the county-specific configuration files included with the source code.

4.1.23 WinEDS trusts the list of precincts for which a Results Cartridge claims to report votes.

Finding: A Results Cartridge can report results for an unlimited number of precincts other than the one it was assigned. If WinEDS tallies such a cartridge, it will accept all the votes on the cartridge, including for other precincts. These votes are included in the Statement of Vote and in some, but not all, reports generated by WinEDS.

Implication: If an attacker has access to a Results Cartridge, she can modify the data on the cartridge to report extra votes for all the precincts in the county. This is achieved by adding or changing **Selection Code** records that identify which ballot each voter received (for example, it may indicate party affiliation in primary elections). When creating reports based on election data that includes the votes from the corrupted cartridge, some reports will include the spurious votes and some will not. Comparison of reported votes against poll worker records will show that more ballots were cast than voters authorized to vote. However, it may be very difficult to determine which Results Cartridge is responsible for the extra votes.[4]

We considered two possible methods for identifying the corrupted Results Cartridge through WinEDS, but realized these methods would not always work:

- WinEDS has a feature to list the number of voters by voting machine. If for any single machine the number of voters exceeds the turnout at the entire precinct, this indicates that the results from this machine are erroneous. However, the corrupted Results Cartridge can report votes for other precincts and still escape detection by this method as long as its total does not exceed the turnout at its own precinct.

- One could insert each cartridge and generate an Election Results by Selection Codes Report from the menu Tools → Cartridge Utilities → AVC Edge for each one, looking for cartridges with more than one Selection Code. However, the selection codes displayed (both in this report and in the pull-down menu of selection codes) are listed by name, and the names shown are determined by the cartridge itself. A compromised Results Cartridge could therefore contain votes for a single Selection Code specifying the wrong precinct, but escape detection by using the name of the legitimate precinct.

Direct examination of the database may be able to identify the corrupted Results Cartridge, though we did not attempt to do this.

[4] It is not unusual for a county to have thousands of Edges, and therefore thousands of Results Cartridges.

Observations: Each vote reported on a Results Cartridge contains a Selection Code[*32] linking the vote to the precinct in which it was cast. For each vote on the Results Cartridge, WinEDS uses this information to determine in which precinct tally the vote will be included. We created a Results Cartridge using WinEDS. Using custom-written tools, we created a Selection Code index file[*33] mapping selection codes to precincts, and containing fabricated names for the selection codes. We then stored several votes on the cartridge, assigning them different selection codes to associate them with different precincts. When we tallied election results using the function Post Election → Resolve Winners, we observed that WinEDS included all the votes stored on the Results Cartridge. Similarly, the Statement of Vote generated via Election → Reporting → Post Election → Statement of Vote Report included the spurious votes. However, the canvass report generated via Election → Reporting → Post Election → Canvass Report *did not* include the additional votes, since it lists for each precinct only the cartridges that were originally assigned to the precinct. The per-precinct totals in this report will not match the totals in other reports that *do* include the spurious votes, such as the Election Returns report.

We generated every report in the Election Day and Post Election sections, but were unable to see how the corrupted cartridge could be identified in any of these reports. We also attempted to generate reports for the individual cartridge using the Tools → Cartridge Utilities → AVC Edge command, and observed that the Election Results by Selection Codes Report showed only the fabricated Selection Code names we had stored on the cartridge, not the actual names or numbers of the precincts affected.

4.1.24 WinEDS writes to an array index read from the Results Cartridge without checking whether it is negative.

Finding: The WinEDS function for printing results reports for a specific cartridge writes to memory using an array index specified by a file on the Results Cartridge, without checking whether it is negative. The file contains logical descriptions of each candidate.

Implication: Each candidate record read from the file[*34] contains an unsigned short describing a party endorsement.[*35] This unsigned integer is then assigned to an index variable of type (signed) short. There is no bounds check on the party endorsement. A bounds check on the index[*36] prevents values that are too large to be valid indices, but does not exclude negative numbers as small as -2^{15}. Subsequently, the constant 1 (as a two-byte short integer) is written to an address determined by a heap pointer and the index read from the file. For every candidate included in the file, the attacker has therefore the ability to set, depending on alignment, two consecutive bytes to the value representing the constant 1, where the targeted memory locations range over 8 KB range of memory. Moreover, this write operation is executed in a loop whose exit condition is completely controlled by the attacker, possibly allowing her to make a series of writes.

We have not attempted to exploit this vulnerability, but we are unable to rule out the possibility that this weakness may be used by an attacker to change the behavior of WinEDS or even to execute arbitrary code on the WinEDS client.

Observations: The vulnerable feature is activated by selecting Tools → Cartridge Utilities → AVC Edge, clicking on Menu → Reports, and requesting a report. The vulnerable code is contained in the function printing candidate endorsements. The array index, which represents is first read into an unsigned short, but then assigned to a *signed* short which is used to address the array. If the unsigned short is large enough, the array index will be negative. We confirmed that WinEDS does not detect a negative index by loading first a well-formed file and then a file containing a negative index, which was assigned to the variable[*37] used to determine the target address of the write operation.

4.1.25 WinEDS increments integers in an array using unchecked array indices read from a Results Cartridge.

Finding: For each candidate running in a race, the tallying algorithm for undervotes uses an entry from a file on the Results Cartridge as an index into an array on the heap, without checking that the entry actually corresponds to a valid index.

Implication: By modifying the indices specified in the file, an attacker can cause WinEDS to write to memory locations outside the buffer. (For instance, she might cause several writes to the same address to set it to a desired value.) Furthermore, the size of the buffer is determined by a file*[38] on the Results Cartridge, so that the effect of the attack can be maximized. Limited by the index, the attacker can write to a range of at most 2 MB of memory. We did not attempt to implement this attack, and, given the amount of control the attacker has, we are unable to rule out the possibility that it might give an attacker who can introduce modified cartridges the ability to change the behavior of WinEDS or even to take control of the WinEDS computer.

Observations: We examined the WinEDS source code, and found that a function reads*[39] the index directly from a file on the Results Cartridge.*[40] The size of the array is determined by a member variable which is in turn determined by the size of a file*[41] on the Results Cartridge. The memory location specified by the array pointer and the index is then incremented*[42] as a 32-bit long integer.

4.1.26 Some WinEDS reporting functions will never terminate if given a malformed Results Cartridge.

Finding: A circular linked list*[43] read from a file*[33] on a Results Cartridge can cause a report generation in WinEDS to run forever. The linked list describes party endorsements for candidates.

Implication: A corrupted Results Cartridge can cause WinEDS to run in an infinite loop. Fortunately, the user can abort the loop by pressing a Cancel button.

Observations: The vulnerable feature is activated by selecting Tools → Cartridge Utilities → AVC Edge, clicking on Menu → Reports, and requesting a report. The linked list is defined using the array indices described in 4.1.25. From the source code, we determined that WinEDS checks whether the list is circular by stopping once it encounters the first element it processed. We modified the file so that the list contains a loop excluding the first element. Loading the file in WinEDS and requesting the report caused the application to run loop until we pressed Cancel.

4.1.27 WinEDS assumes the MPR is authentic and reliable.

Finding: WinEDS relies on the MPR to read and write election information on MemoryPacks, but it does not verify that the MPR is working correctly.

Implication: An MPR, once subverted, may escape detection for a long time, while misreporting election results and corrupting MemoryPacks. The fact that the case of the MPR can be easily opened (4.7.1) and that its software can be easily replaced (4.7.2) makes this a more significant risk. Also, since each Insight is under complete control of the MemoryPack placed in it (4.6.2), a subverted MPR can use its ability to corrupt any inserted MemoryPack to subvert the behavior of any Insight in which such a MemoryPack is later used.

Observations: We examined the WinEDS module that communicates with the MPR*[44] and found no code that tests the functioning of the MPR or challenges the MPR to prove that it is an authentic MPR. When starting a conversation with the MPR, WinEDS asks the MPR what software version it is running,*[45] and trusts the MPR to answer honestly and accurately. Aside from this version number check, we did not find any code in WinEDS that validates the actual software on the MPR.

We constructed a program that simulates a MemoryPack Receiver and used it to communicate with a computer running WinEDS. By observing the commands sent by WinEDS, we confirmed that WinEDS proceeds to request election results from the MemoryPack after asking the MPR for its software version number only once, even if the MPR never reports the correct version number.

4.1.28 WinEDS assumes that MemoryPack data received from an MPR is correct.

Finding: WinEDS performs no checking to detect inadvertent corruption or tampering of vote counts received from the MPR. Vote counts are also not checked to determine whether they are in a reasonable range.

Implication: Since the MPR performs only weak checking of the vote counts on the MemoryPack (4.7.3), the results received from the MPR are not trustworthy; since WinEDS also does not check these vote counts, vote counts on MemoryPacks are vulnerable to undetected tampering.

Observations: We examined the WinEDS module that receives vote counts from the MPR.*44 The communication protocol between WinEDS and the MPR includes a simple checksum after each message, but we did not find any code in WinEDS that verifies checksums, hashes, or signatures on the vote counts themselves.

We constructed a program that simulates a MemoryPack Receiver and used it to transmit results to a computer running WinEDS as if those results came from MemoryPacks. After using WinEDS to initialize a virtual MemoryPack in our simulator, we observed that WinEDS would load any results our simulator transmitted, including, for example, 50,000 votes in a precinct with only 100 voters. We repeated the process with varying vote counts, and confirmed that we did not have to change any MemoryPack information other than the vote count itself in order to load different vote counts into WinEDS.

4.1.29 WinEDS assumes that the MemoryPack serial number received from the MPR is correct.

Finding: The WinEDS database records which MemoryPacks are assigned to which precincts. When WinEDS receives election results from a MemoryPack, it looks up the MemoryPack by its serial number to determine which precinct those results are for. WinEDS obtains this serial number by asking the MPR to read a specific memory location on the MemoryPack. If the serial number corresponds to any known MemoryPack that has not already been processed, it is accepted. The serial number is a seven-digit number entered manually into WinEDS by the operator for individual Insights or generated sequentially by WinEDS for a series of Insights.

Implication: Changing the serial number in a MemoryPack to a different existing serial number allows that MemoryPack to impersonate another MemoryPack, i. e. to have its results treated as results from the other MemoryPack, which can be from a different precinct. Whichever MemoryPack is loaded into WinEDS first will have its results recorded in the database. WinEDS will then refuse to load results from the other MemoryPack, telling the operator "Cartridge was already processed." Whether the operator is likely to be misled by this message, and whether the operator can determine which cartridge is the true one, depends on the procedures used at a particular election office.

Observations: We examined the WinEDS routine that gets the MemoryPack serial number*46 and found no code that verifies this number. We found no reason to believe that MemoryPack serial numbers are supposed to be secret, as they are exposed in the WinEDS interface and printed out on Insight paper tapes.

We constructed a program that simulates a MemoryPack Receiver and used it to transmit results to a computer running WinEDS as if those results came from MemoryPacks. We observed that if our simulator transmitted any existing MemoryPack serial number and a corresponding precinct number, WinEDS would load in results for that precinct. We repeated the process with varying serial numbers and precinct numbers, and confirmed that we did not have to change any

MemoryPack information other than the serial number and precinct number to load in the same results for a different precinct.

4.1.30 WinEDS does not notify the operator if a MemoryPack contains results for extraneous precincts.

Finding: A single MemoryPack can contain results for many precincts. The WinEDS database records which MemoryPacks are assigned to which precincts. When WinEDS finds results on a MemoryPack for precincts that don't correspond to the MemoryPack's serial number, it silently ignores them and proceeds without notifying the user.

Observations: We examined the code for loading the data received from MemoryPacks into the database. This code uses information from the database about which candidate corresponds to which counter on the MemoryPack. Only the expected counters are loaded into the database. We found no code that detects or reacts to the presence of extra precincts or counters on the MemoryPack.

We constructed a program that simulates a MemoryPack Receiver and used it to transmit results to a computer running WinEDS as if from a MemoryPack with results for many different precincts. We observed that the results for expected precincts (corresponding to that MemoryPack's serial number) were loaded and the results for extraneous precincts were not. When extraneous precincts were present, we observed no indication of anything unusual in the WinEDS user interface.

4.2 HAAT

4.2.1 Sequoia System keys hardcoded in .EXE file on removable CompactFlash card.

Finding: The keys used in several components of the Sequoia System to encrypt and authenticate data, such as Voter Cards and Upgrade Cartridges for the Edge, are insecurely embedded in the executable file of the HAAT. In addition, this file resides on an internal CompactFlash card that is easily removed and read on another machine.

Implication: An attacker who gains access to a HAAT (whether used in California or not), may learn all the secrets needed to create counterfeit Voter Cards or Upgrade Cartridges. This might let individuals cast multiple votes during an election or otherwise compromise an election.

Observations: The key used for encryption and decryption of smartcards appears in the clear in the source code of the HAAT application.*[47] When compiled, this key remains evident on the executable file kept on a removable Flash card. The key used for authenticating Upgrade Cartridges on the Edge is the same as the key used for integrity checking of the smartcard on the HAAT.

4.2.2 Access control passwords for the HAAT are stored in the clear on the HAAT's CompactFlash card.

Finding: A HAAT may either be in *prepared* or *unprepared* mode. An unprepared HAAT uses only a hardcoded password that is insecurely embedded in the executable file on the CompactFlash card in the HAAT. This password is also the same for all HAAT units.

A prepared HAAT may be configured to use county-specific passwords. These passwords are stored in a plain-text XML file on a removable CompactFlash card in the HAAT.

Implication: An attacker who gains access to a HAAT or its removable CF card may learn the hardcoded password, as well as the county-specified ones.

Observations: The Red Team removed the CompactFlash card from the HAAT provided by Sequoia to the Secretary of State and provided us with the data on the card. We examined the data and found an XML file containing the `reset`, `firmware upgrade`, and `set date` passwords. The source code verifies these passwords are read from this file.*48

The Sequoia HAAT Security Specification states that the hardcoded password is the same for all HAAT units[6].

4.2.3 The HAAT does not securely erase files.

Finding: The HAAT does not securely erase its configuration files when reset.

Implication: Even after a HAAT is reset, it may be possible to obtain the erased HAAT passwords and any other configuration files from the CompactFlash card.

Observations: The deletion of files is done with a call to a standard Microsoft system library.*49 The behavior of the delete function is therefore out of the control of the HAAT. Typically, deletion is not secure in that deleted files are not thoroughly erased and may be reconstructed in many circumstances.

4.2.4 HAAT software stored on removable CompactFlash card.

Finding: Most of the HAAT software, including its main executable file, libraries, configuration files, and the Windows CE Operating System are stored on a single removable CompactFlash card.

Implication: The entire software may be easily replaced with another CompactFlash card. Even if a HAAT never finds itself in the wrong hands, a recovered CompactFlash card might still contain proprietary secrets (see Sections 4.2.3, 4.2.2 and 4.2.1).

Observations: The Red Team provided us the contents of the internal CompactFlash card.

4.2.5 Firmware is not authenticated by the HAAT before updating.

Finding: HAAT's main application as well as the Operating System (Windows CE) can be updated through the HAAT's user interface. The files, inserted into it with a USB stick, are not validated and are just copied. No effective authentication of the data is done. Version numbers are insecurely compared in an effort to allow only more recent updates to be applied.

Implication: Any file, or set of files, may replace the existing software on the HAAT. Any file may replace the entire image-file of the operating system (Windows CE). This makes it very easy to take control of a HAAT without breaking any tamper-evident seals.

Observations: The directory containing the executable file, libraries and operating system is simply copied from the USB stick to the CompactFlash card.*50 A script also runs an arbitrary command before and after files are copied.

4.2.6 Election configuration files are not authenticated before being loaded on the HAAT.

Finding: Before loading configuration files from a Preparation Cartridge, the HAAT validates the structure of the files to make sure they are correctly formatted. However, the HAAT does not *authenticate* the content of these files, meaning that if some values are changed between the time WinEDS creates the files and the HAAT reads them, the HAAT might not tell the difference.

Implication: If a Preparation Cartridge is inserted into a compromised HAAT, it may modify the cartridge, affecting the configuration of any subsequently prepared devices with that cartridge.

Observations: Some configuration files are copied directly*51 while others are loaded and validated by a specialized *reader class*.*52 However, no secure authentication methods, such as cryptographically strong signatures, are used.

[6] *HAAT Security Specification, document version 1.00*

4.2.7 Backup Cartridge reveals county specified passwords

Finding: An XML file on a Backup Cartridge reveals the passwords for a prepared HAAT. These passwords may be used to set the HAAT in *unprepared* mode. Once unprepared, the software may be upgraded or arbitrarily modified. The feature to back up these files to a standard USB stick is not password protected.

Implication: A *prepared* HAAT (as delivered to a poll worker during an election) may be unprepared, modified, and re-prepared with the backed up files, *without* breaking any security seals. Consequently, anyone gaining brief access to a HAAT may gain full control over it.

Observations: An attacker can create a Backup Cartridge without needing a password.*53 A backed up configuration file reveals the reset password.*48 The hardcoded upgrade/preparation password can be learned through other means (see Section 4.2.2).

4.2.8 The HAAT does not have a physical locking mechanism.

Finding: The hardware specification does not indicate any locking mechanism for the HAAT is used. The HAAT provided to the Red Team by Sequoia, had no locking mechanism to prevent unauthorized access. The case is protected by six Phillips screws.

Implication: A person with physical access to a HAAT may easily and quickly remove, modify, or replace its internal components.

Detection relies exclusively on proper use and verification of seals. A person who gains access to *any* operating HAAT (whether or not it was used in California) may learn the secret keys used to encrypt, decrypt and verify data throughout the Sequoia System used in California, including the data on a smartcard (See Section 4.5.2).

Observations: The lack of a locking mechanism is apparent by inspection. The internal components are designed for ease of access to facilitate maintenance. To protect the physical integrity of the HAAT, tamper-evident seals are recommended to be used over the screws of the case[7]. Tamper-evident seals may detect, though not prevent, access to the internals of the hardware. The HAAT given to the Red Team did not have any tamper-evident seals.

Without practice, it took six minutes for a member of our team to open the case of the HAAT, replace the internal CompactFlash card, and reassemble the HAAT using only a manual Phillips screwdriver.

4.2.9 Rewritable USB media may spread an attack to other HAATs or WinEDS

Finding: A compromised HAAT may write arbitrary data to a Preparation or Upgrade Cartridge (a normal USB stick). In some cases this cartridge may then be inserted to another HAAT or back into WinEDS.

Implication: If an Upgrade Cartridge is inserted into a maliciously programmed HAAT, it may modify the cartridge affecting the configuration of any subsequently updated devices using that cartridge.

A cartridge inserted into a compromised HAAT may provide a communication vector or attack vector to WinEDS if the cartridge for any reason is reinserted into WinEDS. This may happen if the cartridge is reused for another device or for another election.

Observations: The USB Preparation and Upgrade Cartridges are rewritable media. The HAAT can read and write to the USB cartridge. Therefore, a compromised HAAT may change the contents on the cartridge arbitrarily.

HAAT upgrades are not authenticated (see Section 4.2.5). Therefore, HAATs may be fooled into upgrading with a modified version of the firmware.

[7] *Sequoia HAAT System Overview, document version 1.08*

4.2.10 Logging on the HAAT is insecure

Finding: The internal audit logs of the HAAT are stored insecurely on rewritable and removable media. Resetting a HAAT erases the log files on the HAAT. There is no mechanism for verifying the authenticity or the integrity of log files.

Implication: Logs can be easily tampered with. While they may be helpful for detecting unintentional mistakes and actions, nothing prevents an attacker from modifying the logs if physical access to the CompactFlash card, or access through software, is gained (see Section 4.2.2).

Observations: The log file is a regular text file on the internal CompactFlash card.*[52] The reset feature erases the log files.*[54]

4.3 Card Activator

4.3.1 Sequoia System keys hardcoded in .EXE file on Flash memory.

Finding: The keys used in several components of the Sequoia System to encrypt and authenticate data, such as Voter Cards and Upgrade Cartridges for the Edge, are insecurely embedded in the executable file of the Card Activator.

Implication: Anyone able to read the internal Flash memory of the Card Activator may learn the Sequoia System keys. These keys may allow someone to produce new software to run on the Edge, or allow someone to counterfeit Auto Activation Voter Cards or Early Voting lock-unlock cards, letting an ineligible voter cast multiple votes, or unlock Edges during Early Voting.

Observations: The key used for DES encryption is hardcoded in the source code.*[55] The key appears, in the clear, in the compiled executable.*[56] The "signature" key is read in from a configuration file created by WinEDS.*[55] This key is DES encrypted with the hardcoded key.

4.3.2 Card Activator uses no passwords by default.

Finding: Two features on the Card Activator are protected by passwords: booting up, and creating Early Voting lock/unlock cards. Each feature has a separate password file containing an encrypted list of valid passwords. If a password file exists but is empty, the Card Activator requires *no* password.

The password files are loaded from a Preparation Cartridge created by WinEDS. WinEDS always creates an empty password file for boot up. WinEDS also never creates a password file for Early Voting which has the effect of disabling the Early Voting feature.

Implication: Consequently, it appears the Card Activator does not use any software-based access control in practice. Anyone who gains temporary access to a Card Activator may arbitrarily alter its functionality, *without* disrupting any tamper-evident seals.

Note, the password functionality is *not* disabled. If a Preparation Cartridge happens to contain a password file, for instance one created manually, the password functionality will be used. It is not clear how this would be done without knowledge of the proprietary Sequoia encryption keys.

Observations: The function that checks passwords on the Card Activator*[57] treats a blank password file as "valid pin code." The function that WinEDS calls to create the files for the card activator*[58] only ever creates a blank password file. This is apparently the only function in the software we were given that creates the password file. The passwords in the file are DES encrypted, so it stands to reason that the file "can't" be manually created by the county without any additional Sequoia provided software.

The password file for creating Early Voting lock/unlock cards appears to never be created by any software given to us. The consequence is that this functionality is disabled.

Even if these two password files are provided (enabling password access control), features that change the underlying software are not protected by password.*[59]

4.3.3 Card Activator bug allows unlimited attempts to enter the PIN code.

Finding: The password routine (see Section 4.3.2) apparently tries to prevent guessing by beeping if a bad password is entered, and shutting down after two consecutive failures. However, an error in code has the effect of shutting down the *sound* after two consecutive failures, allowing someone to try as many passwords as they want.

Implication: The implication is that the password routine, what should be an integral part of the Card Activator access control, has not been thoroughly tested or reviewed. A possible explanation is that password usage on the Card Activator appears to be an exception, rather than a rule. This finding is not significantly consequential, however, because the password feature is already patently inadequate for access control (see Section 3.3.2). The Card Activator provided to the Red Team was not configured with a password.

Observations: The function in the source code that checks the password contains a counter for the number of invalid PINs entered. It checks whether this counter is greater than 2 and, if so, freezes the Card Activator (asking the user to reboot). Due to an error in the control flow of the password-checking routine, this function is never invoked.*[60]

4.3.4 Rewritable PCMCIA cartridges may spread an attack from a Card Activator to an Edge.

Finding: A compromised Card Activator may write arbitrary data to Preparation Cartridges or Upgrade Cartridges inserted into it. The Preparation Cartridge may in some cases also serve as a Results Cartridge for the Edge.

Implication: If a Preparation Cartridge is also used as a Results Cartridge on the Edge (meaning it is first inserted into the Card Activator to prepare it, then it is inserted into an Edge), a maliciously programmed Card Activator may modify the contents of the Results Cartridge. This modification can compromise an Edge (see Section 4.4.12).

If an Upgrade Cartridge is used on a maliciously programmed Card Activator, the Upgrade Cartridge may be modified to take over subsequently upgraded Card Activators. A Preparation Cartridge can similarly be tampered with to corrupt the configuration of subsequently prepared Card Activators.

Lastly, any cartridge inserted into a compromised Card Activator may provide a communication vector or attack vector to WinEDS if the cartridge for any reason is reinserted into WinEDS. This may happen if the cartridge is reused for another device or for another election.

Observations: The PCMCIA Preparation and Upgrade Cartridges are rewritable media. The Card Activator can read and write to the PCMCIA cartridge.*[61] Therefore, a compromised Card Activator may change the contents on the cartridge arbitrarily. At the demonstration for us by Sequoia representatives the same PCMCIA cartridge was used to first program the Card Activator, then act as the Results Cartridge for the Edge. In this case, the Card Activator could have compromised the Edge.

Card Activator upgrades are not authenticated (see Section 4.3.6). Therefore, Card Activators may be fooled into upgrading with a modified version of the firmware.

4.3.5 The Card Activator has no physical locking mechanism.

Finding: The Card Activator may be opened by unscrewing the base of its case. Seals over the screws may detect tampering, but will not prevent tampering.

Implication: The physical integrity of a Card Activator depends entirely on the choice and use of tamper-evident seals by the county.

Observations: The Card Activator provided to the Red Team did not have any security seals, and was easily opened with a screw driver. Note, physical integrity alone is not enough to protect the software on the Card Activator (see: Section 4.3.2).

4.3.6 Firmware is not authenticated by the Card Activator before updating.

Finding: When the software of the Card Activator is upgraded, no effort is made to verify the authenticity of the new firmware. In other words, no mechanism prevents unauthorized software from replacing the legitimate version. The firmware upgrade process is controlled by the existing firmware.

Implication: Because the software controls its own upgrade, a maliciously programmed Card Activator may prevent itself from being legitimately reprogrammed. This makes it difficult to detect and fix a compromised Card Activator. Because of weak access control mechanisms in the Card Activator, it is not possible to fully confirm the integrity of the firmware after installation or reinstallation of firmware. Tamper-evident seals do not suffice (see: Section 4.3.2).

Because no authentication is performed, the upgrade process is vulnerable to replacing the Card Activator software with an arbitrary file.

Observations: The upgrade function in the Card Activator simply checks for the existence of a particularly named file in an inserted PCMCIA cartridge.*62 If the file exists, it is copied directly by way of a DOS system call. If the file does *not* exist, the user is prompted with a message like "ABC123.EXE not found", revealing the required filename.*63 A malicious programmer could, for example, replace the upgrade function with a function that does nothing, preventing the Card Activator from upgrading the firmware in the future. More sophisticated attacks are certainly possible.

4.3.7 Election configuration files are not authenticated before being loaded on Card Activator.

Finding: The files copied onto the Card Activator from a PCMCIA Preparation Cartridge before an election are not checked to make sure they are valid election files, nor are they authenticated in any way before being copied to make sure they come from a trusted source. Simply having files with the correct filenames on the Card Activator is sufficient to cause the files to be copied over.

Implication: Because no authentication or validation is done on files copied from the Preparation Cartridge, it is possible to arbitrarily modify the election configuration parameters. If the cartridge data has previously been modified by a corrupted device, the Card Activator will happily load the modified files.

Observations: The code for preparing the Card Activator simply tries to copy a list of files from the PCMCIA cartridge to its internal memory. This is done using hardcoded DOS system commands.*64

4.3.8 Undocumented functionality enabled by presence of file.

Finding: The Card Activator has code to handle elections as done in Basingstoke, UK. The code to handle UK elections is slightly different than what is required in California. This code is triggered by the presence of a particular file copied from the Preparation Cartridge. As far as we can tell this feature is undocumented.

Implication: Code paths for undocumented features may introduce new vulnerabilities, and result in behavior that is not expected. This code, though definitely not used in California, still exists and may be executed depending on the undocumented presence of a particular file.

Observations: When the Card Activator starts up and checks its configuration files, it sets a flag based on the presence of a particular file.*65 This flag affects the behavior of the Card Activator, and is different than what is documented for use in California.

4.3.9 Logging is not performed on the Card Activator.

Finding: The source code makes calls to a logging function, but the function does nothing.

Implication: The lack of logging makes it significantly harder to determine the cause of errors that are not maliciously produced; for instance, if the Card Activator is used inappropriately or if a certain function fails. The frequent calls to the log function within the source code, give a false impression that events are logged.

Observations: The Card Activator source code makes calls*[66] to a specific function to write entries to the log file. This function performs no actions, and always returns `true`.*[67]

4.4 Edge

4.4.1 System files reside on Audit Trail and are replaceable.

Finding: The system files, including the boot loader and main binary containing the executable software for the edge, are stored on the Audit Trail and are replaceable.

Implication: Executable code on the Edge may be replaced, changing the behavior of how the Edge handles election results, event logs, and the voter verified paper audit trail. Without a trusted audit log and paper trail, recovery of the correct votes after an attack is likely impossible.

Observations: The Sequoia Red Team was able to access the Audit Trail Compact Flash device directly, view the files, and replace them. Refer to the Sequoia Red Team Report for more.

The executable code on the Edge controls the audit and event log, how votes are counted and recorded, and has full control over the VVPAT printer (see Section 4.4.18). Well designed attacks will not raise suspicion, and may be difficult to detect even with a 1% audit of the voter-verified paper trail.

A compromised Edge also has control over the "firmware update" process, and can choose to ignore any more requests to update the firmware. To recover, the Audit Trail Compact Flash card must be wiped out and completely reloaded before the Edge would return to normal functionality.

4.4.2 Update Cartridge keys are static.

Finding: The keys used to protect the "firmware update" process are statically defined in the code.

Implication: Anyone with physical access to an Edge anywhere (not just in the state of California) may extract these keys. This allows an attacker to forge an Update Cartridge, gaining the ability to quickly load new executable software on any Edge. Section 4.4.1 discusses the implications of this in more detail.

Once the keys have been discovered, the update process on every Edge is compromised.

Observations: The key intended to be used for cryptographic signatures is a static, globally defined value in the source code.*[68] The "update firmware" password check uses a DES encrypted password encrypted with a static, globally defined value in the source code.*[69] The consequences of statically defined keys are explained in Section 3.2.1. The end result is the ability to forge Update Cartridges.

Anyone with physical access to an Edge, and hence access to the binaries, can extract these hardcoded keys with enough time and resources. With old voting machines increasingly turning up on online auction websites, it may be possible to gain this level of access. This also places increased importance on the physical security of every Edge, including those not currently being used.

Additionally, the Sequoia Red Team discovered that the signature check can be bypassed by listing the manifest name first in the Update Cartridge manifest file. See their discussion "Vulnerability in Firmware Update Procedure Allows For File Overwrite" for more detail.

4.4.3 Update Cartridge password check is unsafe.

Finding: The firmware update password check does not prevent an unauthorized user from loading an Update Cartridge.

Implication: Combined with weaknesses from Sections 4.4.2 and 4.4.5, Update Cartridges may be forged and new executable software may be loaded by anyone with physical access to the machine. Section 4.4.1 discusses the implications of uploading new executable software in more detail.

Observations: Section 2.1.4 discusses the update process on the Edge. The password check is intended to prevent an authorized user with a valid Update Cartridge from loading new software on the Edge. However, the DES-encrypted password[*70] is stored on the cartridge itself.[*71] Combined with the weakness from Section 4.4.2, an attacker can forge this password and circumvent the password check.

4.4.4 The key intended for use in cryptographic signatures is hardcoded.

Finding: The keys intended to be used for Consolidation Cartridge signatures and Results Cartridge signatures are hardcoded on the Edge.

Implication: Anyone with physical access to an Edge, and hence access to the binaries and EEPROMs, can extract these hardcoded keys with enough time and resources. This allows these cartridges to be forged by an attacker. Please see Section 3.2.1 for more discussion.

Observations: The keys for integrity protection of the Consolidation Cartridge and Results Cartridge are defined in the source code[*72] and instantiated on the Configuration ROM.

The documentation states that this value can be the same for all customer's machines or a unique value. However, how and when these keys are generated are not discussed. It is unclear how many Edges share the same keys.

4.4.5 Key material unused in Edge signatures.

Finding: The cryptographic signature keys are unused when creating or verifying cartridge file signatures.

Implication: An unauthorized user may be able to reproduce the signatures used to verify Edge cartridges, enabling these cartridges to be forged. Please see Section 3.2.1 for more discussion.

See the observations of Section 4.4.2 for how forged Update Cartridges allow attackers to upload new executable code onto the Edge. Also, forged Results Cartridges or Master Ballot Cartridges may allow attackers to upload new executable code onto the Edge by exploiting weaknesses in Sections 4.4.12, 4.4.13, 4.4.14, 4.4.15, and 4.4.16. Section 4.4.1 discusses the implications of uploading new executable software in more detail.

Forging a Results Cartridge or Consolidation Cartridge allows corruption of the final tally, as discussed in Section 4.1.23.

Observations:

Every cartridge has a header file which specifies the cartridge type,[*73] the Edge serial number assigned to the cartridge, and cryptographic signatures for essential files, including the totals and voter block files.[*74, *75] However, the function handling the seeding of the signature takes the key as an argument but never uses it.[*76]

4.4.6 Encryption uses single DES in ECB mode.

Finding: The encryption/decryption algorithm implemented in the Edge uses single DES in ECB mode.

Implication: Please see the discussion in Section 3.2.2 for how this algorithm is weak to "cut and paste" attacks.

Observations: We analyzed the DES implementation*[77] and found that it implements single DES in ECB mode. The routines are called by the "firmware update"*[78] process, and for handling Voter Cards.*[79]

4.4.7 Some files are not checked for data integrity.

Finding: The Edge does not require CRCs or signatures for files such as the override file or report definition file.

Implication: Override or report definition files may be added to legitimate cartridges and loaded by the Edge. This enables attacks similar to Section 4.4.9.

Observations: The Edge uses an array of file names*[80] and array of flags*[81] to determine if the CRC of a file should be checked. Also, these files do not have associated signatures.*[75]

4.4.8 Randomization of voter blocks is predictable.

Finding: We analyzed the way vote records are shuffled by the Edge, and found that the original sequence of votes can be reconstructed, despite the claims in the AVC Edge 5.0 Security Overview that the shuffling method has "been deemed sufficiently random that it would not reasonably be reversible."

Implication: A person who gains access to the votes stored on a Results Cartridge can determine the original order in which votes were cast. Together with knowledge of the sequence of voters at the polling place, this could allow one to determine how individual voters voted.

Observations: The AVC Edge 5.0 Security Overview claims that:

> "The randomizing function in the AVC Edge ® uses a mathematical pseudo-random number generator that is further randomized by the value of the AVC Edge's internal real time clock at the time of the random number request. This pseudo-random number generator has been reviewed by independent computer experts and been deemed sufficiently random that it would not reasonably be reversible based on the amount of data that would serve as the basis for the reversal."

However, the way the random number generator*[77] is used reduces the possible random sequences to just 10, given the initial (hardcoded) random seed*[78].*[79] By examining the size of the vote tables on disk,*[82] an adversary can determine the state of the generator and recover the vote permutations, undoing voter anonymity.

4.4.9 Event log reports use override strings.

Finding: Event log reports generated by the Edge appear to use override strings.

Implication: Event log reports from a compromised Results Cartridge or from a compromised Edge cannot be used to detect or recover from attacks.

Observations: The Sequoia AVC Edge supports multiple languages, allowing voters to cast votes in languages such as Spanish or Chinese. This is done through override files, which provide translations for the default English messages, ballot text, and images. These files may be provided by the vendor, or created by the counties themselves using WinEDS.

There are approximately $1,727$ strings which may be overridden, falling into 9 different categories. This includes month labels, poll worker display messages, touch screen button labels, touch screen form headers, touch screen form titles, touch screen popup messages, VVPAT report strings, event log report strings, and error messages.

Using the weakness in Section 4.4.7, it is possible to add or modify an override file in the default language directory of a Results Cartridge without breaking any validation or integrity checks. Alternatively, it may be possible using a weakness similar to Section 4.4.12 to place a maliciously crafted override file in the system directory of the Audit Trail. This also opens several attack vectors for social engineering attacks by changing the strings both poll workers and voters see on the Edge.

One of these attacks involve the Edge event log. Every action performed by the Edge is logged in the event log. This includes when the machine powers on and off, when new firmware is loaded, and when votes are cast. The event log is stored on the Audit Trail, and persists after the Edge is powered off. These logs are cleared when the Edge is reset.

Instead of storing event strings such as "Power On", the event log stores numbers representing the event type. When an event log report is created, these types[*83] are used to index into an array of strings containing the event description.[*84] When a report is requested, the Edge checks for the presence of an override file in the default language directory on the Results Cartridge.[*85] If present, it will use the strings in the override file instead of the hardcoded default strings.

4.4.10 Edge may generate report forever.

Finding: Due to poor input validation of endorsements, it is possible to create a circular chain of pointers causing the Edge to enter an infinite loop while generating reports.

Implication: The lack of input validation allows an arbitrary read in memory. Therefore, it is possible for a Results Cartridge to cause the Edge to loop forever when generating reports, disabling this capability. This also causes issues with reliability (see Section 3.4.6).

Observations: The candidate file has a circularly linked list of candidates[*86] used by the Edge to generate some reports. This value is read directly from the candidate file on the Results Cartridge.[*87] When generating a report, the loop[*88] exits when the next index arrives back at the starting point. However, it is possible to change the values to point to anywhere (including outside of the candidate array). By forming a cycle that does not include the starting point, the Edge will loop forever.

We were able to confirm this bug on the Edge machines supplied to the Sequoia Red Team.

4.4.11 Provisional voting mode is poorly indicated.

Finding: When voting provisionally on an Edge using a provisional Voter Card, the only indication to the voter that the vote is being cast provisionally is the string "Provisional Voter" on the VVPAT receipt.

Implication: Combined with weaknesses in Section 4.4.7 and Section 4.5.4, official votes may be recorded provisionally without detection by the voter or poll worker. This effectively removes the votes from the official tally. Election officials will be unable to resolve these votes since no voter identification will be associated with the provisional voter ID. This provides an avenue for attackers to both add and remove votes from the official tally.

Section 5.2.2 discusses possible attack scenarios using provisional voting mode.

Observations: We tested this functionality on the Edges supplied to the Sequoia Red Team. The only noticeable difference between voting normally and provisionally was the string "Provisional Voter" on the VVPAT receipt. We confirmed that using the weakness in Section 4.4.7, we can override this string to something innocuous.

4.4.12 Malformed font files allow loading of arbitrary files.

Finding: Using a malformed font file on a Results Cartridge, it is possible to copy arbitrary files outside of the root directory.

Implication: We were able to confirm this functionality on the Edges provided to the Sequoia Red Team by replacing the main binary, which contains the executable software run by the Edge.*[89]

Using this new executable code may be loaded onto the Edge using a Results Cartridge, compromising the Edge and the election. See the discussion in Section 4.4.1 for more on the implications of uploading new executable software.

Observations: The font file*[90] specifies the fonts which need to be copied onto the Edge. It does not validate the font name,*[91] allowing path characters such as "../" to be included in the font name. There are limitations to the attack. The length of the file name is limited,*[92] and must survive the font loading process which attempts to process and rotate the fonts.*[93]

4.4.13 Malformed font files allow modification of arbitrary addresses.

Finding: Using a malformed font file on a Results Cartridge, it is possible to read and write to arbitrary addresses.

Implication: It may be possible to manipulate these pointers such that an attacker can uploaded new executable software on the Edge.

Observations: Fonts are stored in a special structure type which contains several offsets to different tables in the file.*[94] Since offsets are read directly from the file,*[95] the attacker has full control over them. These offsets are used in both read and write operations, giving the attacker both arbitrary read and arbitrary write abilities. This allows the attacker to construct a font file with exploit code, and overwrite a pointer to cause the program flow to execute this code.

The Sequoia Red Team was able to use this weakness to upload new executable code on the Edge from a Results Cartridge without requiring any interaction. Please see the Red Team Report for more discussion, and Section 4.4.1 for implications of this type of attack.

4.4.14 Parsing a Master Ballot Cartridge may cause a buffer overrun.

Finding: A malformed master file on a Master Ballot Cartridge may cause a buffer overflow on the Edge.

Implication: This overflow will cause the Edge to look for and overwrite a specific value outside of the buffer bounds in the heap. An attacker has very limited control over the overflow, however it may be exploitable by a sophisticated attacker to compromise the Edge.

Observations: The Master Ballot Cartridge is placed in the Auxiliary Port while an empty Results Cartridge is placed in the Results Port. When it is inserted, the Edge searches for a special master file on the Master Ballot Cartridge which lists all of the Edges allowed to use the cartridge and the poll ID and name assigned to each Edge. If the correct serial number is contained in this list, the Edge attempts to parse out the poll information. This parsing routine contains a buffer overflow, causing the Edge to look for and overwrite a specific value outside of the buffer bounds in the heap.*[96]

4.4.15 Loading of bitmap files may cause an integer overflow.

Finding: When the Edge attempts to load a Results Cartridge with bitmap image overrides, translation of the image to IMG format contains an integer overflow.

Implication: An attacker may be able to utilize this overflow upload new executable software onto the Edge. See the discussion in Section 4.4.2 for more on the implications of uploading new executable software.

Observations: When loading image overrides from a Results Cartridge, the Edge converts any BMPs into IMGs.*[97] The Edge reads the bitmap font header, which contains the bitmap width and height, directly from the file and then uses these values later.*[98] This allows an attacker to read as many bytes as desired, yielding a heap overflow.

The Sequoia Red Team used integer overflows found in the font file to load new executable code onto the Edge. Loading image files also involves numerous pointers, and may be equally exploitable.

4.4.16 Memory allocation may cause integer overflows.

Finding: Several memory allocation routines used in the Edge show fragile programming practices that are prone to introduce vulnerabilities.

Implication: If one of the allocation routines, a `calloc` alternative that is used pervasively, is called in an unsafe way, an integer overflow could cause insufficient memory allocation, which might result in a buffer overrun. This might crash the Edge, or, if the amount of memory requested as well as the data written to the buffer can be influenced by an attacker, cause the execution of arbitrary code. Several other memory allocation routines*[99] do not check for signedness. This is contrary to robust programming practices and is prone to introduce errors that might crash the Edge.

Observations: The Edge source code contains a custom implementation*[100] of the standard library function `calloc`. It takes two arguments of type `unsigned int`: the block size in bytes and the number of such blocks to be allocated. The function calculates the number of bytes to be allocated as the product of the two, which is assigned to an unsigned int. If the multiplication causes an integer overflow, the size of the allocated memory chunk will be unexpectedly small. The functions*[99] taking signed arguments all pass these on to a custom implementation of `malloc` without performing any range checks, implicitly casting the argument to an unsigned int. This conversion will could result in a very large unsigned value that would cause memory allocation to fail and would likely crash the Edge.

4.4.17 Bar codes on VVPAT receipt are problematic.

Finding: If enabled, the voter-verified paper audit trail includes bar codes which have a numeric representation of voter information and votes cast.

Implication: Bar codes may violate voter privacy, and hinder a voter's ability to verify the paper audit trail accurately represents his or her vote.

Observations: If enabled, bar codes are printed at the bottom of VVPAT receipt to allow for automation of paper ballot recounts. These bar codes consist of several lines of comma-delimited text. The first line is dedicated to machine, election, and voter information such as the Edge serial number, polling place ID, election CRC, and voter ID. This is followed by a representation of the votes cast. Each position in the line is associated with a specific contest, and the value appearing in that position represents the selection made. For all votes except write-ins, this value is a number. The last line of the bar code includes a "signature" (see Section 4.4.5) to prevent bogus receipts from being included in the paper audit.*[101] The amount of information included on the bar code may lead to issues with voter privacy.

Additionally, bar codes represent votes as a series of numbers, with no clear correlation to the candidates and contests on the ballot. This becomes especially problematic if paper audits are conducted using the bar codes only.

Several issues in this section have demonstrated the ability for a malicious user to take over an Edge. Suppose the compromised Edge changes 30% of the votes cast from one party to another. Three mechanisms are in place to attempt and detect such a discrepancy:

- The Voter-Verified Paper Audit Trail (VVPAT)
- Paper Audits

- Pre-Election Logic and Accuracy Testing (Pre-LAT)

A compromised Edge can modify both the votes stored on the Results Cartridge, and the votes recorded in the bar code. The voter is unlikely to notice the discrepancy in the bar code, since the rest of the paper receipt is accurate. If a paper audit is performed using only the bar codes, the tally from the Results Cartridge will match the paper trail. Finally, the compromised Edge can detect Pre-LAT and print the correct bar code.*102

This attack is detectable if a paper audit does not use the bar code. The correct votes may be recovered from the VVPAT. However, a paper audit must occur and recovery costs time, money, and reduces voter confidence.

4.4.18 Behavior of the VVPAT printer is controlled by the Edge.

Finding: Many aspects of the VVPAT printer's behavior are controlled with ASCII commands sent from the Edge. These include the speed at which paper scrolls forward, print density, and scrolling direction[8].

Implication: An Edge with corrupt firmware can make it difficult for a voter to read the VVPAT receipt. The voter is also susceptible to social engineering attacks conducted by the compromised Edge.

Observations: A table of ASCII printer commands to control print direction, scrolling speed, print density, and other behavior appears in the Edge source code.*103

4.4.19 Connection to VVPAT printer is physically unprotected.

Finding: The serial connection to the VVPAT printer is physically unprotected by any tamper-evident seals or locks.

Implication: Anyone with unobserved access to an Edge can insert a device along the serial connection that would capture voter information.

Observations: The serial connection to the VVPAT printer setup by Sequoia in Sacramento was not protected by any tamper-evident seals.

4.4.20 The Edge contains several instances of dead code.

Finding: The Edge contains several instances of code which never gets called.

Implication: Unused features and dead code traditionally receive less scrutiny, and may provide additional avenues of attack in a system. Please see the discussion in Section 3.4.8 for more.

Observations: We have found several instances of function comments which list no comments, indicating the code is inactive.

For example, a function which executes the Watchdog in the Edge has no callers listed in the function comments, and a grep found no calls of the function.*104 Other examples include some CRC functions.*105,*106 These are just a few examples we found, and not an exhaustive search.

4.5 Smartcards

4.5.1 A duplicated Voter Card may be used to vote more than once.

Finding: If a Voter Card is duplicated between the time it is "Activated" and the time it is used, the duplicate card may be used as-is on any other Edge in the precinct.

[8]The sample printer we had access to did not appear to be mechanically capable of scrolling backwards, however.

Implication: A voter may vote more than once, defeating the access control mechanism of the Voter Card.

Observations: An Edge has no way of being notified by another Edge that the card has been used. The data structure is static and not associated with any particular Edge. The process requires access to a smartcard reader, but not knowledge of the secret keys.

4.5.2 A Voter Card may be forged with knowledge of the hardcoded keys.

Finding: Knowledge of Sequoia's proprietary hardcoded keys allows someone to read the contents of Voter Cards, and forge new cards. Note, the Sequoia keys are universal and hardcoded, meaning they can be easily learned by compromising *any* HAAT, Edge, or Card Activator (see Sections 4.2.1, 4.3.1 and 4.4.2).

Implication: A voter may vote more than once, defeating the access control mechanism of the Voter Card. Passable counterfeit Voter Cards may be mass produced.

Observations: The only uniquely identifying data on a Voter Card of any given precinct is the activation time. The Edge checks to make sure a card with that time has not already been used recently. Giving Voter Cards unique activation times is sufficient to fool the Edge into accepting counterfeit cards.

4.5.3 A vendor-supplied Voter Card may be swapped with a voter-supplied smartcard.

Finding: Neither the HAAT nor the Card Activator authenticates a smartcard before programming it, which makes it a valid Voter Card.

Implication: Voters may introduce their own smartcards to be used with legitimate Voter Cards during an election.

Observations: After a voter signs in, a poll worker gives the voter an activated Voter Card. The voter takes this card to an Edge, and casts his or her vote. The voter then takes now deactivated Voter Card back to the poll worker. The poll worker places this card in a pile of cards to be reused and activated for other voters.

Instead of returning the genuine Voter Card, the voter could return his or her own specially programmed smartcard. The Voter Cards appear to have a vendor-specific printed label, which would need to be duplicated for this attack to remain undetected. Otherwise, the smartcard is included in the mix of genuine Voter Cards.

The Card Activator and HAAT attempt to use cryptography to generate active Voter Cards. However, no authentication of the inserted smartcard is securely performed by the Card Activator or HAAT before this creation process occurs. It will attempt to create a valid Voter Card on any smartcard inserted.

4.5.4 A "smart" Voter Card may surreptitiously switch provisional and eligible voter's votes.

Finding: A smartcard masquerading as a genuine Voter Card may be able to switch votes from official to provisional.

Implication: An attacker can introduce a smartcard that changes how a voter's vote is recorded on the Edge, potentially letting unqualified voters cast official votes and forcing qualified voters to cast provisional votes. The only difference the qualified voter would see is a single line on the VVPAT receipt header indicating the Provisional ID number. See Section 4.4.11 for more.

Observations: Section 4.5.3 discusses how a smartcard may be swapped in as a genuine Voter Card. The Voter Cards used by Sequoia are smartcards that act as storage devices, without any processing capability. However, some smartcards come with microprocessors allowing on-card computation. This may allow a smartcard to arbitrarily change its data between the time it is programmed and the time it is read by the Edge. It may also decide to change its data based on the type of reader used to access its contents.

4.6 Insight

4.6.1 The HPX chip is physically replaceable.

Finding: The HPX firmware controlling the behavior of an Insight resides on a removable and re-writable memory chip.

Implication: Anybody with access to an Insight can replace the chip and change the behavior of HPX. A malicious HPX could avoid loading the APX software from legitimate MemoryPacks and instead selectively accept or reject ballots, or count votes incorrectly. The capabilities of the attacker seem only constrained by the size of the chip storing the malicious HPX.

Observations: The HPX resides on an EPROM chip[9] that sits in a socket on the Insight main board. The Insight case is locked, but the wrench-like key appears to be a very simple, so that the lock is likely to be easily picked. The simplicity of the key suggests that it might be the same for all Insights.

4.6.2 APX on MemoryPack has full control of the Insight.

Finding: The APX software loaded from the MemoryPack can execute arbitrary code and thereby have full command of the Insight.

Implication: Malicious modifications to the APX or programming errors in it can cause it to behave in unexpected and undesired ways. In particular, it could selectively reject valid ballots and accept invalid ballots, miscount ballots, slow down the verification process, or write malicious data to the MemoryPack to exploit any weaknesses in WinEDS (see Sections Section 4.1.28, 4.1.29, and 4.1.30).

Observations: By reading the HPX assembly code, we confirmed that the HPX firmware loads and executes the APX code on the MemoryPack (see also Section 4.6.3).

4.6.3 Integrity checking of the HPX and APX is performed by the HPX and APX themselves, respectively.

Finding: The HPX checks its own integrity using a CRC. After it transfers control to the APX, the APX also checks its own integrity using a CRC. In terms of security against tampering, both of these checks are similar in nature to asking patrons at a bar to check their own IDs.

Implication: A modified HPX chip or a MemoryPack with a modified APX could skip their own integrity checks and proceed to take control of the Insight (see Section 4.6.2).

This weakness might cause reliability issues: If some of the data and APX code on a MemoryPack has been corrupted, the integrity check could cause the Insight to crash or exhibit erratic behavior until the MemoryPack is replaced.

Observations: We examined the source code for the HPX and APX and found the routines for APX integrity checking within the APX.

In collaboration with the Red Team, we altered the APX program on a MemoryPack to print a message of our choice on its paper tape and skip its own integrity check. We inserted this altered MemoryPack into an Insight and saw the message printed on the paper tape, which confirmed that our altered APX had control. We observed that the Insight proceeded to print the message "ALL CHECKSUMS O.K." on its paper tape and then accept ballots as usual.

[9]This chip can be erased by exposing it to ultraviolet light, and then reprogrammed.

4.6.4 Integrity checking of the HPX is not adequate to detect tampering.

Finding: The integrity of the HPX is checked by the HPX using a CRC stored on the same memory chip in the Insight that contains the HPX. CRCs do not provide protection against intentional tampering.

Implication: Replacing the chip on which the HPX is stored will not be detected by the Insight. Replacing the chip with one containing malicious firmware instead of HPX would give an attacker full control of the Insight (see Section 4.6.1 for further implications).

Observations: We examined the source code for the HPX and found that the HPX startup routine[*107] calls a subroutine[*108] that checks the CRC.

4.6.5 Integrity checking of the APX is not adequate to detect tampering.

Finding: The integrity of the APX is checked using a CRC stored on the same MemoryPack that contains the APX. CRCs do not provide protection against intentional tampering.

Implication: Anybody can modify the contents of a MemoryPack so that it still results in the same CRC, and will therefore pass the Insight's integrity check.

Observations: We manually traced the source code for the APX, and confirmed that it calls a CRC routine to check itself against CRCs stored on the MemoryPack.

4.6.6 Integrity checking of the election parameters is not adequate to detect tampering.

Finding: The integrity of the election parameters is checked using a CRC checksum stored on the same MemoryPack that contains with the election parameters. CRCs are not computed using a secret key.

Implication: Anybody that has modified the contents of a MemoryPack can compute the CRC of the updated contents and pass the integrity check performed by the Insight.

Observations: We manually traced the source code for the APX from the point where the HPX hands over control to the point where the Insight is ready to accept ballots, and confirmed that it calls a CRC routine to check the election parameters on the MemoryPack.

4.7 MPR

4.7.1 The MPR does not have a physical locking mechanism.

Finding: On the MPR provided to us by Sequoia, there was no mechanism for locking the case to prevent tampering. Removing four screws is sufficient to open the case.

Implication: An individual with access to the MPR can completely subvert its functioning (for example, by replacing components with a soldering iron). By doing so, one could cause any MemoryPacks inserted into the MPR to be rewritten with any data, including altered results, altered election parameters, or altered Insight software. Since WinEDS does not check the integrity of the data read from MemoryPacks 4.1.28, tampering with the MPR is also sufficient to change all of the election results read into WinEDS thereafter via the MPR.

Observations: We asked members of the Red Team to open the case of the MPR and photograph its contents. They opened the case by removing four screws.

4.7.2 The MPR's permanent software resides on a socketed EPROM.

Finding: The permanent part of the MPR software resides on an EPROM (erasable, programmable memory chip) in a socket on the MPR main board.

Implication: An individual with access to an MPR can replace or reprogram the chip in a matter of minutes, thereby completely subverting the functioning of the MPR. Reprogramming the chip is faster and easier than replacing it with a soldering iron. As described in 4.7.1, by subverting the MPR one can gain control over all MemoryPacks later inserted into the MPR.

Observations: We asked members of the Red Team to open the case of the MPR and photograph its contents. They confirmed that case contains a socketed chip labeled "MPR 2.15" with a sticker. Underneath this sticker, they found an EPROM erasing window.

4.7.3 Integrity checking of MemoryPack results by the MPR is not adequate to detect tampering.

Finding: The MPR checks the integrity of the vote counts on the MemoryPack using a CRC. CRCs do not provide protection against intentional tampering.

Implication: Since WinEDS performs no integrity checking of data received from the MPR, and the integrity checking performed by the MPR can fail to detect tampered data, tampered vote counts on a MemoryPack can be loaded into WinEDS without being detected.

Observations: We examined the source code for the MPR software and found that the routine[*109] for transmitting results from the MemoryPack to WinEDS checks CRCs on the MemoryPack data,[*109] but performs no other integrity checking. Based on the source code for the MPR and the Insight, we found that the data structures on the MemoryPack only contain a designated location for a CRC, and found no mention of other integrity checking mechanisms.

4.8 Optech 400-C

4.8.1 WinETP in counting station mode does not encrypt or authenticate its communication with the master WinETP system.

Finding: Communication with the master WinETP system is not encrypted or authenticated, so it can be intercepted and altered in transit by anyone who has gained access to the local network.

Implication: If the Optech 400-C attempts to load an election coding file from the master system over the network, anyone on the same network can intercept and modify the file in transit. Together with the vulnerabilities described in Sections 4.8.5 and 4.8.8, this can potentially be used by anybody with access to the network to execute arbitrary code on the Optech 400-C.

Observations: We examined the source code for WinETP and found that election coding files are transferred over the network using two particular classes[*110] for handling network communication. We could not find evidence of any encryption being performed or encryption keys supplied to this class.

4.8.2 WinETP contains a hardcoded key in the source code.

Finding: There is a 16-byte key hardcoded as a constant in the 400-C source code and labeled as a "hash key."

Implication: Hardcoding keys in software makes keys difficult to keep secret, as the keys will be the same in all copies of the software and can be extracted by examination of the executable program. (However, due to a programming error, the key is not used; see Section 4.8.3.)

Observations: We directly observed this key in the source code. The key resides in the function for authenticating the integrity of the election information saved by the 400-C in the "election coding file".*111

4.8.3 WinETP's hashing routine does not use the key passed in.

Finding: The hashing module in the 400-C software*112 takes a key as a parameter (such as one of the above hardcoded keys), but does not use the key.

Implication: Since no secret needs to be known in order to compute the hash, anyone wishing to tamper with the election coding file can compute a new hash for the tampered file that will be accepted as valid by WinETP. Also see Section 3.2.2.

Observations: We observed that the key is passed to the initialization method for the hashing module*112 and then stored only in two locations (a member variable of the hashing object*113 and a member variable of an object within the hashing object*114) and neither location is ever used.

We compiled and tested the hashing module, and confirmed that the hash output for a given input string is the same regardless of the key supplied.

4.8.4 WinETP stores the hash for the election coding file in the file itself.

Finding: Optech 400-C Security Specification states that the election coding file, which describes the ballots, offices, precincts, and reports, contains a "hash of the file to prevent tampering" (page 2-1). This hash, which uses no key as mentioned above, is stored in the file itself.

Implication: Anyone wishing to supply a tampered election coding file to WinETP can easily compute its hash (see Section 4.8.3). Including the newly computed hash is sufficient to make any tampered file pass integrity checking without detection.

Observations: We examined the routine for hashing the election coding (EC) file.*111 and found that it is called in exactly two places: a routine that validates the EC file*115 and a routine that writes the EC file header.*116 The latter routine writes the EC file header to the same file that it passes to the EC hashing routine.*111 The EC hashing routine tells the hashing module to store the resulting hash in a field inside the EC file header structure.*117 The input to be hashed includes the file header and the rest of the EC file contents after the file header.

4.8.5 WinETP does not check the integrity of the election coding file when the file is loaded over the network.

Finding: Although WinETP checks the hash in the election coding file when it is loaded from disk, it neglects to perform this check on an election coding file loaded over the network.

Implication: Since network communication is not encrypted or authenticated, an attacker can modify the election coding file in transit over the network, and tampering will not be detected because the integrity check is not performed.

Observations: We observed that the routine for hashing the EC file is called in exactly two places in WinETP (see Section 4.8.4), one of which writes the hash and one of which checks the hash. The routine that checks the EC file hash*115 is called only by one function, the function that opens an election file.*118

This latter function to open an election is called in only three places. In two of these places*119 it is called immediately after a call to the routine that writes a newly computed hash into the EC file header.*116 The third call site is in the method that implements the File → Open command.*120 This method decides whether to call the function to open an election file from a disk*118 or the function that opens an election file over the network.*121 Therefore, the code path for opening an election file over the network never calls the routine*115 for checking the hash.

4.8.6 Integrity checking by WinETP of the precinct results file is not adequate to detect tampering.

Finding: WinETP uses a CRC to check the integrity of precinct results files. CRCs are not secure to detect intentional tampering.

Implication: Anyone with write access to the floppy disks or other writable storage media on which precinct results files are stored can alter the precinct results stored by WinETP, and WinETP will not detect the tampering or notify the operator that anything is wrong.

Observations: We examined the source code for WinETP and found that the function that reads the precinct results calls*[122] a function to compute a CRC.*[123] We found no evidence that any integrity check secure against intentional manipulation is performed.

4.8.7 The R-Code interpreter reads from an array without checking the array index, which is dictated by R-Code instructions.

Finding: An R-Code program can cause the R-Code interpreter in WinETP to read data beyond the allocated size of an array.

Implication: A malformed R-Code program can cause WinETP to crash.

Observations: An R-code instruction that reads a counter*[124] takes a 12-bit argument which is used to index an array*[125] whose allocation size is determined by the election header.*[126] For some arguments and election headers, this is likely to result in an index out of bounds.

4.8.8 The E-Code interpreter writes to an array without checking the array index, which is dictated by E-Code instructions.

Finding: An E-Code program can cause the E-Code interpreter in WinETP to write arbitrary data beyond the allocated size of an array, thereby controlling a region of memory 16 384 bytes long on the heap starting at the beginning of the array. The interpreter executes an E-Code program from the election coding file when a ballot is scanned.

Implication: An attacker who is able to modify or supply the election coding file that contains the E-Code program can cause heap corruption when a ballot is scanned and may be able to take control of the Optech 400-C computer. Since, in counting station mode, the election coding file can be intercepted and altered in transit over the network (see Section 4.8.1), anyone who gains access to the network the 400-C is connected to may be able to inject an E-Code program, corrupt the heap, and thereby take control of the 400-C.

Observations: We examined the source code for the E-Code interpreter*[127] and found that one of the available opcodes loads an arbitrary value into a 16-bit register.*[128] Another opcode*[129] stores the value of this 16-bit register into a 2-byte element of an array indexed by a 13-bit argument to the opcode. We did not find any code that checks this 13-bit index to ensure that it is in range.

We found that this array is allocated by a routine*[130] that receives the allocation size of the array as an argument, and that this argument is determined by election parameters*[131] in the election coding file. The E-Code instructions also reside in the election coding file.

We compiled the E-Code interpreter (adding the necessary stubs and definitions so that it would compile separately) and created a simple E-Code program to store a value at an arbitrary index. We ran the program with the interpreter and confirmed that it would store any 16-bit value at any index from 0 to 8191 in the array of unsigned 16-bit integers.

4.9 Platform

4.9.1 Windows will execute autorun files on U3 USB smart drives.

Finding: The Windows laptop provided to the Red Team executes autorun files from U3 USB start drives automatically.

Implication: U3 USB smart drives can take control over the Windows machines in which they are inserted, including WinEDS computers and Optech 400-C machines. If U3 USB smart drives are used and reused to prepare HAATs for an election, a compromised HAAT can take control over WinEDS.

Observations: Most Windows machines are configured by default to automatically execute autorun files on CD/DVD devices. U3 USB smart drives are a special type of USB memory stick which pretends to be a CD/DVD device to add autoplay capability under Windows, and can be used as normal USB sticks if necessary. The autorun feature may be enabled and disabled by setting the `NoDriveTypeAutoRun` registry key in Windows. The Red Team confirmed that the laptop provided by Sequoia was configured to automatically execute the autorun file from a U3 USB smart drive.

USB sticks are normally used for the HAAT firmware update process (see Section 2.1.2), and to prepare the HAAT for an election (see Section 2.1.2). Depending on the county procedures in place, a single USB device may be reused to program multiple HAATs and across multiple elections. Due to the backwards-compatibility of U3 USB smart drives, they may replace USB sticks in this process.

Section 4.2 details multiple possible avenues for compromising a HAAT. Section 4.2.9 discusses how a compromised HAAT may be able to write to the USB device used for preparing elections.

Therefore, a patient attacker may be able to compromise WinEDS by compromising a HAAT. Once compromised, the HAAT must wait for the next election cycle. When a U3 USB device is inserted to prepare the HAAT for an upcoming election, it can add a specially crafted autorun file to the USB key. The next time this key is inserted into the WinEDS machine, it may automatically run the autorun file, and end up compromising the WinEDS election management system.

4.9.2 WinEDS Windows version and configuration unspecified

Finding: The precise version of Windows over which WinEDS runs is not specified, nor are the elements that should be included in or excluded from its configuration.

Implication: Because the precise platform and configuration is not specified, there is no way to assure that potential security vulnerabilities introduced by interactions between WinEDS and various Windows bugs and features do not occur.

Observations: Conflicting instructions are given in different documents. A test document[10] asks to test only against a specific version of Windows ("Windows XP with Service Pack 2" for WinEDS clients and "Windows Server 2003" or "Windows XP with Service Pack 2" for the WinEDS database server). But in most other documents, any of many different versions of Windows are permitted ("Windows 98, Me, 2000 or XP"[11] for workstations and "Windows 2000 or .Net Server with latest service pack"[11] for the WinEDS database server). In particular, neither of the platforms recommended in the Installation Guide for the database server is suggested in the Test & Verification Specification.

[10]WinEDS 3.1 Test & Verification Specification, document version 1.02, January 2006
[11] WinEDS 3.1 Installation Guide, document version 0.04, January 2006

CHAPTER 5

Combined Implications and Attack Scenarios

Many of the weaknesses described here can be combined in a way that amplifies the effects of what might otherwise be relatively small security breaches.

Figure 5.1 illustrates how subversion of one component can give the potential to gain control of other components. Malicious software can travel along the arrows in this diagram to infect other components in the Sequoia system, possibly over the span of more than one election.

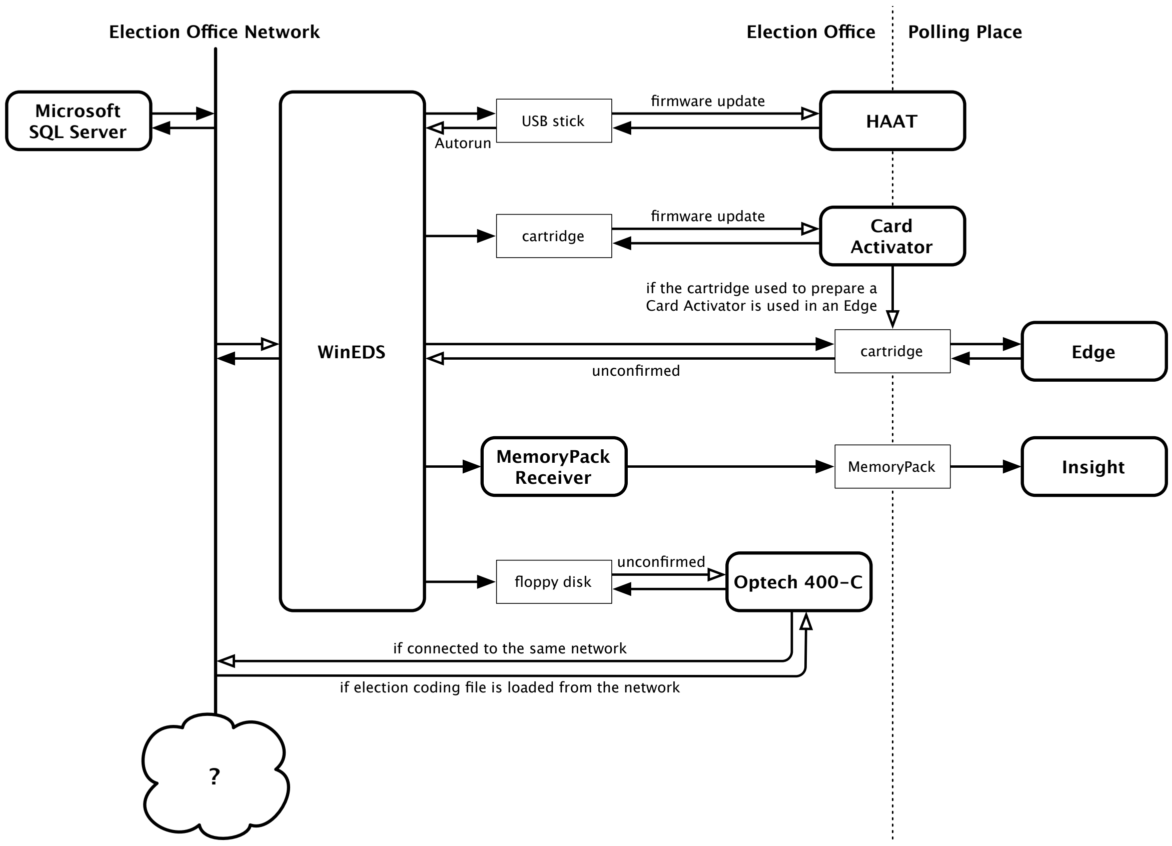

Figure 5.1: An arrow from component A to component B means that A can take control of B during typical usage. A hollow-headed arrow from A to B means that A may be able to take control of B under some circumstances (explained in the text).

The arrows pointing *to* the election office network illustrate that components on the network can eavesdrop on network traffic and modify it in transit, since network transmissions are not encrypted or authenticated (see Sections 4.1.2 and 4.8.1). The arrows leading *from* the election office network illustrate that anyone on the network can take control of the WinEDS database server (see Sections 4.1.2 and 4.1.5), may be able to take control of WinEDS (see Section 4.1.13), and may be able to take control of an Optech 400-C if it is attached to the same network (see Sections 4.8.1, 4.8.5, and 4.8.8).

Note in particular the loops in the diagram: between the USB stick and the HAAT, between the cartridge and the Card Activator, and between the cartridge and the Edge, and between the floppy disk and the Optech 400-C. Each of these loops is a potential vector for viral infection.

- A USB stick can take control over a HAAT if the USB stick is used to perform a software update (see Section 4.2.5). If any USB stick that has been used with a subverted HAAT is then used to update other HAATs, it can transmit the infection to the others.
- A cartridge can take control over a Card Activator if the cartridge is used perform a firmware update (see Section 4.3.6). If any cartridge that has been placed in a subverted Card Activator is then used to update other Card Activators, it can transmit the infection to the others.
- A cartridge can take control over an Edge when it is used for voting (see Sections 4.4.12 and 4.4.13). An infected Edge can then infect any cartridge that is placed in it, and if that same cartridge is then used in other Edges without being erased, it can infect them as well. If the PCMCIA cartridge used for preparing a Card Activator is then used in an Edge, a subverted Card Activator can initiate this infection.
- An election coding file on a floppy disk (or other removable storage, such as a USB stick, CD, or DVD) may be able to take control over an Optech 400-C if the Optech 400-C loads an election coding file from the disk and then scans ballots (see Sections 4.8.3, 4.8.4, and 4.8.8). An infected Optech 400-C that can then infect any writable media that are placed in it, and if that same media is used to transfer an election coding file to other Optech 400-Cs, it can infect them as well. An infected Optech 400-C may also be able to infect other Optech 400-Cs over a local network if they load election coding files over the network (see Section 4.8.5).

The arrows leading toward WinEDS are of particular note:

- A USB stick, if it has the U3 feature, could take control over WinEDS when inserted into a WinEDS client (see Section 4.9.1). A USB stick can become infected by a subverted HAAT. If USB sticks (not shown in the figure) are also used to transfer files to and from the 400-C, a USB stick can also become infected by a subverted 400-C.
- A Results Cartridge might be able to take control over WinEDS (see Section 4.1.25), though we have not confirmed this method of attack.
- Anyone on the internal network can tamper with communication from the database to WinEDS. Exploiting the handling of a format string from the database may allow an attacker to take control over WinEDS (see Section 4.1.13).

As indicated by the arrows that lead from WinEDS to all the other components, taking control of WinEDS may allow an attacker to control all other election equipment.

5.1 Avenues of Attack and Recovery

The way certain attacks can be combined also affects the ability of election administrators to detect and recover from them. Figure 5.2 summarizes some of the significant security implications of compromised components, organized by the access an individual must abuse to carry out an attack.

A person with access to...	**can potentially...**	Overall election reporting can detect that tampering has occurred	Analysis of the affected component can confirm that tampering has occurred	All affected votes can be recovered by a hand recount
a WinEDS computer	take control of all election data	–	–	–
	take control of any Edge in which a Results Cartridge prepared on this computer is used for voting, and retain control after the cartridge is removed	–	✓	–
	take temporary control of any Insight in which a MemoryPack prepared on this computer is inserted	–	✓	✓
any WinEDS account	erase any table in the election database using WinEDS, even if this WinEDS account has no such access rights	–	✓	✓
	modify anything in the election database using a SQL client, even if this WinEDS account has no such access rights	–	–	–
the Election Office network	sniff database passwords and then modify anything in the election database using a SQL client	–	–	–
an Edge	take full control of this Edge (which includes all future behavior, ballot presentation, and data on cartridges inserted thereafter)	–	✓	–
a single Results Cartridge	take full control of any Edge in which this cartridge is used for voting, and retain control after the cartridge is removed	–	✓	–
	modify any votes stored on this cartridge	–	✓	✓
	add votes to all other precincts when this cartridge is loaded	✓	✓	✓
a HAAT	take full control of the HAAT (which includes all future behavior, card activation, and data on USB sticks inserted thereafter)	–	✓	–
	discover the Sequoia System encryption and signature keys, HAAT passwords, and transmission keys	–	✓	–
a USB stick for a HAAT	take control of any HAAT for which this USB stick is used to perform a firmware upgrade	–	✓	–
	take control of any WinEDS computer in which this USB stick is inserted, if it is a U3 USB stick	–	✓	–
a Card Activator	take full control of the Card Activator	–	✓	–
	take full control of an Edge by way of a Results Cartridge	–	✓	–
	take control of other Card Activators during a firmware update	–	✓	–
a single Voter Card	cryptanalyze the Sequoia System encryption key, used in future elections	–	–	–
	vote multiple times	✓	✓	–
an Insight	take full control of the Insight (which includes all future behavior and data on MemoryPacks inserted thereafter)	–	✓	✓
a single MemoryPack	take temporary control of any Insight in which this MemoryPack is inserted, until this MemoryPack is removed	–	✓	✓
	modify any votes stored on this MemoryPack	–	–	✓
	supersede the votes on a MemoryPack in another precinct, if this MemoryPack is loaded before the other MemoryPack	✓	✓	✓
a MemoryPack Receiver	take full control of this MPR (which includes all future behavior and data on MemoryPacks inserted thereafter)	–	✓	✓
	modify data on any MemoryPack inserted into this MPR	–	✓	✓
	take temporary control of any Insight in which such a MemoryPack is later inserted	–	✓	✓
	modify any votes loaded from MemoryPacks via this MPR	–	✓	✓
an Optech 400-C	take full control of the 400-C (which includes all future behavior, ballot scanning, and data on floppy disks inserted thereafter)	–	–	✓
a floppy disk used with a 400-C	modify any votes that were stored on the disk by a 400-C	–	–	✓
	take control of any 400-C that scans ballots after loading election files from this disk	–	✓	✓

Figure 5.2: Significant security implications of attacks against various components of the Sequoia system (based on the configuration provided to us and the Red Team by Sequoia).

The three columns on the right side of the table highlight significant aspects of post-election detection or recovery. Check marks in these three columns indicate attacks for which:

- **Overall election reporting can detect that tampering has occurred.** These attacks are detectable even when one is not specifically looking for a particular problem. However, it may still be difficult to identify which component has been affected and to recover from tampering. Attacks without check marks in this column are of particular concern, since, in the absence of specific suspicion and targeted analysis, such attacks will go undetected.

- **Analysis of the affected component can confirm that tampering has occurred.** These attacks can be detected by intentionally analyzing a specific suspected component for a specific suspected attack. It is possible to confirm that something is wrong, but only if one knows where to look and has the right tools. Attacks without check marks in this column are not detectable without recounts, and even recounts may be insufficient in some cases.

- **All affected votes can be recovered by a hand recount.** Attacks without a check mark in this column can cause votes to be corrupted or lost in such a way that the true count can never be recovered. Most of these are attacks that subvert the presentation of the ballot; if a voter is not shown the correct ballot, no true record of their vote will exist to be recounted.

The VVPAT provides one illustration of the difference between detection in general and confirmation of a suspected attack. A VVPAT offers a second copy of an Edge's votes[1] against which reported results can be compared. But if the votes on the Results Cartridge have been changed, the tampering will escape detection unless someone suspects a problem, decides to audit that specific Results Cartridge, and examines the corresponding Edge's VVPAT printout.

Many of these attacks establish lasting control over a device by replacing its internal software. As noted in Section 3.1.3, it can be difficult to determine whether the software on suspected devices has been subverted. This software has complete control over the machine, including the "firmware update" process by which this software is normally upgraded or replaced. Consequently, one cannot rely on the normal update process to repair corrupted software; once corrupted, it cannot be relied upon to replace itself.

5.2 An Example: Provisional Voter Attacks

In this section, we describe two example attack scenarios that illustrate the leverage an adversary can gain from exploiting weaknesses in combination with one another. These examples are not intended to illustrate particularly realistic or worst-case scenarios, but simply to demonstrate how the composition of weaknesses can greatly aid the attacker. This section does not attempt to exhaustively (or even representatively) identify a full range of scenarios an attacker might attempt.

In particular, we use for the purposes of our examples here the weaknesses in the Edge (Section 4.4), HAAT (Section 4.2), and Card Activator (Section 4.3) regarding provisional voting. We discuss some of combined attack vectors arising of these weaknesses, along with an example of a voter-based attack.

For these scenarios, suppose there is an upcoming election in which two parties, the Montagues and Capulets, are competing against each other in a close race (that is, one in which a small number of precincts are likely to determine the outcome). Let Mercutio be someone who wants to see the Montagues win the election, and is willing to do what is necessary to see this happen.

5.2.1 Scenario 1: Voter-Based Smartcard Attack

An important aspect of this attack scenario is the ability to forge a Voter Card. This requires both the knowledge of the Voter Card key and access to a valid Voter Card (see Section 4.5.2).

[1]The Edge also stores a second copy of recorded votes on its internal Audit Trail disk. Like the VVPAT, the internal copy can confirm a suspected discrepancy, but not until it is examined.

We assume our adversary is able to discover the hardcoded key used to encrypt Voter Cards. (As noted, this requires temporary access to just one Edge, HAAT, or Card Activator anywhere in the world to extract the key and compromise the election process for all that use these systems.)

Similarly, multiple avenues exist for procuring a Voter Card from a previous election. For example, there are palmtop-sized devices that can read a smartcard. A voter could place the device in a pocket or purse and bring it into the polling booth. Within a reasonable timeframe, the voter could copy the data from the Voter Card, cast a vote, and return the Voter Card to the poll worker. Alternatively, it may be possible to "forget" to return the Voter Card after voting.

The Attack

Suppose Mercutio is capable of forging Voter Cards, and creates a handful of smartcards which are programmed to swap 15% of the votes cast with the card to provisional. On election day, Mercutio chooses a handful of polling places which vote primarily for the Capulets. He shows up first thing in the morning, and asks to vote provisionally. Before leaving, he swaps in one of his own smartcards. Due to the poor indication of provisional voting mode on the Edge, most voters fail to notice the change (see Section 4.4.11).

The Effect

The number of votes that are prevented from being included in the final official tally from locations dominated by the Capulets is enough to swing the election. The attack may be detectable, as a large number of provisional votes will be recorded. However, to protect voter privacy, these provisional votes were not linked to voter identities, they cannot be resolved.

5.2.2 Scenario 2: Compromised Edge Attack

Suppose Mercutio is able to obtain a position as a poll worker. Using weaknesses such as those in Sections 4.4.12, 4.4.13, 4.4.14, 4.4.15, and 4.4.16, Mercutio is able to create a Results Cartridge that uploads custom executable software onto the Edge.

Alternatively, if Mercutio is able to get a temporary position with WinEDS access, he can use weaknesses in Sections 4.1.3, 4.1.5, 4.1.6, 4.1.7, 4.1.8, and 4.1.15 to gain access to the WinEDS database. Then, he could change the election definition entries in the database to potentially cause WinEDS to create Results Cartridges that upload his custom code, compromising all Edges in the election.

The Attack

Mercutio creates new executable software for the Edge that switches 30% of votes for the Capulets to the Montagues. To prevent detection by the voter-verified paper audit trail, the Edge displays the correct vote on the receipt. However, it indicates in the receipt header that the vote is provisional, and then quickly scrolls past the string (see Section 4.4.18). The voter is unlikely to pay attention to the header with the Edge serial and date information, and focuses on the actual vote entries. (Alternatively, the Edge can void VVPAT records for the Capulets after the voter walks away. The Red Team took this approach, as described in their report.) After the voter has left, the Edge prints an extra vote for the Montagues to replace the Capulets vote which was recorded provisionally.

After the polls close and the final reports are requested, the Edge can purposely fail to report the provisional votes and ensure the event logs are innocuous. Additionally, the Edge can detect pre- and post-election testing and behave normally during tests.

The Effect

The number of votes switched may be sufficient to swing the election. The official number of votes will match the number of voters that signed in at the polling locations, since every added vote for the Montagues is offset by votes for Capulets that were instead recorded provisionally.

The Edge can hide the provisional votes cast in the reports, avoiding any extra suspicion. Therefore, no one will attempt to resolve these provisional votes.

Should a 1% recount occur, the number of “official” non-provisional votes on the paper audit trail will still match the number of voters. If only a few Edges are compromised and are not included in the 1% recount, the extra provisional votes on the audit trail will never be discovered.

Even if detected, the compromised firmware means that the Edge audit trail and paper trail cannot be trusted. The recovery options may be to just throw out those votes, or conduct a new election.

CHAPTER 6

Conclusions

We found pervasive security weaknesses throughout the Sequoia software. Virtually every important software security mechanism is vulnerable to circumvention. The integrity of elections conducted with the system depends almost entirely on the physical security of the equipment and the procedural controls under which election operations are conducted.

Whether the software vulnerabilities we describe can be compensated for with procedural and physical security mitigations depends on a range of factors, most of which were beyond the scope of this study. However, we caution that mitigation will place considerable additional pressure on physical security features (such as locks and seals) and human procedures (such as two-person control by poll workers). Many of the physical security features and procedures typically used with the Sequoia system appear to have been engineered under the assumption that the underlying software is considerably more secure than it actually is, and thus may not provide sufficient protection in light of the vulnerabilities discussed here.

Designing robust, practical, and effective procedures that substantially reduce the risks identified in this report would itself be a very complex task, requiring a broad range of computer security, physical security, legal, and operational elections expertise. As a starting point, we attempted to identify mitigation strategies for the vulnerabilities we discovered. Unfortunately, we were unable to find practical strategies that reliably prevent exploitation of some of the system's weaknesses. Fixing some of the problems will require substantial changes to the software and the architecture.

In fact, we are not optimistic that acceptable practical and secure mitigation procedures are even possible for some of the Sequoia system's components and features, at least in the absence of a comprehensive re-engineering of the system itself.

The problem is compounded by the inter-related nature of many of the vulnerabilities and the relative ease with which certain attacks can be carried out. As the table in Figure 5.2 summarized, even brief exposure of many system components to an attacker can have ramifications beyond the components themselves.

Of particular concern is that virtually every software mechanism related to counting votes is exposed, directly or indirectly, to compromise through tampering with equipment that is deployed in the field. In many cases, tampering sufficient to cause compromise requires only brief physical access and may leave behind little or no evidence.

We are regrettably unable to suggest with confidence any comprehensive strategy for mitigating the vulnerabilities in the Sequoia system that simultaneously provides a high assurance of security, maintains accessible DRE voting, and substantially incorporates existing hardware and software.

Postscript

Public confidence in the electoral process is perhaps the most elemental measure of a democracy's health. Now more than ever, that confidence depends not only on our trust in the institutions and officials that run our elections, but on our trust in the increasingly complex (and often invisible) machinery of voting itself.

We applaud the California Secretary of State for seeking independent review of the trustworthiness of the voting technology on which her state depends and are grateful for the privilege of serving such an unquestionably worthy civic goal.

Among the most vexing problems in computing is the difficulty of engineering reliable software. No general method exists for building good software, or even for determining whether programs behave as we intended.

Computer security is thus an exasperatingly complex and imprecise enterprise. There is no "magic bullet" for assuring that crucial systems can withstand adversaries intent on circumventing them. We are thwarted not only by our inability to write reliable software, but by the silent determination and ingenuity of those who seek to exploit the inevitable flaws that escaped our notice. And we neglect the human factor only at great peril, since perfect security that fails to fit within the constraints of the real world is worse than irrelevant.

The only approach known to work (if all too imperfectly) is relentless and wide scrutiny, applied across a system's architecture and throughout every phase of its construction and deployment.

This review will therefore have meaningful impact only if it serves as a step toward an ongoing public dialog, engaging system vendors, local election officials, outside experts and voters.

We are eager to continue the discussion.

APPENDIX A

Threat Model

The first step in security analysis of a system is to define the *threat model*. The threat model for a system is intended to describe the goals an attacker might have (e.g., to manipulate the vote count) and the types of attackers that might attempt to attack the system (e.g., voters, poll workers, etc.) as well as the capabilities available to each type of attacker. It is equally important to describe the threats that are out of scope for the analysis. This study was chartered to consider the security of voting systems proper only, not that of California's entire election system, and therefore many possible attacks are out of scope for this report.

A.1 Reference Model

In order to simplify the analysis, we assume a common reference model, which distills what are hopefully the essential features of all of the voting systems involved in this study. The system consists of the following components.
In the polling place:

- Management stations (MS)
- Direct recording electronic (DRE) voting machines, attached to Voter Verified Paper Audit Trail (VVPAT) printers
- Paper ballot optical scanners (opscan)

At Election Central:

- An election management system (EMS)
- High-speed paper ballot optical scanners (e.g., for absentee votes)

The election can be thought of as proceeding in three stages (for simplicity, we are ignoring early voting centers):

Pre-voting:: Before election day, election officials use the EMS to set up the election. They generate the ballot definition(s) and record them onto media for distribution. During this stage, voting machines are also prepared and distributed to polling places.

Voting:: On election day, voters arrive at the polling place, are verified as being permitted to vote, and cast their ballots.

Post-voting:: After the polls are closed, the votes are tallied, the official canvass (including the one percent manual recount) is performed, and the results are certified.

The relationship between these components is shown in Figure A.1.

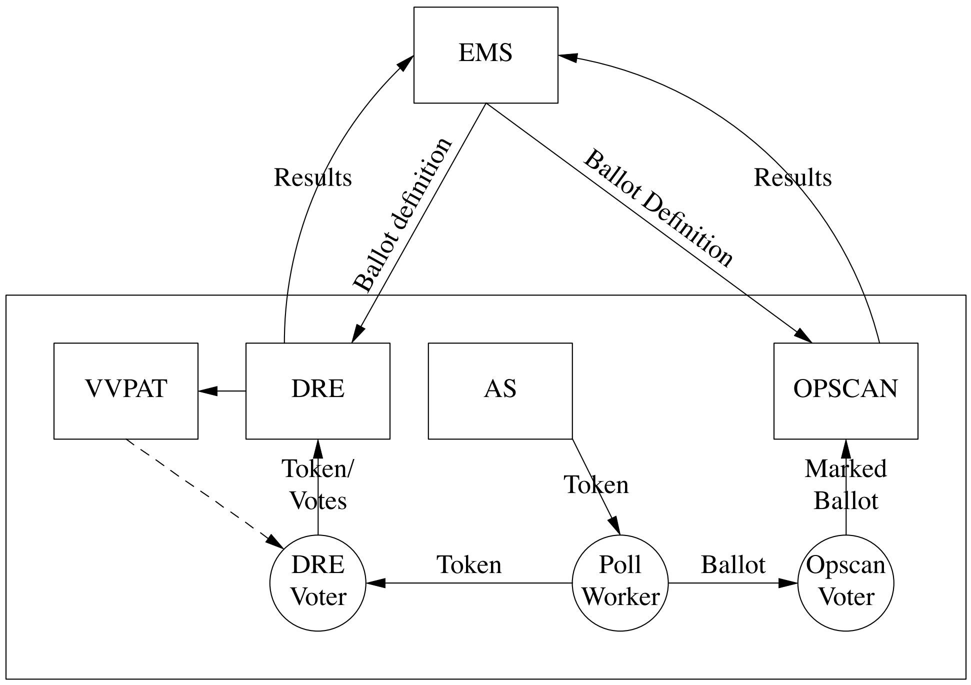

Figure A.1: Reference architecture

A.1.1 Pre-Voting

In the pre-voting phase, election officials need to:

- Create the election definition.
- Print the paper ballots used for optical scan systems.
- Reset the local voting equipment and potentially load the election definitions.
- Distribute the local voting equipment to the polling places.

There is some variation among voting system vendors, but in general the EMS is used to create the ballot definition files. These are then loaded onto some memory card/cartridge and/or directly onto the voting machines. The vote counters in the machines are reset, the internal clocks are set to the correct time, and the machines are then shipped out to the local polling places or provided to poll workers to be hand-carried to the polling place. The ballot definition files must be protected from tampering and so memory card/cartridges are generally distributed with some physical security measures, either by sealing them into the local equipment at the central office or by distributing them in a sealed package. Seals may take the form of tamper-evident tape or may take the form of metal or plastic loops, individually numbered, which once installed can only be removed by cutting them.

A.1.2 Voting

Once the polls open on election day, voting can begin. The exact details of the voting phase differ with the technology and manufacturer in use, but there is a fair amount of commonality across manufacturers within a given technology (DRE, opscan).

Optical scan machines. Opscan voting can most easily be thought of as machine-counted paper ballots. All the procedures here could be replicated by humans with appropriate audit controls.

When an opscan voter enters the polling place, and is verified as permitted to vote, he or she is given a blank paper ballot. He or she marks the ballot with a pen or pencil and the ballot is then mechanically counted with an optical scanner. This can be done either locally or centrally. In the local case, the precinct has a scanner which counts the ballots as they are inserted. In general, the voter personally inserts the ballot into the scanner. Precinct-based optical scanners can detect "overvoting" and reject such ballots, giving the voter an opportunity to correct the error. At the end of the day, the scanner's electronic records are then sent back to the county for aggregation with records from other precincts. The paper ballots are also sent back, for auditing and recounts.

With central counting, untabulated ballots in their original ballot box are sent back to Election Central (i.e., the county's election headquarters) where they are tabulated with a high-speed scanner under supervision of the election officials. Central and precinct-based tabulation may be mixed in a variety of ways. Central tabulation is naturally used for absentee ballots and can also be used for audits and recounts of precinct-cast ballots, whether or not they were originally tabulated in the precinct.

DRE machines. DRE voting is fundamentally different from opscan voting. Instead of entering their vote on paper, the voter uses a computer-based *graphical user interface* (GUI). Once the vote has been "cast," an electronic records of the vote is stored locally, in the DRE machine (and, in the case of the Hart system, a copy is also stored in the management station). At the end of the day, these electronic records may either be extracted from the DRE machines on memory cards, or may be transmitted via modems, or the DREs themselves may be transported to Election Central. In any case, the EMS will collect the electronic records from each precinct and will tabulate them electronically.

As with opscan voting, in DRE voting, the voter enters the polling place and first establishes his or her eligibility to vote. However, some method must be used to limit authorized voters to casting only one vote. In all the DRE systems in this study, the poll worker uses an administrative device to issue the voter a token of some sort. The voter may then take this token to any voting machine and vote once. With Diebold and Sequoia, the token is a smartcard. With Hart, it is a four-digit "Access Code."

Once the voting machine is activated, it takes the voter through each contest and allows him or her to select candidates. DRE machines automatically forbid overvotes (too many votes cast in a contest) but not undervotes (too few choices cast in a contest). The voter is then presented with an opportunity to review his ballot and then commits to it ("casts" it), at which point it is recorded to local storage.

In California, a *voter-verified paper audit trail* (VVPAT) is required. On all the machines under study, this takes the form of a sealed printer with a continuous spool of paper attached to the DRE. Before the voter confirms his ballot, a summary is printed out on the printer and displayed through a glass window. Once the voter accepts or rejects the ballot, an appropriate indication is printed on the VVPAT record. When the voter casts the ballot, the VVPAT record is marked as accepted and scrolled out of sight. Because the paper scroll is held behind glass, it becomes more difficult for a voter to "stuff" additional ballots into the machine or to take the record of their vote home, incorrectly, as "receipt."

Once the election is over, the local results are transmitted to the Election Central, typically by shipping some form of removable memory device from the voting machine. The VVPAT paper rolls, perhaps still sealed in their printers, are also sent to the Election Central to be used in the legally required 1% audit.

In larger counties, many vendors offer the ability to establish "regional processing centers." Election results are delivered by courier from the precincts to the centers, aggregated, and then communicated back to Election Central via electronic means (modems, Internet connections) or via courier.

Typical deployments. There are two common models for deploying this equipment in the polling place. In the DRE-only model, every polling place contains some number of DREs, most or all voters vote on the DREs, and there are no optical scan machines in the polling place. In the hybrid model, every polling place contains one optical scan machine and one or more DREs; voters may have the option whether to vote on paper ballots or using the DRE, or the DRE may be reserved for voters with disabilities and all others may vote on paper ballots. In California, each county determines which model is most appropriate for their needs.

A.1.3 Post-Voting

After the election is over the election officers need to do (at least) three things:

1. Tabulate the uncounted opscan and absentee ballots.
2. Produce combined tallies for each contest based on the records received from the individual precincts and the tallies of centrally counted ballots.
3. Perform the official canvass. This may involve reconciling the number of voters who have signed in against the number of ballots cast, performing the statutory 1% manual recount, and other tasks.

The first two tasks are relatively straightforward, though it's important to note that the second task is typically performed based on electronic records. In the most common case, the precincts send back memory cards containing the election results and those cards are added directly to the tally without any reference to the paper trail (except, of course, for centrally tabulated opscan and absentee ballots).

The 1% manual recount compares the paper records for a given set of votes to the reported vote totals. In the case of opscan ballots, this means manually assessing each opscan ballot. In the case of DREs, it means manually assessing the votes on the VVPAT records. Note that while in principle the opscan tally may differ slightly from the paper ballots due to variation in the sensitivity of the mark/sense scanner, the VVPAT records should exactly match the DRE records.

A.2 Attacker Goals

At a high level, an attacker might wish to pursue any of the following goals or some combination thereof:

- Produce incorrect vote counts.
- Block some or all voters from voting.
- Violate the secrecy of the ballot.

An attacker might wish to pursue any of these goals either generally or selectively. For example, an attacker might target a certain subset of voters (e.g., registered Republicans, or voters who live within a certain geographic area) to attack. Also, the ability to determine how an individual voted can be used to enable vote buying or voter coercion, either individually or en masse.

A.2.1 Producing Incorrect Vote Counts

The most obvious attack on a voting system is to produce incorrect vote counts. An attacker who can cause the votes to be recorded or counted in a way that is different from how people actually voted can alter the outcome of the election. There are a number of different ways to influence vote counts, including:

- Confuse voters into voting differently than their intent.

- Alter the votes, within the computer, before they are recorded.
- Alter votes in the vote storage medium, after they were originally recorded.
- Corrupt the vote tabulation process.

Which attacks are feasible depends on the attacker capabilities.

A.2.2 Blocking Some or All Voters from Voting

Two classical techniques to influence election outcomes, regardless of the election technologies in use, are voter education / encouragement (i.e., "get out the vote") or voter suppression. If we limit the context to attacks specifically on voting systems, an attacker might mount a similar attack by making it very difficult for certain classes of voters to vote. For example, an attacker might:

- Arrange for some subset of machines to malfunction, possibly those in precincts in which voters of the opposite party are overrepresented.
- Arrange for machines to selectively malfunction when voters attempt to vote in a certain way.
- Arrange for all the machines in an election to malfunction.

The first two of these attacks are selective attacks and could be used to influence the outcome of an election. A more global attack is primarily useful for denial-of-service or to invalidate an election, but could potentially be used for extortion as well. These attacks would generally require tampering with the software within the voting machines.

A.2.3 Violating Ballot Secrecy

An attacker who cannot influence voting directly might still be able to determine how individuals or groups voted. There are two natural applications for this kind of attack:

- Vote buying / voter coercion
- Information gathering

In a vote buying or voter coercion attack, the attacker pays individual voters to vote in a specific way or threatens retribution if they do not. However, in order for such an attack to be successful, the attacker needs to be able to verify that the voter in fact voted the way he agreed to. Note that the buyer does not need to be able to determine with absolute certainty how a voter voted, but merely needs a high enough confidence that defection by bribed voters becomes unattractive. The primary benefit of traditional secret-ballot voting is that that it allows the voter to cast a vote in complete secrecy, defeating the attacker's ability to validate a voter's cast ballot and thus making vote buying or coercion unattractive.

Another possible reason to violate voter secrecy is to gather information on a large group of people. For example, an attacker might wish to determine which voters were sympathetic to a particular political party (but had not registered with it) and target them for investigation, surveillance, or even targeted mail or telephone solicitations. Alternately, an attacker might wish to publish a given voter's votes in an attempt to influence public opinion about them. In either case, ballot secrecy is required to block these attacks.

A.3 Attacker Types

A voting system can be subject to attack by a number of different types of attackers with different capabilities and who will therefore be able to mount different kinds of attacks. We consider the following broad classes of attackers, listed roughly in order of increasing capability.

Outsiders: have no special access to any of the voting equipment. To the extent that voting or tabulation equipment is connected to the Internet, modems, wireless technologies, and so forth, an attacker can mount network or malware-based attacks. Outsiders may also be able to break into locations where voting equipment is stored unattended and tamper with the equipment.

Voters: have limited and partially supervised access to voting systems during the process of casting their votes.

Poll workers: have extensive access to polling place equipment, including management terminals, before, during, and after voting.

Election officials: have extensive access both to the back-end election management systems as well as to the voting equipment that will be sent to each precinct.

Vendor employees: have access to the hardware and source code of the system during development and may also be called upon during the election process to assist poll workers and election officials.

Note that these categories are not intended to be mutually exclusive—a given attacker might have the capabilities of more than one category. For example, it might be possible to combine the limited physical access available to a voter with a network attack such as an outsider would mount. In addition, not all members of a given class have identical capabilities; an on-site vendor employee has a different level of access than an employee who only works with the source code. However, the purpose of this classification is to guide analysis, not to provide a complete taxonomy of attackers.

A particular focus of security analysis is *privilege escalation*. In many cases, one participant in the system is forbidden to perform actions which are normal for another participant in the system. A key feature of a secure design is enforcing such restrictions. For example, a voter should only be allowed to vote once, but poll workers are in charge of allowing people to vote and therefore must be able to authorize new voters to access the system. A voter who was able to use legitimate access to the voting terminal to acquire the ability to authorize new voters would be an example of privilege escalation. Similarly, poll workers are entrusted with maintaining the integrity of their polling place. If a poll worker was able to use authorized access to one polling place to influence or disrupt the voting equipment located in other polling places, that would be another example of privilege escalation.

A.3.1 Outsiders

An outsider to the system has no authorized access to any piece of voting equipment. They may be completely outside the system or may be physically present (perhaps as an observer) but not able to physically touch the equipment. Such an attacker has limited capabilities in the context of this review. They might, for example, enter the polling place with guns and forcefully manipulate the voting systems (at least, until the police arrive). This sort of attack is explicitly out of our scope, although the "booth capture" problem is very much a real concern outside the U.S. [1].

Outsiders may have the power to mount network- or malware-based attacks. Both election management systems and the development systems used by the voting vendors typically run on general purpose operating systems—Windows in the case of all the systems in this review. If those machines are connected to the Internet or connected to machines which are occasionally connected to the Internet or the outside world in any way (laptops are a popular channel), an attacker might manage to infect the systems and thereby alter the software running on the election management systems or even the polling place systems. In that case, individuals anywhere in the world may have the opportunity to attack the voting system.

Outsiders may also have the power to physically tamper with voting equipment. In many counties, voting equipment is stored unattended at the polling place overnight before the election.

[1] Rohde, D. "On the New Voting Machine, the Same Old Fraud". *New York Times.* April 27, 2004.

While polling places may be locked overnight, most polling places are low-security locations; they may be located at a school or church or public building or a citizen's garage. Consequently, an attacker who is willing and able to break into the polling place, either by surreptitiously picking the lock or by forcibly entering, can likely obtain unsupervised physical access to the voter equipment for at least several hours. This kind of attack does require the outsider to be physically present and take on some risk of discovery.

We note that an outsider may be able to impersonate other roles in the system, such as a vendor representative or an election official. As an example, an outsider might mail CDs containing a malicious software upgrade to the election official in packaging that closely resembles the official packaging from the vendor.

A.3.2 Voters

It is very easy to become a voter in California and so it is expected that any attacker who wants to can acquire a voter's capabilities in at least one precinct. Unlike an outsider, a voter has physical access to a voting machine for a short period of time. That access is partially supervised, so that we would not expect a voter to be able to completely disassemble the voting machine. However, in order to preserve the secrecy of the ballot, it is also partially unsupervised. The details of the level of supervision vary to some extent from machine to machine and from county to county. In particular, the difference between optical scan and DRE is relevant here.

Optical scan machines. In optical scan voting, the voter marks the ballot himself but the ballot is simply a special piece of paper. The voter then inserts the ballot into the optical scan equipment, typically under the supervision of the poll worker. Ordinarily, poll workers would be observing voters as they feed their ballots into the scanner. Therefore, the voter probably cannot tamper with the scanner without being detected. This does not entirely preclude voter access to open I/O ports on the scanner but does make it more difficult. The most likely avenue of voter attack is by the ballot itself. For example, a voter might attempt to have multiple ballots recorded or might mark maliciously crafted patterns on the ballot intended to subvert the scanner. Such specific patterns could also be used to trigger a dormant "Trojan horse" (i.e., activate pre-installed malicious software) to induce the machine to begin cheating. Such a compromised machine might otherwise behave correctly.

DRE machines. In DRE voting, by contrast, the voter has mostly unsupervised access to the voting terminal for a significant period of time. The front of the terminal is deliberately hidden by a privacy screen in order to protect voters' secrecy and therefore the voter has the opportunity to mount a variety of attacks. Any section of the machine that is accessible to a voter inside the privacy screen, including buttons, card slots and open I/O ports, must be assumed to be a potential point of attack. The voter also has an opportunity to input substantial amounts of data to the system via the official user interface. It may be possible to use this interface to compromise the machine.

In all the systems studied here, the voter is also provided with some kind of token used to authorize access to the DRE terminal to accept the voter's vote. In the Sequoia and Diebold systems this is a smartcard and in the Hart system it is a four-digit access code. As the voter has access to these tokens, they are also a potential target of attack and the voter might attempt to substitute his or her own token or subvert them (in the case of smartcards).

It's important to note that the privacy afforded to voters by voting in a polling place is intended to be mandatory, but there are many steps a voter may take, while being bribed or coerced, to violate this privacy. This may include the use of cell-phone cameras, the placement of identifying marks on paper ballots, or the placement of unusual voting patterns or specific write-in votes on any voting system. Voters may also be able to take advantage of "curb-side voting," where available, to have private access to a voting machine inside their car (where they may then have tools that would be infeasible to bring into a polling place and the privacy to use them).

Because any United States citizen and California resident is potentially a voter in California, it must be assumed that any attack which requires only voter access is practical.

A.3.3 Poll Workers

Local poll workers have a significant capability that voters do not have: they have legitimate access to the management capabilities of the equipment. For example, the poll worker has the ability to authorize voters to vote. Note that although in principle this may give the poll worker opportunities for malfeasance, this risk may be mitigated by procedural controls. For example, a poll worker who controls the management station can in principle authorize a voter to vote an arbitrary number of times simply by issuing multiple tokens. However, polling places would normally have procedures in place to block or at least detect such attacks: because poll workers must perform their duties in public view, such malfeasance might be noticed by other poll workers or other voters; moreover, if at the end of the day there were more votes cast than registered voters who signed in, that would indicate a problem. Purely technical means may, alone, be insufficient to prevent such attacks, while procedural mechanisms may be sufficient to address such risks.

Depending upon county practices, poll workers may also have long-term unsupervised access to voting equipment. In some counties, voting equipment is stored in the houses or cars of individual poll workers prior to the election. For example, some counties provide the chief poll worker at each polling place with DREs or opscan machines to store and deliver to the polling place. Even counties that deliver DREs and opscan machines by commercial transport may provide the chief poll worker with other equipment (smartcards, smartcard activation devices, management consoles, etc.) before the election. And even if equipment is stored in a secured polling place, dual controls may not be in place to prevent individual poll workers from accessing the area on their own. This provides a number of opportunities for tampering with equipment. Many pieces of equipment include seals to detect such tampering, but each system must be individually analyzed to determine whether these seals are effective and whether they protect all the relevant access points (see Section A.5).

It must be noted that poll workers are primarily volunteers and are subject to extremely minimal security screening, if any is performed at all. In many counties the need for poll workers is so great that any registered voter who calls and offers to serve sufficiently far in advance is almost sure to be hired, and poll workers are often allowed to request to serve at a particular precinct. In practice, we must assume that any attacker who wants to be a poll worker can do so. Therefore an attack which requires poll worker access to a particular precinct is quite practical.

A.3.4 Election Officials

County election officials and county staff have three significant capabilities that poll workers do not have:

- Access to functionality of local voting equipment which may be restricted from poll workers.
- Access to large amounts of local voting equipment between elections.
- Access to the back-end election management system used for equipment management, ballot creation and tabulation.

For reasons of administrative efficiency, this access might be unsupervised or only loosely supervised, depending upon county practices.

The first two capabilities imply greater ability to mount the sort of attacks that poll workers can mount. An election official with access to the warehouse where voting machines are stored might be able to compromise all the equipment in a county rather than merely all the machines in a precinct. In addition, the procedures for some equipment require that they be sealed—for example, the memory cards or results cartridges may be sealed inside the machine—before they are sent to the precincts. Because this sealing happens under the supervision of election officials, those officials might be able to bypass or subvert that process.

The third capability is wholly unavailable to the local poll worker. The back-end election management systems typically run on general purpose computers which are used by the election officials. If those systems are subverted they could be used to mismanage (compromise) polling place voting equipment, create fake or incorrect ballots, and to miscount votes. This subversion could happen in at least three ways. First, many attacks can be mounted using only the attacker's authorized access and within the context of the technical controls which the systems are expected to enforce. For example, the software may offer an opportunity for election official to make "corrections" to vote tallies. Such "corrections" might be incorrect. Second, an election official might find a way to defeat the technical access controls in the election management software and tamper with its vote records.

Third, the official could directly subvert the computers on which the software runs. It is a truism in computer security that if an attacker has physical access to a general purpose computer he can eventually gain control of it. This may be achieved in a number of ways ranging from software attacks to directly compromising the system hardware, but in most cases is quite straightforward. The clear implication of this fact is that if election officials have unsupervised access to the election management systems, the integrity of those systems is provided purely by the integrity and honesty of those officials, not by any technical measures.

A.3.5 Vendor Employees

Finally, we consider attackers in the employ of the vendors. Such attackers fall into two categories: those involved in the production of the hardware and software, prior to the election, and those present at the polling place or Election Central warehouse during an election. An individual attacker might of course fall into both categories.

An attacker involved in the development or production of the software and hardware for the election system has ample opportunity for subverting the system. He or she might, for example, deliberately insert malicious code into the election software, insert exploitable vulnerabilities or back doors into the system, or deliberately design the hardware in such a way that it is easily tampered with. Such attacks are extremely hard to detect, especially when they can be passed off as simple mistakes, since mistakes and bugs are extremely common in large software projects and good-faith mistakes may be very difficult to distinguish from deliberate subversion. Note that for such an attack to succeed it is not necessary for the attacker to arrange for uncertified software or hardware to be accepted by election officials or poll workers. Rather, the vulnerabilities would be in the certified versions. Neither the current certification process nor this review is intended or able to detect all such vulnerabilities.

A vendor employee may also be present in the county to assist election officials or poll workers. For example, the employee might be present at Election Central during or after the election to help election officials, either answering questions or helping to fix or work around malfunctioning equipment. A vendor employee might also be present at Election Central to help install or maintain the voting system or to train county staff or poll workers. A vendor employee might even be present at the polling place or available by phone to assist poll workers or answer questions. Such an attacker would have access to equipment comparable to that of poll workers or election officials, but would also have substantial freedom of movement. Because they are being asked to correct malfunctions and install and configure software, activities which are actually intended to subvert the equipment are much less likely to be noticed. To the extent to which the systems have hidden administrative interfaces they would presumably have access to those as well. Finally, vendor employees pose a heightened risk because they may have access to multiple counties which use the vendor's equipment, and the ability of one individual to able to tamper with voting equipment in multiple counties increases the scope of any potential subversion.

A.4 Types of Attacks

We can categorize attacks along several dimensions:

- *Detectable vs undetectable.* Some attacks are undetectable: they cannot be detected, no matter what practices are followed. Others are detectable in principle, but are unlikely to be detected by the routine practices currently in place; they might be detected by an in-depth forensic audit or a 100% recount, for example, but not by ordinary processes. Still other attacks are both detectable and likely to be detected by the practices and processes that are routinely followed.

  The potential harm caused by the former two classes of attacks surpasses what one might expect by estimating their likelihood of occurrence. The mere existence of vulnerabilities that make likely-to-be-undetected attacks possible poses a threat to election confidence. If an election system is subject to such attacks, then we can never be certain that the election results were not corrupted by undetected tampering. This opens every election up to question and undercuts the finality and perceived fairness of elections. Therefore, we consider undetectable or likely-to-be-undetected attacks to be especially severe and an especially high priority.

- *Recoverable vs unrecoverable.* In some cases, if an attack is detected, there is an easy way to recover. In contrast, other attacks can be detected, but there may be no good recovery strategy short of holding a new election. In intermediate cases, recovery may be possible but expensive (e.g., recovery strategies that involve a 100% manual recount impose a heavy administrative and financial burden).

  Attacks that are detectable but not recoverable are serious. Holding a new election is an extreme remedy, often requiring contentious litigation. There are scenarios where a new election cannot fairly be held: for example, redoing one county's part of a statewide race once the other counties' results become known would be unfair to the other counties.

  In addition, unofficial election results, once announced, tend to take on certain inertia and there may be a presumption against abandoning them. When errors are detected, attempts to overturn election results can potentially lead to heated partisan disputes. Even if errors are detected and corrected, the failure has the potential to diminish public confidence, depending upon the circumstances. At the same time, detectable-but-not-recoverable attacks are arguably not as serious as undetectable attacks: we can presume that most elections will not be subject to attack, and the ability to verify that any particular election was not attacked is valuable.

- *Prevention vs detection.* Often, there is a tradeoff between different strategies for dealing with attacks. One strategy is to design mechanisms to prevent the attack entirely, closing the vulnerability and rendering attack impossible. When prevention is not possible or too costly, an attractive alternative strategy is to design mechanisms to detect attacks and recover from them.

  Most election systems combine both strategies, using prevention as the first line of defense along with detection as a fallback in case the preventive barrier is breached. For example, we attempt to prevent or deter ballot box stuffing by placing the ballot box in the open where it can be observed and charge poll workers with keeping an eye on the ballot box. At the same time, we track the number of signed-in voters, account for all blank ballots, and count the number of ballots in the box at the end of the day to ensure that any ballot box stuffing that somehow escapes notice will still be detected. This combination can provide a robust defense against attack.

- *Wholesale vs retail.* One can distinguish attacks that attempt to tamper with many ballots or affect many voter's votes ("wholesale" attacks) from attacks that attempt to tamper with only a few votes ("retail" attacks). For example, attacks that affect an entire county or a large fraction of the precincts within a county are typically classified as wholesale attacks, whereas attacks that affect only one voter or one precinct are typically classified as retail attacks. This is a useful distinction because, in most contests, retail fraud is not enough to change the outcome of the election. Because wholesale fraud has a more significant impact, we focused especially on analyzing whether the systems are vulnerable to wholesale fraud.

- *Casual vs sophisticated.* Some attacks require little technical knowledge, sophisticated, advance planning, resources, or access. For example, stealing an absentee ballot out of someone's mailbox is a classic low-tech attack: anyone can execute such an attack, and no special qualifications or skills or organization is needed. In contrast, other attacks may require deep technical knowledge, specialized skills or expertise, considerable advance planning, a great deal of time, money, or other resources, and/or insider access. This study examines both sophisticated technical attacks as well as casual low-tech attacks.

When discussing vulnerabilities and potential attacks on the Sequoia voting system, where possible we identify these distinctions to enable readers to form their own judgments about the severity and impact of those attacks.

A.5 Mechanisms for Tamper Sealing

Virtually every election system makes extensive use of tamper seals as a part of its security design. This section presents a brief summary of how these mechanisms work and the level of sophistication an attacker must have to violate them.

Tamper resistance refers to the ability of a system to deter an attacker from gaining access to the system. This could take the form of software controls (e.g., careful limits on the protocols spoken across networks) to procedural controls (e.g., the use of strong passwords) to hardware mechanisms (e.g., strong locks). Tamper resistance generally refers to the amount of time, effort, and/or sophistication required to overcome a security mechanism.

Tamper evidence is the flip-side of the coin to tamper resistance, representing the extent to which an attacker's attempt to overcome a tamper-resistance mechanism leaves behind evidence of that tampering. For example, in a typical home, a burglar could easily break in by putting a brick through a window. A plate-glass window offers little tamper resistance, but strong tamper evidence (i.e., the effort that would be required by a burglar to reinstall a broken window and clean up the broken glass is quite significant). For contrast, a typical door lock is far more tamper resistant than the glass window. However, if a skilled burglar can pick the lock, there will be little or no evidence that it had been picked.

In the context of voting systems, there are a number of tamper sealing mechanisms commonly used:

Key locks: To prevent access to memory cards or sensitive machine ports, many voting machines place a plastic or metal door in front of these ports, using a key lock. Assuming the keys are suitably controlled (and unauthorized duplication is prevented), attackers would be prevented from accessing the protected ports. Of course, if bypassable lock mechanisms are used, or if access to the locked compartment can be gained without opening the lock, then the locks will offer neither tamper resistance nor tamper evidence as has been observed with both Diebold [2] and Nedap/Groenendaal [3] voting systems.

Wire loops: Many voting machines have adopted a mechanism commonly used with traditional ballot boxes—the use of holes through which a metal or plastic wires loops may be fitted. These seals have much in common with standard "tie wraps;" once fitted, the wire loop cannot be loosened; it can only be physically cut off. Like key locks, the loops, when sealed, lock a physical door in place. In common election practice, these seals are stamped or printed with individual serial numbers. Those numbers are then logged when the seals are installed and again when they are cut to detect the substitution of an alternate seal. An attacker with

[2] Feldman, A., Halderman, J.A., and Felton, E. "Security Analysis of the Diebold AccuVote-TS Voting Machine." *2007 Usenix/ACCURATE Electronic Voting Technology Workshop.* Boston, MA. August 2007. (to appear).

[3] Gonggrijp, R. and Hengeveld, W-J. "Studying the Nedap/Groenendaal ES3B Voting Computer: A Computer Security Perspective." *2007 Usenix/ACCURATE Electronic Voting Technology Workshop.* Boston, MA. August 2007. (to appear).

simple tools may be able to clone the serial numbers from old wire loops to new ones without detection [4].

Tamper-evident tape: Adhesive tape can be printed with numbered labels in the same fashion as wire loops. Typically, two different adhesives are used, such that if/when the tape is removed, part of the label will remain stuck to the lower surface while part of the label will be removed with the tape. Technology of this sort is commonly used for automobile registration and inspection stickers. Anecdotal evidence suggests that it may be possible to peel back the tape and replace it without this being easily observable [5].

A recent study of tamper seals considered 244 different seal designs and found that "the majority could be defeated—removed and replaced without evidence—by one person working alone within about two minutes and all of these devices could be thwarted within about 30 minutes" [6]. Needless to say, such seals cannot be counted on, alone, to provide significant security protections for electronic voting systems.

Of course, these mechanisms can be augmented through other procedural means, including requiring multiple people to be present when machines are handled or maintaining video cameras and other locks on the storage areas of the elections warehouse. Johnston also recommends that officials have genuine seals in their hands to compare against the seals being inspected [7].

The use of tamper-evident or tamper-resistant technologies, as such, must be evaluated in the broader context of procedures and policies used to manage an election. Weaknesses in these procedures cannot be overcome by the application of tamper-resistant / tamper-evident seals. Also, the attacker's motivation must also be considered. Perhaps the attacker does not care if an attack is evident, so long as it cannot be recovered from.

[4] Shamos, M. Oral Testimony to the *NIST Technical Guidelines Development Committee (TGDC).* NIST. September 2004.

[5] Rubin, A. *My Day at the Polls – Maryland Primary '06.* Blog entry at `http://avi-rubin.blogspot.com/`

[6] Johnson, R. "Tamper-Indicating Seals." *American Scientist.* Nov-Dec 2006.

[7] Johnson, R. "Tamper-Indicating Seals." *American Scientist.* Nov-Dec 2006.