Source Code Review of the Diebold Voting System

Joseph A. Calandrino
Princeton University[1]

Ariel J. Feldman
Princeton University

J. Alex Halderman
Princeton University

David Wagner[2]
U.C., Berkeley

Harlan Yu
Princeton University

William P. Zeller
Princeton University

July 20, 2007

This report was prepared by 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.

[1]All author affiliations are for identification only.
[2]Team leader.

Executive Summary

This report is a security analysis of the Diebold voting system, which consists primarily of the AccuVote-TSX (AV-TSX) DRE, the AccuVote-OS (AV-OS) optical scanner, and the GEMS election management system. It is based on a study of the system's source code that we conducted at the request of the California Secretary of State as part of a "top-to-bottom" review of California voting systems.

Our analysis shows that the technological controls in the Diebold software do not provide sufficient security to guarantee a trustworthy election. The software contains serious design flaws that have led directly to specific vulnerabilities that attackers could exploit to affect election outcomes. These vulnerabilities include:

- **Vulnerability to malicious software**
  The Diebold software contains vulnerabilities that could allow an attacker to install malicious software on voting machines or on the election management system. Malicious software could cause votes to be recorded incorrectly or to be miscounted, possibly altering election results. It could also prevent voting machines from accepting votes, potentially causing long lines or disenfranchising voters.

- **Susceptibility to viruses**
  The Diebold system is susceptible to computer viruses that propagate from voting machine to voting machine and between voting machines and the election management system. A virus could allow an attacker who only had access to a few machines or memory cards, or possibly to only one, to spread malicious software to most, if not all, of a county's voting machines. Thus, large-scale election fraud in the Diebold system does not necessarily require physical access to a large number of voting machines.

- **Failure to protect ballot secrecy**
  Both the electronic and paper records of the Diebold AV-TSX contain enough information to compromise the secrecy of the ballot. The AV-TSX records votes in the order in which they are cast, and it records the time that each vote is cast. As a result, it is possible for election workers who have access to the electronic or paper records and who have observed the order in which individuals have cast their ballots to discover how those individuals voted. Moreover, even if this vulnerability is never exploited, the fact that the AV-TSX makes it possible for officials to determine how individuals voted may be detrimental to voter confidence and participation.

- **Vulnerability to malicious insiders**
  The Diebold system lacks adequate controls to ensure that county workers with access to the GEMS central election management system do not exceed their authority. Anyone with access to a county's GEMS server could tamper with ballot definitions or election results and could also introduce malicious software into the GEMS server itself or into the county's voting machines.

Although we present several previously unpublished vulnerabilities, many of the weaknesses that we describe were first identified in previous studies of the Diebold system (e. g., [26], [17], [18],

[19], [33], [23], and [14]). Our report confirms that many of the most serious flaws that these studies uncovered have not been fixed in the versions of the software that we studied.

Since many of the vulnerabilities in the Diebold system result from deep architectural flaws, fixing individual defects piecemeal without addressing their underlying causes is unlikely to render the system secure. Systems that are architecturally unsound tend to exhibit "weakness-in-depth" — even as known flaws in them are fixed, new ones tend to be discovered. In this sense, the Diebold software is fragile.

Due to these shortcomings, the security of elections conducted with the Diebold system depends almost entirely on the effectiveness of election procedures. Improvements to existing procedures may mitigate some threats in part, but others would be difficult, if not impossible, to remedy procedurally. Consequently, we conclude that the safest way to repair the Diebold system is to reengineer it so that it is secure by design.

Table of Contents

List of Issues

CHAPTER 1

Introduction

For years, many computer security researchers have been calling for state governments to conduct thorough, independent security studies of their electronic voting equipment. California was among the first states to do so, and we thank the Secretary of State for commissioning this study and for providing top-to-bottom access to the source code under review. We feel honored to have had the opportunity to participate.

Our analysis shows that the Diebold software we studied contains serious design flaws that have led directly to specific vulnerabilities, which attackers could exploit to affect election outcomes. By breaking the seal on just one voting machine, a criminal could launch a vote-stealing virus that could spread to every machine in a county. By manipulating a voting machine for a few minutes after the election, a corrupt volunteer poll worker could determine how each person who used that machine voted. These and many other attacks are feasible.

Furthermore, because the Diebold system suffers from systemic flaws, not just implementation defects, we cannot conclude that the list of vulnerabilities that we present in this report is exhaustive. Indeed, even as we write, we are uncovering new vulnerabilities and learning of additional vulnerabilities being identified by others [24]. Systems with architectural weaknesses tend to be fragile — even as known flaws are fixed, new ones tend to come to light.

Part of the promise of electronic voting is that technological and procedural safeguards can be combined to conduct elections more securely than ever before. The Diebold system does not live up to this promise, however, because its vulnerabilities allow a reasonably sophisticated attacker to surmount almost every technological barrier that is in place. As a result, the security of elections conducted on the Diebold system depend almost entirely on the effectiveness of election procedures.

Leaving the Diebold software largely unchanged and relying on procedural changes to mitigate the threats that we describe may seem like an appealing option, but we consider this to be a risky approach. First, although procedural changes are valuable, we are not confident that they can be completely effective. There are some vulnerabilities that are difficult, if not impossible, to mitigate procedurally. Second, because the Diebold system is vulnerable in so many ways, the procedures required to protect it would likely be extensive, complex, and hard to follow. Hence, we worry that despite the best efforts and intentions of election officials, the procedures would not be followed perfectly every time and the system would sometimes be left open to attack.

The severity of the design flaws in the Diebold system and our lack of confidence in the ability of changes in election procedures to compensate for them leads us to conclude that the surest way to repair the system is to redesign it.

About this Report This report was prepared by 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 Diebold voting system source code.

The Diebold system source code review team was located at Princeton University and consisted of the six authors of this report. We frequently consulted with the Cleveland State University-based

document review team[1] and the UC Davis-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 Diebold system source code on June 8, 2007. Work ended on July 20, 2007 with the delivery of this report, at which time we destroyed all proprietary materials.

Organization This report is organized as follows. The remainder of this chapter introduces the scope of our study, the methods we used, and the limitations of our findings. Chapter 2 describes the overall architecture and individual components of the Diebold election system as it is typically deployed. In Chapter 3, we highlight the most serious attack scenarios we found and discuss their implications. We identify high-level design and architectural problems in Chapter 4, followed by specific vulnerabilities in the individual components in Chapter 5. In Chapter 6, we discuss technical and procedural approaches to improving the security of the Diebold system. We conclude our report in Chapter 7. Appendix A outlines a generic threat model for large electronic voting systems such as the Diebold system. Appendix B contains additional details about the problems we found as well as source code excerpts; since it may contain vendor-proprietary information, this appendix has been designated "private."

1.1 System Overview

The Diebold software we reviewed is part of a system that includes touchscreen direct recording electronic (DRE) voting machines and optical scan voting machines for use at polling places as well as election definition, management, and counting software and hardware for use at a county election headquarters. The specific software components that we reviewed were:

- The GEMS 1.18.24.0 election management system
- The AccuVote-TSX DRE, including:
  - BallotStation version 4.6.4
  - Bootloader version BLR 7-1.2.1 and "Wildcat" platform
- The AccuVote-OS Precinct Count optical scan machine, version 1.96.6
- The AccuVote-OS Central Count optical scan machine, version 2.0.11.4[3]
- Vote Card Encoder, version 1.3.2
- Key Card Tool, version 4.6.1
- VC Programmer, version 4.6.1

In total, the reviewed software comprises about 300,000 source lines of code (SLOC) written in a variety of programming languages, including C, C++, and assembly language. (See Table 1.1.)

[1] Candice Hoke (team leader), Dave Kettyle, and Tom Ryan

[2] Robert P. Abbott (team leader), Mark Davis, Joseph Edmonds, Luke Florer, Elliot Proebstel, Brian Porter, Sujeet Shenoi, and Jacob Stauffer

[3] The state certification of the Diebold voting system identifies version 2.0.12 of the AccuVote-OS Central Count software, and that is the version that the Red Team was given. However, based on directory names and the contents of the source code, we appear to have been given the source code for version 2.0.11.4. We cannot account for the discrepancy between these versions and we have no way of knowing the impact it might have on our findings.

Component	*SLOC*	*Language(s)*
AV-OS Central Count 2.0.11.4	24K	(asm, C, C++)
AV-OS Precinct Count 1.96.6	20K	(asm, C)
AV-TSX Ballot Station 4.6.4	65K	(C++)
AV-TSX bootloader and "Wildcat"	71K	(asm, C, C++)
GEMS 1.18.24.0	116K	(C++)
Key Card Tool 4.6.1	1K	(C++)
Voter Card Encoder 1.3.2	1K	(C)
VCProgrammer 4.6.1	2K	(C++)
(total)	300K	

Table 1.1: The number of non-blank, non-comment source lines of code in each voting system component, as counted by David Wheeler's `sloccount` 2.26. All numbers have been rounded to the nearest thousand lines of code.

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. Line-by-line source code analysis is an extremely time-consuming and laborious endeavor. Consequently, given the size of our team and the time we were allotted, it was unrealistic for us to attempt to examine every line of code or to find every defect in the Diebold system.

Instead, we focused our attention on the portions of the code that were most likely to have an impact on security and reliability. We also used the Fortify static analysis tool[4] to identify potential problem areas that warranted further manual investigation. We made no attempt to catalog all defects that might enable any particular kind of attack. Once we found several related vulnerabilities in the same portion of the code, we stopped looking for other vulnerabilities of the same type. 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 aimed at altering election results, denying service, and compromising ballot secrecy. We also placed a special emphasis on systemic issues that might go beyond individual vulnerabilities. In general, our review attempted to explore questions such as:

- What are the trusted components of the system, when are they trusted and for what purposes? What parties are trusted and for what purposes? What are the implications of compromise of trusted components?
- 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?
- Are security failures likely to be detected? Are audit mechanisms reliable and tamper-resistant? Is data that might be subject to tampering properly validated and authenticated?
- Can an untrusted or minimally trusted user escalate his capabilities beyond those for which he was authorized?
- 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?

[4] We are grateful to Fortify Software for making the tool available to us for this project at no charge.

- 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?

1.3 Limitations of this Report

This report is an analysis of the source code to a particular version of the Diebold voting system and its conclusions do not necessarily apply to other versions of the Diebold system. It is also not intended to be an analysis of the security or reliability of the Diebold hardware.

Our analysis is based on the source code and documentation that we received from the State of California, Diebold, and the Independent Testing Authorities (ITAs). We made no attempt to validate the materials provided to us, nor do we have any way of knowing whether the source code provided to us matches the software that is actually used on election day. Any omissions or inaccuracies in the materials that were provided to us could have led to inaccuracies in this report.

Although we do not have a complete list of the Diebold voting system software, we are aware of several software components that the system uses that we did not receive. They are:

- JResultsClient
- The firmware for the AccuView Printer Module
- The Windows CE operating system used on the AccuVote-TSX
- The Windows operating system and other applications used on GEMS PCs
- Third-party libraries, such as the standard C libraries provided by the compiler vendor

Some of this software, such as the standard C libraries, may be classified as unmodified COTS (commercial off-the-shelf) software under the federal voting standards and thus may be exempt from disclosure to testing labs. Other software, such as JResultsClient, was apparently written by Diebold but was still not made available to us. In the absence of source code to the Windows CE operating system used on the AV-TSX, we were not able to verify whether it qualifies as unmodified COTS under the provisions of the federal voting standards.

This report is not intended to be a comprehensive evaluation of California election procedures. Nevertheless, in the course of our analysis, we discuss the extent to which various election procedures are able to mitigate security vulnerabilities in the Diebold software.

CHAPTER 2

Architecture

In this chapter we provide a high-level overview of the components of the Diebold system and describe how they are used in a typical deployment, as illustrated in Figure 2.1.

2.1 Components at Polling Places

There are several components that might be found at polling places, depending on county election practices:

- The *AccuVote-OS (AV-OS) Precinct Count* is an optical scan voting machine. During an election, voters mark paper ballots and feed them into the AV-OS. The AV-OS scans their ballot, interprets the marked votes, increments running counts of the number of votes for each candidate, and deposits the ballot into a sealed ballot box. If the AV-OS detects overvotes (voting for more candidates than the allowed number of candidates in a contest), it can return the ballot to the voter for correction.

  Officials or poll workers configure the AV-OS for each election by inserting a memory card into a slot on the front of the machine. The memory card stores the names of races and candidates, interpreted code used for printing reports, and the running tallies of votes for each candidate. At the end of the election, poll workers remove the memory card from the AV-OS and officials at election headquarters upload the results to a tabulation system to determine the result.

  The AV-OS memory card uses a non-standard interface format but acts only as a passive storage device. It contains all the election-specific information and can be used in any AV-OS machine.

  The AV-OS runs custom election software written by Diebold. The software is a monolithic application that executes directly in a single-threaded fashion on the microprocessor. There is no operating system and no support for multi-user operation, timesharing, or memory protection.

- The *AccuVote-TSX* (AV-TSX) is a DRE voting machine. It interacts with the voter via a touchscreen LCD display, and it supports audio ballots for increased accessibility.

  The AV-TSX is configured for each election by inserting a memory card into a slot behind a locked door on the side of the machine. The memory card is a standard PCMCIA flash storage card. Before the election, the file system on the memory card stores the election definition, sound files, translations for other languages, interpreted code that is used to print reports, and other configuration information.

  As each ballot is cast, the AV-TSX stores an electronic record of the votes associated with that ballot onto a file on the memory card. At the close of polls, the AV-TSX counts all of the votes

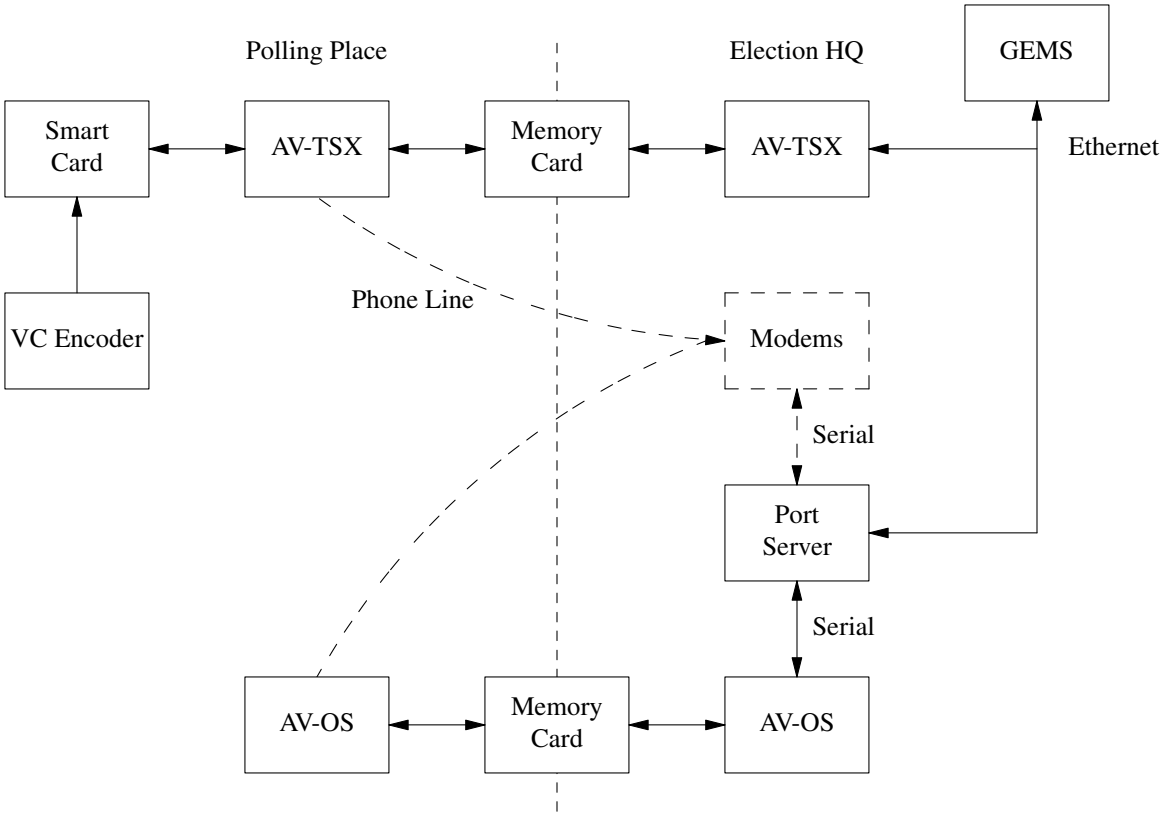

Figure 2.1: **Components of the Diebold system in a typical county.** At election headquarters, there is a GEMS server connected via Ethernet to one or more central-office AV-TSX machines and through a port server to one or more central-office AV-OS machines. These machines read and write memory cards, which are used to transfer ballots to machines at the polling place and to read back election results. Polling places also contain voter card encoders, which program smart cards that allow voters to access the AV-TSX machines. Optionally, modems are used to transfer unofficial ballot results from the polling places.

and prints a summary tape showing the vote tallies. After the election, poll workers remove the memory card from the machine and send it to election headquarters so that the electronic vote records can be uploaded for tabulation.

The AV-TSX also contains a printer attachment that is used for printing a voter-verifiable paper audit trail (VVPAT) corresponding to each ballot cast by the voter. Before casting their ballots, voters have an opportunity to examine printed VVPAT records and confirm that they accurately represent their intent.

Internally, the TSX contains much of the same hardware found in a general-purpose PC. It contains a 32-bit Intel xScale processor, 32 MB of internal flash memory, and 64 MB of RAM. The TSX runs version 4.1 of Microsoft's Windows CE operating system with modifications written by Diebold. An application called *BallotStation* runs on top of the operating system and provides the user interface that voters and poll workers see. BallotStation interacts with the voter, accepts and records votes, counts the votes, and performs all other election-related processing. The TSX also contains a custom bootloader and other low-level support software.

- *Smart cards* are used with the AV-TSX to authenticate voters and poll workers. Each smart card is a piece of plastic in the shape of a credit card with an embedded computer chip that

can communicate with the AV-TSX when inserted into a slot on the side of the machine. Smart cards are used for several purposes:

  - *Voter cards* are used to authenticate voters. When a voter signs in, a poll worker gives them an activated voter card. The voter inserts the card into an AV-TSX, and the machine allows her to cast one ballot. Once the vote has been recorded, the AV-TSX deactivates the voter card so that it cannot be used to vote a second time. The voter returns the card to poll workers, who can reactivate it for subsequent voters.

    Alternatively, in some jurisdictions poll workers activate the voter card and then insert it into the AV-TSX unit for the voter, so that voters do not have to insert it themselves.
  - *Supervisor cards* are used to authenticate poll workers. The chief poll worker would normally be given a supervisor card. When the supervisor card is inserted into an AV-TSX unit, the poll worker is presented with extra functionality not available to voters, such as the ability to close the polls or examine audit logs. Supervisor cards would normally not be provided to voters.

- The *vote card encoder* is a calculator-sized device used by poll workers to generate new voter cards. Poll workers insert voter cards into the device to activate them. Optionally, workers can indicate which ballot style the voter should receive for split precincts or primary elections.

- *Memory cards* are used by the AV-OS and AV-TSX, as described above.

  Practices regarding use of memory cards vary from county to county. Typically, voting equipment might be delivered to the polling place with memory cards already sealed into them. In other words, before the election, county staff program the memory cards with election definition files, insert them into AV-OS or AV-TSX units at the warehouse, and place a tamper-evident seal over the memory card door. Then they ship the AV-OS or AV-TSX units to polling places. Alternatively, memory cards can be provided to poll workers separately and poll workers can insert the memory cards into the AV-OS or AV-TSX units on election morning before the polls are opened.

  After the close of polls, there are several options for how memory cards can be returned to the county. One option is that poll workers can break the seals on the memory card doors, remove the memory cards, and transport them back to county headquarters. Another option is that the equipment can be returned with the memory cards still sealed inside them, and after receiving the equipment at the warehouse, county staff can break the seals and remove the memory cards. Counties normally choose one of these two options.

We note that some counties may not use all of these components. Some counties do not use the AV-OS; in those counties, all (or most) voters vote on AV-TSX machines, and polling places are typically equipped with enough AV-TSX units to handle the expected turnout. Other counties use both the AV-OS and the AV-TSX, and a polling place might contain both an AV-OS for scanning paper ballots and one or more AV-TSX units for accessibility. Counties in the latter category can offer voters the option of voting by paper ballot or by touchscreen, or they can require most voters to use the paper ballot and reserve the AV-TSX unit for voters who need the accessibility or language support it provides.

2.2 Components at Election Headquarters

In addition, there are a number of components present at the county's elections headquarters:

- *GEMS* is an election management software application that runs on an ordinary desktop PC. GEMS is used to control many aspects of the election, including designing ballots, downloading election definition files to voting machines, compiling election results, and reporting the election outcome.

GEMS is a Windows application. Typically, it runs on a PC configured by the vendor running Windows 2000 or Windows XP as well as a number of commodity software applications (e. g., Adobe Acrobat reader). GEMS uses Microsoft's Jet database technology (the database engine used by Microsoft Access).

- The *AccuVote-OS Central Count* is an optical scan machine used for scanning and counting paper ballots at the election headquarters. It is commonly used to scan absentee (vote-by-mail) ballots as well as provisional, damaged, duplicated, or enhanced ballots. The AV-OS Central Count machine connects to GEMS via a serial link, and its operation is controlled by GEMS. It scans ballots and interprets ballot marks, but it then immediately uploads a record of each vote to GEMS and does not attempt to tabulate or keep any record of votes.

  The AV-OS Central Count uses essentially the same hardware as the AV-OS Precinct Count, but the two models run very different software.

  The AV-OS Central Count is normally used in conjunction with the AccuFeed unit, which feeds paper ballots into the scanner at a controlled rate. A small infrared (IR) sensor attached to the AV-OS unit, and then the AccuFeed and AV-OS communicate by IR to control ballot feeding.

- An *Ethernet network* would typically be used to connect many of the devices at the central office. As mentioned below, devices on the Ethernet network might include the GEMS PC, a port server device, AV-TSX units, other PCs (e. g., a PC running JResultsClient, for displaying unofficial election results to observers on election night), and potentially PCs used for unrelated purposes.

  Diebold employees normally set up an Ethernet network and configure the devices on the network for the county. Of course, over the lifetime of the voting system, installation and configuration decisions are at the county's discretion. In some counties this Ethernet network might be strictly isolated. However, we presume it is also possible that this network might be connected to the election department's internal network or the county intranet; we were not provided any detailed information on individual county practices, and we were not able to rule out such a possibility. We did not find any clear prohibition in the system documentation that forbids connecting other devices or networks to this Ethernet network.

- One or more AV-TSX units would normally be connected to the GEMS PC by Ethernet. These AV-TSX units are identical to AV-TSX units used in the polling place, but they serve a different function: they are used to read and write AV-TSX memory cards. We will call any AV-TSX unit that is used in this fashion a "central-office AV-TSX" to distinguish it from an AV-TSX unit that is used by voters. These AV-TSX units in principle can also be connected by serial cable, but Diebold has told us that counties would normally use an Ethernet network instead of a serial cable. County staff insert a PCMCIA Ethernet card into one of the PCMCIA slots on the AV-TSX and then connect the AV-TSX to an Ethernet hub. The BallotStation application provides the result upload capabilities and interfaces with the GEMS server over the network.

  Before the election, once election administrators have laid out the election on the GEMS server, county staff use GEMS and the central-office AV-TSX units to write election definition files onto memory cards. Staff must prepare one memory card per AV-TSX that will be deployed in the field. For instance, a county might have 2000 AV-TSX units that will be deployed in polling sites throughout the county on election day, and might have 5 central-office AV-TSX units used for writing memory cards. County staff would then write those 2000 memory cards by inserting each memory card into a central-office AV-TSX unit, one at a time, and instructing GEMS to make the proper election definition files available for the machine to download. Once all memory cards have been programmed, they can be inserted into AV-TSX units destined for the field.

  After the election, as poll workers return memory cards to county headquarters, county staff use the central-office AV-TSX units to read results files from the memory cards and upload

them to GEMS for tabulation. GEMS inserts the vote data into its database and tabulates the votes.

- One or more AV-OS Precinct Count units would also normally be connected to GEMS by serial cable. These central-office AV-OS units serve a purpose analogous to that of the central-office AV-TSX units. They are controlled by GEMS and used to read and write memory cards intended for use with AV-OS units in polling places.

  Note that AV-OS memory cards are not compatible with AV-TSX memory cards, since they are of a different shape and use a different technology. Therefore, AV-OS memory cards must be read and written by central-office AV-OS units, while AV-TSX memory cards must be read and written by central-office AV-TSX units.

- One might also find a device used to expand the number of serial links that can be connected to the GEMS PC. The sample GEMS setup examined by the Red Team used a *Digi PortServer II*, which is an embedded device that connects to the GEMS PC via an Ethernet network (using TCP/IP) and provides up to 64 serial ports that can be connected to central-office AV-OS units or modems [3]. Alternatively, one might find a Digi device that works similarly except that it is connected to GEMS by a serial link instead of via an Ethernet network. These are commodity devices sold on the open market. They are used to expand the number of AV-OS units that can be connected to GEMS. Because ordinary PCs normally have only one serial port (or at most a few serial ports), this provides a way to connect many AV-OS units to the single GEMS PC.

- Several types of *smart cards* are also used at county headquarters to authenticate county staff to the AV-TSX units. In particular:

  - *Central election administrator cards* are used to authenticate county staff. Insertion of a central administrator card into a AV-TSX unit yields access to additional functionality, such as the ability to configure the AV-TSX unit. These cards would normally not be provided to voters or poll workers and would be closely held by county workers.
  - *Security key cards* are used to update the cryptographic keys on the AV-TSX. These cards are security-sensitive and should only be available to trusted county staff. Security key cards are created using the *Key Card Tool*, a Windows software application installed on a PC in county headquarters.

  Note that, while there are four types of smart cards in total, their internal components are physically identical: the four types of smart cards differ only in the data that has been written to them, and in the label that is printed on their exterior. When a smart card is inserted into the AV-TSX, the AV-TSX uses the data it reads on the card to determine what type of smart card was inserted.

- One might find *modems* that can be used to accept communications over the public telephone network. We discuss modems in detail in Section 4.1.8.

CHAPTER 3

Major Attacks

In the course of our source code review, we identified numerous issues that might allow an attacker to compromise the integrity, reliability, and secrecy of elections run on Diebold systems. These problems are compounded since attackers can exploit multiple vulnerabilities in combination to carry out more powerful attacks. In this chapter, we discuss two kinds of combination attacks that we believe are among the most serious that we have identified. The first attack could allow a single technically sophisticated person with limited access to election equipment to spread a voting machine virus to all machines within a county. A virus could subtly switch votes from one candidate to another, or cause widespread disenfranchisement by overwriting the machines' firmware. The second attack could enable election officials or other insiders to violate ballot secrecy and discover how voters voted.

3.1 Voting Machine Viruses

Like desktop PCs, computer voting machines are vulnerable to viruses. Viruses are malicious software programs that spread automatically from machine to machine. Viruses pose a particularly severe threat to voting security because they can spread invisibly in the background, even when procedural safeguards that limit physical access to the machines are followed. We believe it would be possible for a sophisticated attacker, by exploiting several of the vulnerabilities that we discovered, to launch a powerful virus that would spread from even a single infected voting machine to all the AV-OS and AV-TSX machines within a county.

In prior work, Feldman, et al. [14] demonstrated a working voting machine virus that could spread automatically between AccuVote-TS units if the machines were booted with infected memory cards inserted. From our review of the source code, it appears plausible that the attacks Feldman, et al. identified remain feasible on the AccuVote-TSX. Other vulnerabilities that we identify in this report create further avenues for spreading viruses that are potentially more dangerous than previously known mechanisms.

Creating a voting machine virus like the one we describe below would require moderate to sophisticated programming skills and access to voting equipment, but both are likely available on the black market. A single person with these capabilities could create a virus. An AccuVote-TS was recently listed on eBay, and an attacker unable to purchase one could attempt to steal one instead. Machines purchased or stolen from other states could be just as useful to attackers as ones from California (if the machines had the same software version), so improving physical safeguards within the state will only have limited benefit.

We now describe one scenario where a voting machine virus could spread throughout a county's election system (see Figure 3.1). Many variations on this scenario are possible, so attempts to fix this problem must not focus exclusively on the specifics of this attack.

1. **Initial infection of an AV-TSX**

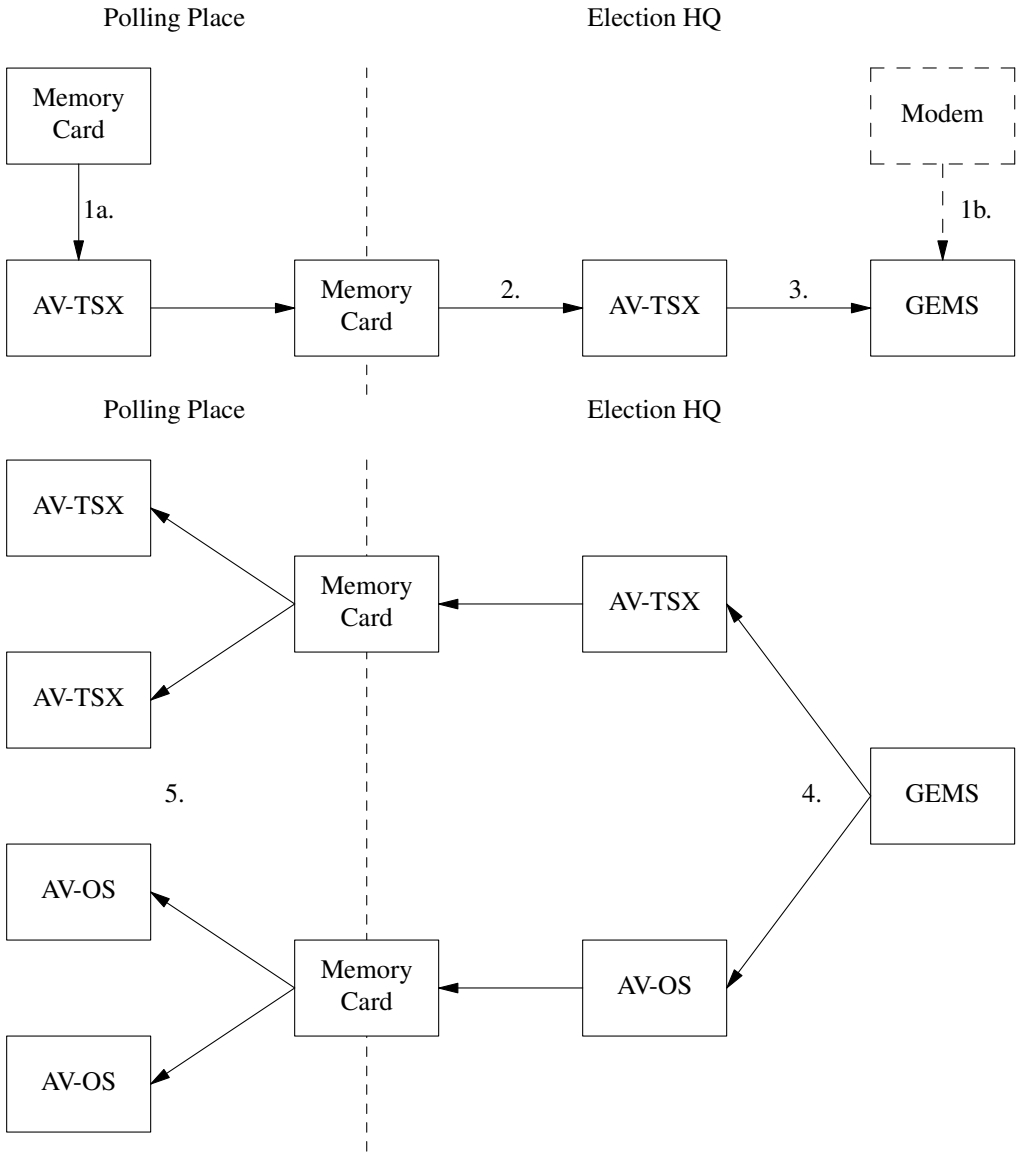

Figure 3.1: **Propagation of a virus over the course of two election cycles.** During the first election: (1a) An attacker temporarily inserts a memory card containing a voting machine virus into an AV-TSX, infecting the machine. (2) After the election, poll workers remove the memory card containing ballot results from the infected machine and send it to election headquarters for tabulation; the virus has corrupted the files on the card, so inserting it into a central-office TSX infects that machine. (3) The infected central-office TSX attacks the GEMS PC over the Ethernet network by using known vulnerabilities in Windows; when the attack succeeds, the virus infects the GEMS server. (1b) Alternatively, if the county uses modems to return unofficial election results, an attacker can target the GEMS server directly over the modem connection, infecting it directly.

During the next election cycle: (4) The virus running on the GEMS server infects memory cards when officials download the new ballots; these cards are placed in voting machines throughout the county. (5) On election day, the virus executes its payload, which may involve altering votes or otherwise disrupting the election.

The attacker, after developing the virus in advance of the election, needs only momentary physical access to an AV-TSX or memory card in order to initiate the infection.

One way to infect the initial machine would be to exploit the insecure software update mechanism described in Issue 5.2.1 or Issue 5.2.2.[1] If memory cards ship separately from machines, the attacker could intercept a memory card en route and copy the virus onto the card. The virus would be installed on the AV-TSX when the machine boots with the card in place. AV-TSX memory cards are commodity PCMCIA cards that can be bought on the open market and read and written using any laptop, so no special equipment is required to mount this attack. If, instead, machines ship with memory cards sealed in place, the attacker would need to gain physical access to a machine, break the seal and unlock the lock, replace the memory card with one containing the virus, reboot the machine to install the virus, reinsert the original memory card, and relock the enclosure. Though this may sound complicated, the Red Team has told us that software updates can be installed with less than one minute of physical access in a manner that would likely raise minimal suspicion from poll workers. The only physical evidence would be the single broken seal.

Some counties use modems to transmit unofficial election results back to election headquarters after polls close. In this case, an attacker could attempt to infect the GEMS server directly by connecting to it over the modem. As discussed in Section 4.1.8, this would allow the attacker to skip directly to step 4 below.

2. **Viral spread to the central-office AV-TSX**

   After the election, officials remove the memory cards from each machine and take them to the election headquarters, where a small number of AV-TSX machines are networked to a GEMS server. The next step in the virus's lifecycle is to infect these central-office AV-TSX units.

   On the initially infected AV-TSX, the virus can manipulate the election database file stored on the removable memory card by exploiting Issue 5.2.6. The attacker could design the virus to corrupt the file to exploit problems, such as Issue 5.2.16, that allow the execution of arbitrary code during the result upload stage. Later, officials take the memory card with the manipulated election description and place it into a central-office AV-TSX. When officials initiate the upload function, the attacker's code executes and infects the central AV-TSX machine with the virus. A well-crafted virus might be able to do this without causing any visible signs of foul play.

3. **Attacking the GEMS machine**

   As soon as the virus infects the central-office AV-TSX, it can begin attacking the GEMS machine. In a typical deployment, as described by Diebold, the GEMS machine and the central-office AV-TSX machines attach to a single Ethernet switch and communicate using TCP/IP. This means that the GEMS PC exposes a large attack surface to the AV-TSX. Vulnerabilities in the PC's operating system (Windows), network drivers, and network services could all be attacked. The hacker community is already aware of exploitable flaws in some of these components. Even if automatic patches exist for these commodity components, the PC's software may not be up-to-date.

   The Red Team's report describes how they were able to use widely available exploit tools to exploit holes in Windows and take control of the GEMS PC from another PC on the same subnet [3]. A virus running on the central AV-TSX could be programmed to perform a similar attack. After gaining control of the GEMS PC, the virus would install itself and proceed to the next phase of its lifecycle. It could hide itself from system administrators and from common security tools[2] using rootkit techniques [16].

[1] For example, the initial infection might replace the machine's bootloader software. This bootloader could then install high-level infection software [14].

[2] Standard anti-virus software would be unlikely to detect a special-purpose voting machine virus that had infected the GEMS server. Such software does not exist for the AV-TSX and AV-OS.

4. **Spreading back to the field**

   At the beginning of the next election cycle, the infected GEMS system can spread the virus to the voting machines used in the field. It might spread to AV-TSX systems by tampering with the election data files as they are downloaded to memory cards that will be distributed to polling places. By introducing deliberate errors into these files, the virus could exploit vulnerabilities (e. g., Issue 5.2.13, Issue 5.2.15) that will allow virus code to execute on the systems during voting. The virus could also spread to AV-OS Precinct Count machines in a similar manner by exploiting Issue 5.1.2. Since typical procedures call for every memory card used in the county to be created using the GEMS server, this step would allow the virus to infect every AV-OS and AV-TSX machine used by voters.

3.2 Virus Payloads

What harm can a voting machine virus or other wide-scale compromises do? Among the most dangerous payloads would be an attempt to shift a close race by subtly stealing votes and an attempt to disrupt an election by launching a large scale denial-of-service attack. Procedural countermeasures might not be sufficient to defend against these attacks, as discussed in Chapter 6.

1. **Subtle vote stealing**

   An attacker could use a voting machine virus to reprogram a large number of AV-TSX or AV-OS machines to steal votes. When programming the attack, the attacker could decide which votes to steal (e. g., from particular candidates, races, or parties), how to steal them (e. g., by adding, deleting, or switching votes from one candidate to another), and when to execute the attack (e. g., only in closely contested races, or only in precincts with certain voting patterns). California's mandatory voter-verifiable paper audit trail (VVPAT) provides a valuable defense against electronic vote stealing, but it will not necessarily be able to detect and correct every kind of attack — particularly in races with a narrow margin of victory.

   In a close election, one particularly dangerous scenario would be a widely-spread virus that subtly shifts votes between candidates on both the paper and electronic records. Suppose the candidates are named Alice and Bob. A Bob supporter could reprogram the machines to look for voters who select Alice. One percent of the time, after the voter has selected Alice, they machines could behave as if the voter had picked Bob, displaying a vote for Bob on the confirmation screen and on the printed paper record. A cleverly designed virus wouldn't interfere with an attempt to correct the problem, so voters who notice the error could cancel the printed VVPAT record and change the selection back to Alice.

   An attack like this might shift enough votes to cause the wrong result in a close election, but could it really be done without being detected? Several factors favor the attacker. First, assuming that only a small fraction of voters would carefully review the paper VVPAT record, many voters would overlook the problem and allow incorrect votes to be recorded in both the electronic and paper records. Second, while a few voters might report the problem to poll workers, election officials would have difficulty determining whether the cause was voter error or a problem with the machines. This is similar to problems that Sarasota County, Florida experienced with its DRE voting machines in the November 2006 election [11]. In one race during that election, hundreds of voters reported that the machines displayed the wrong selection on the summary screen, or that they failed to show the race on the summary screen at all. Some observers eventually concluded that the cause was voter error due to a poorly designed ballot layout.

   Even if officials suspect an electronic attack, the virus author could take countermeasures to thwart later investigation. The attacker could tamper with the system logs to remove traces of the virus's activity, and remove the virus after the election when the machine powers on again. By the time an investigation is commenced, most of the evidence of the problem could be destroyed.

Finally, even in the best case when officials do detect the virus, they might have difficulty undoing its effects without holding a new election — thus, the vote stealing attack becomes, at best, a massive denial of service attack. It would probably be impossible to tell how many votes had been shifted as a result of the attack, since the electronic and paper records would both reflect the fraudulent result.

A virus could also shift vote totals in the reports produced by AccuVote-OS Precinct Count machines. Conceivably, this would need to be paired with a corresponding attack that alters the paper ballots. This attack is considerably more difficult to accomplish on a large scale since the AV-OS scanner is unable to alter ballots.

Another notable but less-damaging attack by a virus author would be to re-program the AV-OS to selectively allow overvotes for a disfavored candidate. A voter who accidentally overvotes would not be notified by the AV-OS of her mistake and would not be given a chance to fix her ballot. After the election, if an overvoted ballot is rescanned, the overvote would likely be deemed invalid by officials. This attack might raise suspicion since AV-OS-scanned ballots should have been previously checked for overvotes, but the damage would be done by the time the attack was detected.

Other means of attacking the VVPAT are discussed in Section 3.3.

2. **Massive denial of service**

   Rather than stealing votes directly, attackers might choose a more passive strategy and attempt to disrupt the election process itself by disabling machines, destroying vote records, or slowing down voting. These attacks could be targeted at precincts that are likely to support an opposing candidate (or even triggered only after the virus detects that the opposing candidate has won a certain portion of the votes on a machine). Alternatively, the attack could be carried out indiscriminately in hopes of causing such widespread disruption that the election would be postponed.

   What kinds of disruption could a virus cause? The AccuVote-TSX memory architecture exposes important system code, including the bootloader, operating system kernel, and voting software, to tampering by other software running on the machine. Malicious software on the machine could overwrite the bootloader or other parts of the software, rendering the machine inoperable. Repairing this damage would require a visit from a service technician and possibly even a return to the factory. Feldman, et al. demonstrated how such a denial of service attack was possible with the AccuVote-TS [14].

   An attacker would have many choices about when and where to trigger an attack and what kind of damage to do. Some attacks might be very difficult to distinguish from non-malicious hardware and software malfunctions.

   Denial-of-service attacks on the AccuVote-OS machines would be slightly less damaging. If an attack causes the AV-OS machines to break down, only voters who invalidly fill out their paper ballot (e. g., overvote) will be affected, since they will not be warned or given the chance to correct their mistakes.

3.3 Attacking the VVPAT

The design of the VVPAT mechanism used by the AV-TSX poses a threat to the secrecy of voter ballots and places a limit on its ability to detect malicious software. Two aspects of the design are especially problematic:

- The AV-TSX contains a movable flap over the VVPAT window and can be open or closed. When it is closed, the voter cannot see the VVPAT record and is unlikely to know in advance that the flap needs to be opened. If the flap is closed when the voter walks up to the machine, they may cast their ballot without checking the VVPAT record and without even being aware

that they could have checked the VVPAT record. The flap closes easily, and once it closes, it might stay closed for many voters.

Consequently, the presence of this flap may reduce the number of voters who check the VVPAT record. This makes audits less effective because it undermines the presumption that the VVPAT record accurately represents the voter's intent. This flap may also heighten the damage done by paper jams and printer failures: if the flap is closed when the printer jams, then several voters may continue voting before anyone notices that multiple VVPAT records have been destroyed and rendered unreadable by the unnoticed printer jam.

- The AV-TSX VVPAT uses a reel-to-reel printer mechanism. The system contains a spool of blank thermal paper which feeds through the printer mechanism, then past the window where it is visible to the voter, and then winds onto a second take-up spool. The machine records votes continuously on the spool of paper without cutting it in between voters. Consequently, election workers have an opportunity to associate voters with the paper records by matching the order in which voters used the machine to the order of the records on the paper.

Most voters only vote at a polling place once or twice every two years. Consequently, each time they use the equipment, they will effectively be learning for the first time how to use it. Voters cannot be expected to know in advance the intricacies of how the machines work: instead, they will likely be learning this as they go along. This creates the possibility that malicious software on an AV-TSX could trick voters by subtly deviating from the normal protocol for printing VVPAT records.

To illustrate the risks, we list four hypothetical attacks that an attacker who has subverted the software on the AV-TSX could mount. This list of example attacks is not intended to be exhaustive or comprehensive.

1. In our first attack, the malicious AV-TSX behaves honestly for 90% of the voters, and randomly selects 10% of the voters to cheat. For those unlucky voters, it behaves legitimately except that it always prints candidate X on the VVPAT record, whether the voter selected candidate X or his rival. If the voter notices the error and spoils her ballot, the machine allows her to go back and change her selection, and it behaves honestly for the remainder of that voter's session. If the voter does not notice, the machine casts the ballot. The machine records an electronic vote record matching whatever is printed on the VVPAT record.

   If we assume that perhaps 50% of voters will not bother to check what is on the VVPAT record, the machine will succeed in stealing up to 5% of the votes, which may be enough to overturn the outcome of a close race if every machine in the county contains this malicious software. Moreover, this attack leaves no permanent evidence that would be detected during the 1% manual tally or the official canvass. In principle, an election official who counted the number of spoiled VVPAT records might notice the increase in spoiled VVPAT records [4], but we are not aware of any county that currently performs this check as part of the official canvass.

   We are not convinced that increasing the number of spoiled VVPAT records by an amount equal to 5% of the total number of ballots would be enough to provide a clear indication of fraud. Even if suspicions were raised, the evidence might still admit multiple possible interpretations and thus, there might be no indisputable evidence of fraud that a court could use to throw out the election results. Furthermore, even if the attack is detected and a court is persuaded to order a new election, the controversy could undermine voter confidence. This is an example of an attack that cannot be mounted against manually-marked paper ballots.

2. In our second attack, a malicious AV-TSX unit cheats 5% of the voters by deviating from the normal desired operation in only one small way. When the unlucky voter casts her ballot, the machine does not scroll the VVPAT record up into the security canister. It simply leaves the VVPAT record showing in the glass window and prints a message on the screen saying "Your vote was recorded. Thank you for voting." Shortly after the voter walks away, the

machine prints "CANCELLED" under the VVPAT record to spoil it, scrolls it up into the security canister, prints a new VVPAT record containing votes for the attacker's preferred candidates, and scrolls that into the security canister. This might be difficult to detect.

3. Alternatively, one can imagine that when the unlucky voter casts her ballot, the AV-TSX machine immediately prints "CANCELLED," then prints a new VVPAT record, and scrolls that up into the security canister, all in one action. If the printer scrolls the paper fast enough, it might be difficult for a first-time voter to notice what went wrong. If an occasional unlucky voter does happen to notice the misbehavior, she cannot demonstrate to anyone else that the machine cheated her: by the time she can call over a poll worker or another witness, the evidence is gone and it is too late. Even if a poll worker peers over the shoulder of the next few voters (violating their ballot secrecy) to see if the problem recurs, it is unlikely that the poll worker will notice any further problem due to the small number of voters targeted. We fear that even if a few voters do notice the issue and complain, their complaints might be discounted or there might be little that poll workers can do about it.

4. Yet another possible attack applies to counties that use DREs for provisional voting. Suppose that the attacker wants to favor candidate X over candidate Y and is able to introduce malicious software onto the county's AV-TSX units. When a provisional voter steps up to use the machine, the malicious software observes which candidate the voter selects. If the voter selects candidate Y, the machine behaves honestly and prints a VVPAT record that is correctly marked as provisional, prints the challenge code associated with this provisional voter, and correctly records the vote electronically as a provisional vote. However, if the provisional voter selects candidate X, then the machine prints a VVPAT record and makes an electronic record as though this were a non-provisional voter. Later, when a subsequent non-provisional voter tries to vote for candidate Y, the machine prints a VVPAT record and makes an electronic record as though that subsequent voter were a provisional voter (using the challenge code associated with the earlier provisional voter). In effect, each provisional voter who votes for candidate X is matched up with a non-provisional voter who votes for candidate Y, and the provisional status is swapped. Later, during the resolution of provisional voters, if some provisional votes are discarded because the voter was not eligible to vote, then officials will be discarding votes for candidate Y that should have counted and retaining votes for candidate X that should have been discarded. The number of votes that can be stolen in this way is governed by the number of provisional voters whose votes are ultimately rejected. If this number exceeds the margin of victory, this attack strategy can change the outcome of the election. This kind of attack would not be detected by the 1% manual tally or during the official canvass because the electronic records do match the VVPAT records exactly and because the number of voters and provisional voters are not affected by the attack. The only chance to detect this attack is for voters to notice the provisional status of their votes, but it is not clear whether voters would notice this or would understand the consequences if they did.

Malicious software might also try to increase the odds of avoiding detection by targeting its attacks towards voters who are less likely to notice the fraud. For instance, instead of picking voters to cheat at random, the malicious software might watch for voters who appear to be having trouble (as evidenced, for instance, by their use of the "Help" button, by slow progress through the ballot, or by multiple attempts to change their selections) and selectively defraud these voters.

Would these attacks succeed in avoiding detection? We do not know. To the best of our knowledge, it is an open question whether voters would notice. We are not aware of any studies that have tested these attacks, but there are reasons to be concerned [13]. It is possible that none of these attack strategies would succeed. We would like to believe that these kinds of attacks would be detected. However, it is also possible that an attacker who studied human behavior could come up with methods to steal elections without detection. The point is that we do not know whether malicious software could fool voters into accepting fraudulent VVPAT records. Given that the security of the AV-TSX relies upon the assumption that the VVPAT record will accurately represent

voter intent, this uncertainty surrounding VVPATs and voter behavior may be of concern. It seems to cast some doubt on how much we can rely upon the VVPATs.

Of course, these kinds of attacks are only possible if the attacker can find a way to replace the software on the AV-TSX. However, the VVPAT was intended as an independent check upon the operation of the machines, and these risks undermine the independence of the VVPAT records. These attacks are possible because the VVPAT printer and spool are entirely under software control, so if the software is subverted, it can control how the VVPAT record is printed. This architectural feature of the AV-TSX is unfortunate. It might be better to have an architecture that left no plausible avenue for an attacker to subvert the paper trail.

Manually-marked paper ballots scanned with the AV-OS are not subject to these kinds of human factors risks. Because the process of marking the ballot does not involve any interaction with complex technology, there is no opportunity for corrupted devices to try to influence the voter ballot marks. The marks on the paper ballot record the voter's intent unmediated by technology. This may make manual recounts and the 1% manual tally of the AV-OS more effective and less susceptible to subversion by malicious software. Consequently, it seems plausible that voting systems based upon the AV-OS may prove to be more resilient to technical attack than voting systems based upon the AV-TSX.

3.4 Attacking Ballot Secrecy

In addition to threats to the accuracy of election results, we are charged with identifying problems that could threaten the secrecy of voter selections. Secrecy makes it difficult for voters to sell their votes, since they can't prove to anybody else how they voted. Ballot secrecy also helps voters stand up to intimidation by those who threaten to harm them if they do not vote a certain way. We found a variety of issues with the AV-TSX that pose significant threats in this area.

As we describe in detail in later sections, the machine stores votes in the order in which they were cast (Issue 5.2.19); it stores them together with a record of the time they were cast and, if a specific configuration option is enabled, prints this time in a barcode on the paper VVPAT record (Issue 5.2.20); and it assigns them each an encrypted serial number that can be decrypted to discover the order of voting (Issue 5.2.21). Any one of these problems could leak enough information about the votes to reveal how individuals voted.

Exploiting these problems would require three resources. First, an attacker would need access to the voting data — either the barcoded VVPAT records or the election results file from the memory card or voting machine. Second, if attacking the election results file, the attacker would need to know the data key used to encrypt the results file and generate ballot serial numbers; Issue 5.2.5 explains how an attacker with access to a single voting machine can determine this county-wide key. Third, an attacker would need to know on which machines target individuals cast their votes, as well as the time of their votes or their positions in the sequence of votes cast. For example, in a targeted attack, a human observer or hidden camera could observe how many people voted on a machine before the targeted individual. In a broader breach of privacy, the attacker could learn the order of voters from the polling place sign-in list, if that list records the order in which voters sign in. There are several ways that an attacker could obtain this information, but we are particularly concerned that all of the necessary items could be obtained with relative ease by corrupt poll workers or election officials.

Of course, most poll workers and election officials are honest. Poll workers volunteer their time for what is a critical but largely thankless job. What concerns us is that a malicious person who wants to attack the election can purposely volunteer as a poll worker in order to obtain access to sensitive data. Regardless of whether or not any poll workers actually are malicious, the fact that the AV-TSX makes it possible for malicious officials to determine how individuals voted may be detrimental to voter confidence and participation.

CHAPTER 4

Systemic and Architectural Issues

Given the consequences of election fraud and the importance of public confidence in elections, voting systems and software must be designed from the ground up to be secure. Building a secure system involves identifying the threats that it could face and producing a design that not only counters those threats but employs defense-in-depth to limit the damage that any undiscovered vulnerabilities could cause. It also requires the use of defensive programming techniques to minimize software defects and the use of sound software engineering practices to ensure that software developers are properly trained and that source code is properly reviewed before release.

In our analysis of the Diebold system, we found significant systemic weaknesses in its design and implementation as well as in the engineering practices used to develop it. Our analysis is based both on our direct examination of the system's source code and on an interview that we conducted with Talbot Iredale, Software Development Manager, Diebold Election Systems [1].[1]

4.1 Design

4.1.1 Large Attack Surface

Experienced security practitioners often also recommend analysis of the "attack surface" of a software system. The attack surface is the interface that is exposed to the attacker. This includes all operations that the attacker can invoke, any data that the attacker can control, protocols that the attacker can participate in, and so forth. The larger the attack surface, the more degrees of freedom the attacker has in crafting attack strategies. A bug in any code that is exposed to an attacker may lead to an exploitable vulnerability, so a large attack surface also means that a large volume of code is security-critical. Consequently, systems with a large attack surface tend to be more prone to security vulnerabilities.

The Diebold voting system has a large attack surface. Exposed interfaces include:

1. The user interface on the AV-TSX.
2. The protocol spoken between the AV-TSX and the smart card.
3. The content of election database and other files on the AV-TSX memory card, as read by AV-TSX units in the field.
4. The content of the ballot results files on the AV-TSX memory card, as read by other AV-TSX units.
5. The data transmitted between GEMS and a central-office AV-TSX, when the two are connected by Ethernet or a serial link or modem.
6. The protocol spoken between the smart card and the Voter Card Encoder.

[1] We wish to thank Mr. Iredale for his time and his useful insights.

7. The marks on the paper ballot, as scanned by the AV-OS.[2]
8. The election configuration and other data on the AV-OS memory card, as read by AV-OS units in the field.
9. The election results data on the AV-OS memory card, as read by central-office AV-OS units.
10. The data transmitted between GEMS and a central-office AV-OS, when the two are connected by a serial link or by modem.
11. The interface between multiple GEMS installations during regional processing.

Some of these interfaces are complex and present many opportunities for attack. All of them could potentially be manipulated by an attacker. Given this, one would expect that the risk of exploitable vulnerabilities is high. That expectation was borne out during our examination of the source code.

Our analysis of the risks associated with each of these exposed interfaces is as follows:

1. The user interface on the AV-TSX is complex. It seems to have been implemented carefully for the most part, although we did find one buffer overflow that appeared to be possible to exploit (see Issue 5.2.17).
2. The protocol spoken between the AV-TSX and the smart card is fairly simple. The code was not written defensively (see Section 4.2.2 for a definition of defensive programming) but appears to be free of noticeable security vulnerabilities. However, at the protocol level, the design does not appear to have been as successful (see, e. g., Issue 5.2.7, Issue 5.2.8, and Issue 5.2.9).
3. The content of election database and other files on the AV-TSX memory card, as read by AV-TSX units in the field: The format of these files is complex and rich in features, so this is an especially dangerous area for vulnerabilities. The election database contains a marshaled version of a complex data structure, leaving many opportunities for vulnerabilities in the demarshaling code and in malicious election database files that violate expected invariants. The code that reads these files was not written defensively and we suspect that developers may have failed to consider the possibility that the memory card could contain malicious data. We found many design- and implementation-level defects in the code that reads these files. See, e. g., Issue 5.2.1, Issue 5.2.2, Issue 5.2.3, Issue 5.2.13, Issue 5.2.15, and Issue 5.2.14.
4. The content of the election results files on the AV-TSX memory card, as read by other AV-TSX units: The format of these files is of medium complexity. The code involved in reading these files was not consistently written using defensive programming and we found implementation-level defects with serious consequences in this portion of the file. See, e. g., Issue 5.2.16.
5. The data transmitted between GEMS and a central-office AV-TSX, when the two are connected by Ethernet or a serial link or modem, involves a complex protocol. The code involved in interpreting it appears to validate most inputs, but there were some exceptions: we did find several implementation-level defects that would allow an attacker to mount attacks across this interface. See, e. g., Issue 5.2.18.
6. The protocol spoken between the smart card and the Voter Card Encoder appears to be fairly simple.
7. The marks on the paper ballot, as scanned by the AV-OS, present a fairly simple interface and we did not see much opportunity for malicious manipulation of this data to subvert the security of the AV-OS.

[2] In this section, AV-OS refers to the AV-OS Precinct Count machine, not the AV-OS Central Count machine.

8. The election configuration and other data on the AV-OS memory card, as read by AV-OS units in the field: The format of these files is of medium complexity. The code was not written defensively and we found several design- and implementation-level defects in the code that reads these files. See, e. g., Issue 5.1.3, Issue 5.1.5, Issue 5.1.9, Issue 5.1.10, and Issue 5.1.11.

9. The election results data on the AV-OS memory card, as read by central-office AV-OS units: The format of these files is fairly simple, but the code was not entirely free of vulnerabilities. See Issue 5.1.6.

10. The data transmitted between GEMS and a central-office AV-OS, when the two are connected by a serial link or by modem, involves a communications protocol that is of medium complexity. We did not study this code in any depth.

11. The interface between multiple GEMS installations during regional processing: We did not study this code in any depth.

It is interesting to note that the attack surface of the AV-TSX appears to be larger than that of the AV-OS. This is partially a consequence of the fact that the AV-TSX provides more functionality, but it is also a consequence of the way that users interact with these devices. One might predict, based on this analysis, that the AV-TSX would be at greater risk of attack than the AV-OS.

4.1.2 Complexity

The Diebold system is a complex computing system. Complexity is the enemy of security. All code has bugs; the only way to be sure that software will be secure is to arrange for its design and implementation to be so simple and so small that one can inspect all of it and be confident that all of the bugs and defects in the code are found. By that criterion, the Diebold software is too complex to secure. Put another way: If the Diebold system were secure, it would be the first computing system of this complexity that is fully secure.

One crude measure of software complexity involves counting lines of source code. As may be seen from Table 1.1, the AV-TSX (with 136K SLOC, not counting the COTS OS) is a more complex codebase than the AV-OS (with 20K SLOC, and no OS). This provides a second reason why one might expect the AV-TSX to be at a greater risk of security vulnerabilities than the AV-OS. This was at least partially borne out by our analysis of the source code, as mentioned above.

One principle of secure design is to architect the software so that it has a small Trusted Computing Base (TCB). The TCB is that portion of the software whose correctness suffices to ensure that the system security requirements will be met. The system must be designed to ensure that the TCB cannot be bypassed or subverted. That is, the TCB must be protected from attack and must be written to ensure that the rest of the system cannot violate the security policy even if the rest of the system is compromised or malicious.

The Diebold software not appear to have any clearly defined TCB. It is a monolithic system, with no clear trust boundaries. It does not use compartmentalization. Similarly, we found little evidence of any attempt to apply *defense in depth* or to follow the *principle of least privilege*, two standard principles of secure system engineering.

Due to this architecture, a breach of any part of the software may lead to security violations and breaches of the rest of the software. In this sense, the system is fragile. It is like an oceanliner built without watertight doors: a hole anywhere below the waterline is liable to sink the entire ship. Because code of any significant complexity or scale inevitably has bugs, defects, and flaws, this architecture makes it all but inevitable that the Diebold voting software will have exploitable security vulnerabilities.

4.1.3 Misplaced Trust

In our judgment, the Diebold software places too much trust in people and other components of the system. For instance, the software trusts — relies upon — the memory card to contain files from a legitimate, authorized source. In other words, the software is written with the expectation that

the contents of the memory card come from a benign source, and the software does not effectively defend itself against malicious files on the memory card. That trust seems misplaced: it is too easy for an attacker to tamper with the contents of a memory card. When that expectation is violated, the integrity of the software can be breached.

This theme appears throughout the voting system. In many places where two components communicate, both components rely on each other to be benign, which renders them vulnerable to attack if the security of the component happens to be breached. For instance, the GEMS server trusts the central-office AV-OS and AV-TSX units and everything else that is connected to its own Ethernet network. This trust is dangerous. While those devices might be protected against physical tampering, they must handle data that comes from the field and thus might be malicious. Those devices are at heightened risk of subversion, and it would be safer if GEMS and other system components were written to defend against subversion by malicious devices on the same network.

This risk is especially pronounced if county practices involve taking AV-OS or AV-TSX units that were used in the field in a prior election and connecting them to GEMS in a future election. We cannot realistically protect units in the field from physical tampering. Therefore, we must assume that any unit that has spent time overnight at a polling location could have been tampered with by an attacker and its firmware corrupted. If a compromised AV-TSX unit were later plugged into the GEMS Ethernet network and repurposed as a central-office AV-TSX, at that point a malicious device would be plugged into the GEMS Ethernet network. While GEMS could have been written to protect itself from attack by other devices on the same network, it was not. Once any malicious device is allowed to connect to the central-office Ethernet network, there are no effective technical barriers preventing all other devices on the network from being penetrated.

As another example, all AV-OS and AV-TSX units rely upon — are vulnerable to attack by — the central-office GEMS election management system. GEMS can silently program memory cards so that they contain firmware update files or malicious election database files that will replace the running software on every unit in the field with malicious, corrupted software.[3] This means that the consequences of a breach of GEMS security are more severe than they need to be. For instance, while we might be prepared to entrust the GEMS operator not to manually adjust vote tallies (provided that such actions would be logged and could be detected), there is no reason why a rogue GEMS operator should be granted the power to undetectably replace the software on every AV-OS and AV-TSX unit in the county. Yet this is effectively what the Diebold voting system allows.

This type of pervasive trust makes the Diebold system brittle: a small security breach can have large consequences out of proportion to the initial breach. That, in turn, places an unnecessary burden on procedural protections, because even a brief violation of procedure or a small, seemingly negligible breach of the chain of custody can have disproportionately harmful effects.

4.1.4 Bidirectional Information Flow

The Diebold voting system includes a bidirectional flow of data. Information flows from GEMS to every unit in the field (via memory cards), and from all units in the field back to GEMS (again, via memory cards). This poses some risk of viral spread of infection. In particular, if (1) a memory card or unit in the field can be corrupted and (2) there are any exploitable flaws in the handling of data on the memory card, then a virus may be able to spread from one unit in the field to GEMS and then back to every unit in the field. In practice, we found that both prerequisites are met. Due to the complexity of the data on the memory card, any system of this architecture seems to be at high risk of viral spread. The bidirectional flow of data heightens the impact of these vulnerabilities by allowing viruses to spread throughout the system.

This is not a necessary property of a voting system. For instance, it would be possible to have one central-office application for programming memory cards for distribution to the field and a second application for reading memory cards from the field and tabulating results, with firewalls to ensure that any penetration of the second application cannot affect the first application. See

[3] GEMS can permanently replace the firmware stored on every AV-TSX in the field. In the case of the AV-OS, GEMS cannot replace the stored AV-OS firmware, but GEMS can cause the running software to be altered. That alteration will persist until the AV-OS unit is powered down and the memory card removed, but it is not permanent.

Section 6.10. However, the Diebold voting system was not designed with those kinds of firewalls in place, and it was not constructed in a way that would provide inherent resistance against the spread of virally propagating malicious code.

4.1.5 Insufficient Controls on Code Integrity

The Diebold devices do not contain strong controls to protect the integrity of their software. For instance, the AV-TSX can be upgraded in any of several ways simply by placing unauthenticated files on a memory card. The GEMS application can be upgraded simply by installing new software on the GEMS PC. All of the software on the AV-TSX and GEMS PC is installed on writable storage (either non-volatile flash memory or a magnetic hard disk). For instance, the bootloader, WinCE operating system, and BallotStation are all stored on writable storage on the AV-TSX and can all be upgraded using authorized channels. The consequence is that if an attacker can ever run malicious code on any of these machines, even once, the attacker can permanently replace all of the software stored on the machines. Moreover, because the upgrade process is under software control, there is no easy way to reset a machine and restore it to a safe state once its software has been corrupted. An AV-TSX or GEMS server, once infected, is very difficult to disinfect with confidence.

This is an architecture that is also shared by commodity PCs. However, it is not an inevitable or necessary property of a computing system. The AV-TSX could have been constructed so that its software was stored on write-once storage (e. g., using PROM or EPROM technology). Such a design would greatly improve the integrity of the machine's code, because a virus or malicious code would have no way to permanently overwrite the device firmware (or at least, the part that is stored on write-once storage). Under such an architecture, the effects of a virus could not persist across reboots, and in particular, the act of powering down a machine between elections would eliminate the virus. An even more aggressive architecture might involve rebooting the machine after every voter, so that any malicious code that made its way into memory while one voter was voting would not be able to persist in memory to affect the next voter. The AV-TSX unit and GEMS application were not constructed in this way, and as a result a virus or security breach in one election can affect every subsequent voter who uses that machine in every subsequent election.

In contrast, the AV-OS provides significantly better controls to protect the integrity of its firmware. The Red Team has informed us that the firmware on the AV-OS is stored on an internal EPROM chip. When the AV-OS is powered up, it loads its software from the internal EPROM chip. While an individual with physical access to the AV-OS unit could overwrite or modify the firmware stored on that chip, malicious software cannot. Consequently, if the running software on AV-OS becomes corrupted with malicious software at any point, rebooting the AV-OS will reload the software from its internal EPROM chip and thus (in the absence of physical tampering) will clear out any malicious software. In other words, even if the AV-OS becomes infected by a virus, the virus cannot install itself permanently on the AV-OS. This is a significant security advantage for the AV-OS.

4.1.6 No Way to Verify Code Integrity

The Diebold AV-TSX and GEMS machines do not provide any secure way for an election official to verify whether the software resident on the machine has been modified. For instance, a cautious election official might wish to occasionally spot-check a random sample of machines to confirm that they have the correct software installed. Unfortunately, GEMS and the AV-TSX provide no way to do that securely.

The AV-TSX does report its version number when it is powered up. However, this version number cannot be trusted. If the device software had been corrupted and overwritten with a malicious replacement, the replacement could simply lie about its version number and report exactly what the legitimate software would report.

Version numbers can be helpful for detecting accidental failures to install the correct version of the software. For instance, if the certified version of the software is version 4.6.4 but the machine reports version 4.5 or 4.7, then one can be certain that the machine is running the wrong version

of software (or, alternatively, that the certified version is so buggy that it fails to report its own version number accurately). However, if the machine reports version 4.6.4, all we know is that either the machine is running the proper software or else it is running improper software that was programmed to misreport its version number. To use an analogy credited to Dan Wallach, if the airport police walk up to a stranger at an airport and ask him "Are you a terrorist?" and he replies "No", have we really learned anything? Similarly, if we ask a voting machine "Are you running malicious software?" and the voting machine's software reports that it is not, there is no reason to trust it. Consequently, while self-reported version numbers may be helpful at detecting accidental misconfiguration or inadvertent error, they are not useful at detecting security breaches [22].

Voting machines can be built that do not have this flaw. For instance, the voting machine could be designed so that its bootloader is stored on write-once storage (PROM or EPROM). The bootloader could make a record of the cryptographic hash of the software that it loads and print that record on the zero tape[4] before the election. Or, the bootloader could contain a public key and could check that the software is properly signed before loading it. Or, the device could use a Trusted Platform Module or other secure hardware technology, such as that standardized by the Trusted Computing Group (TCG). Such a design would allow the operator to verify that the machine's software has not been modified or altered. Neither GEMS nor the AV-TSX has this capability.

The inability to detect malicious software, or verify its absence, makes devising effective defenses against virally propagating malicious code and other kinds of attacks more difficult. Formulating mitigation strategies to detect viruses would be easier if the AV-TSX provided a way to securely verify that the software resident on the machine has not been altered. The absence of this detection capability makes it harder to be fully confident that the voting system has not been subverted and heightens the impact of vulnerabilities that can be used to propagate malicious code virally.

This criticism does not apply to the AV-OS. The AV-OS does not need a way to detect whether its firmware has been corrupted by virally propagating malicious software, because its design already prevents this from happening in the first place, as explained in Section 4.1.5.

4.1.7 Reliance on COTS Software

In several places, the Diebold system relies on commercial off-the-shelf (COTS) software written by third parties. The use of COTS software has significant cost and efficiency advantages for the voting system vendor, as it allows the vendor to reuse existing code and avoid reinventing the wheel. However, this reliance on third-party COTS software also has security disadvantages.

First, GEMS runs on Microsoft Windows and relies on the security of the Windows operating system. Unfortunately, the version of Windows used in the Diebold system has a number of security vulnerabilities. In addition, securing Windows requires keeping the system fully up-to-date on all security patches. Unfortunately, the special circumstances associated with voting systems make it difficult to keep the Windows operating system patched and up-to-date. The Diebold system is tested and certified with a specific version of Windows; changing or upgrading that version might invalidate the certification and may not be permissible. Also, the most common way of keeping Windows machines up-to-date is to connect them to the Internet and have them regularly poll the Microsoft website to see whether there are any security patches to be installed. That approach cannot be applied here: it is not safe to connect the GEMS PC to the Internet at any time. Connecting the GEMS machine to the Internet, even just for a moment to download patches, creates a security exposure.

For all of these reasons, it is likely that the GEMS machine will be running a version of Windows that does not have the latest security patches applied and that is vulnerable to known, published attack methods. This prediction was confirmed by the Red Team's experiments [3]. When the Red Team downloaded a standard attack tool widely available on the Internet and pointed it at the GEMS machine, it immediately identified and exploited a known vulnerability in the Windows installation on the machine. As a consequence, if any malicious or compromised device were ever

[4] When the AV-OS and AV-TSX are powered up on the morning of election day, they print a *zero tape* on their thermal printer. The zero tape is intended to provide evidence that no votes are currently stored on the machine.

connected to the same network as the GEMS machine, then we expect that the device would be able to subvert the GEMS PC. We consider it unlikely that even a careful county system administrator would be able to configure the GEMS PC in a way to prevent such attacks from being successful. This is a risk with relying on commodity software that relies on frequent patching for its security.

Also, the security of GEMS against insider attacks and manipulation by a rogue GEMS operator relies on the security of Windows. Microsoft Windows was not designed with this threat model in mind. First, Windows treats the user (the human operator) as trusted. For instance, Windows does not attempt to prevent the user from executing any program of the user's choice or modifying any file to which the user has access, while GEMS security relies on preventing precisely this type of execution and modification. In particular, GEMS includes access controls as part of the GEMS application, but it does not control what the user can do by interacting directly with Windows or with other Windows applications. This design effectively allows a human operator intent on wrongdoing to bypass GEMS and tamper with the vote tallies in the election database, the election definition files, the GEMS application itself, and other data on the GEMS PC. Put another way, the security properties of Microsoft Windows do not seem in line with the security requirements of GEMS.

Other COTS software used is better aligned with the security goals of the Diebold system. For instance, both the AV-TSX and GEMS use the OpenSSL library for communication security. The OpenSSL library is a de facto industry standard and was an excellent choice for that purpose, as the threat model that OpenSSL was designed to withstand matches precisely what the Diebold voting system needs. However, even OpenSSL has bugs. Hursti [19] notes that security vulnerabilities have been reported in the version of OpenSSL used by the AV-TSX, 0.9.7e. We do not know whether these bugs can be exploited in the context of the Diebold system, but they illustrate the need to keep COTS software up to date.

A second disadvantage of reliance on COTS software is that the use of COTS software makes analyzing and gaining full confidence in the security of the voting software more difficult. Both GEMS and the AV-TSX rely heavily on third-party COTS software: for instance, the AV-TSX uses Windows CE. Diebold did not provide us with the source code to any of this third-party COTS software, presumably because its licenses forbade it. This prevented us from analyzing the COTS software to determine whether it contains any material security risks or whether there may be any deleterious interactions between the COTS software and the vendor-written software. Because the security of the Diebold voting system relies on the security properties of this COTS software, and because we were not provided with the access required to analyze the COTS software, we may not have identified all vulnerabilities related to its use.

4.1.8 Modems and Other Networks

Background: Regional processing The Diebold system supports a configuration, known as regional processing, where the county central location is augmented with several regional return centers distributed around the county. After the close of polls on election night, poll workers transport memory cards, paper ballots, and other election supplies to the nearest regional return center rather than to county headquarters. In counties with many precincts, this reduces the traffic and congestion at any one return center and enables the county to scale its processing of unofficial results on election night. In counties with polling places dispersed over a broad geographical region, regional processing makes it easier for poll workers to return materials without having to drive large distances.

Regional processing allows for hierarchical processing of election returns. At county headquarters, there is a master GEMS machine that is used to coordinate the entire election. Each regional return center has a "client GEMS" installation that is used to read memory cards, accumulate votes from the memory cards, and upload them to the master GEMS machine at county headquarters. Thus, each regional return center would have a GEMS PC as well as one or more AV-TSX machines (used for reading AV-TSX memory cards) and one or more AV-OS machines (used for reading AV-OS memory cards). The master GEMS machine performs the final tabulation, manages the authoritative election data for the entire county, and produces reports and final election results.

The client GEMS installation at each regional return center needs a way to upload data to the master GEMS. How that is accomplished appears to be dependent upon county practices. One supported method is to use a private network (e. g., a dedicated T-1 line or a county intranet). Another possibility is to use modems to communicate over the public telephone network. It appears to be technically possible to use the public Internet or to use virtual private network (VPN) software to establish a communication channel over the public Internet; we did not attempt to determine whether this is permitted by California law. The specific configuration is apparently left to the county's discretion: the system documentation does not appear to contain any prohibitions or limitations on which configurations are supported or allowed. It is our understanding that California election code may restrict which types of connections are permitted.[5]

Background: Modems The Diebold system can be configured to use modems for several purposes:

- AV-OS units contain an internal modem that can be used on election night to transmit election results to GEMS after the polls close. The GEMS server at county headquarters can be connected to a modem bank to receive election results, and the AV-OS memory card can be configured with the telephone number and other information needed for the AV-OS to dial the phone number of the modems at county headquarters. At that point the poll worker is prompted to confirm the uploading of vote tallies, and then they are transmitted over the phone. There appears to be no authentication or encryption of this phone call.
- Similarly, AV-TSX units contain an internal modem that can be used to upload election results to GEMS on election night. The AV-TSX can be configured with the telephone number, username and password to connect to the GEMS server. In contrast to the AV-OS, the AV-TSX software optionally uses SSL to encrypt and authenticate the call. SSL is a standard protocol for secure communications, and it is widely believed to be secure if used properly. The GEMS server is loaded with a public/private key pair as well as a certificate signed by Diebold. The GEMS server provides the certificate to the AV-TSX, and the AV-TSX validates that it is talking to a party with a public key that has been signed by Diebold before allowing the communication to proceed.
- In counties that use regional processing, the client GEMS PC at the regional return center can dial up the master GEMS at county headquarters over the phone and upload election results to the master GEMS for tabulation. This allows county officials to compile unofficial election-night results in a timely fashion. This connection also appears to use SSL to encrypt and authenticate communications between the client GEMS software and the master GEMS.

In all three cases, the GEMS PC must be connected to a modem bank that is connected to the public telephone network. One way to accept modem connections from AV-TSX machines is to enable Microsoft Remote Access Server (RAS) on the GEMS PC. RAS provides a way for clients to dial in and send TCP/IP packets encapsulated using the PPP protocol. It appears that, if modems are used, the GEMS modems will accept phone calls from anyone who knows the right telephone number to dial and can supply the correct username and password.

Our understanding is that once the AV-OS successfully connects to the GEMS modem and once the communication link has been established, the communication protocol between GEMS and the AV-OS is the same as that used when the AV-OS is connected to GEMS by a serial cable. If a memory card is inserted in the AV-OS, the poll worker can upload the election results on that memory card. If a blank memory card is inserted, the poll worker can download election configuration and data onto the memory card over the phone, just as would be done when writing a memory card via a central-office AV-OS unit. As far as we know, these transactions can be initiated only upon poll worker request — GEMS cannot request communication with the AV-OS.

[5] California Elections Code §19250 (f) states: "A direct recording electronic voting system shall not be connected to the Internet at any time." California Elections Code §19250 (h) states: "A direct recording electronic voting system shall not be permitted to receive or transmit wireless communications or wireless data transfers." We did not attempt to determine whether these two provisions would apply to GEMS or to regional processing centers.

Once the AV-TSX successfully connects to the GEMS modem, a TCP/IP session is established and the communication proceeds following the same protocol that is used when a central-office AV-TSX is directly connected to GEMS via Ethernet or serial cable. This protocol also allows a poll worker to download an election database onto the memory card over the phone. As far as we know, these transactions can be initiated only upon poll worker request — GEMS cannot request communication with the AV-TSX.

When regional processing is used, our understanding is that the client GEMS dials the master GEMS, uses PPP to establish a TCP/IP session, and then proceeds to communicate over TCP/IP using a proprietary protocol designed for this purpose.

This understanding of how modem communication works is based upon our examination of the source code and an interview with a Diebold software developer [1]. However, we have not had the opportunity to confirm our understanding by observing the equipment in operation, and it is difficult to be confident in this description from our examination of the source code. Therefore, it is possible that our understanding of this subject is incomplete or mistaken.

Our understanding is that modems are intended to be used for unofficial results. Modems can be used to transfer electronic results, but those electronic results must be checked in some way as part of post-election procedures. That checking could potentially involve comparing the number of votes cast against the number of voters who signed in on the roster sheet; comparing the paper summary tapes printed by the voting machines at the close of polls to the unofficial results in some or all precincts; and/or discarding the results uploaded by modem after election night and subsequently reading in the results from memory cards from scratch.

As far we can tell, it appears that it may be left to county discretion whether and how modems are used. The system documentation does not appear to contain any prohibition on use of modems. We note that there may be legal restrictions associated with use of modems.[6]

Analysis The use of modems and other shared communication networks in the Diebold voting system poses a special risk to security. The system documentation emphasizes that modems should only be used to transmit unofficial results, and implies that this addresses the security risks associated with these forms of networking. However, the situation is more complex than that.

There are two broad categories of risk associated with any kind of networking of voting equipment:

- *Communication security:* Transmitting information over a shared network opens the potential for an attacker to eavesdrop on, tamper with, or disrupt data while in transit. The Diebold system provides appropriate controls to either prevent or detect and recover from these types of threats. Transmitting only unofficial vote results, and only doing so after the polls have closed, ensures that eavesdropping is harmless, as that data will soon be made public in any event. This also ensures that tampering with that data can be detected during the official canvass. Detection does require that election officials diligently compare every vote total printed on every summary tape against the unofficial results or compare every vote total stored on every memory card against the unofficial results, which places an extra requirement on the official canvass, but this burden is manageable. By ensuring that the system has multiple fallback methods for transferring vote totals in case the network is unavailable or has been disrupted by an attacker, the system provides a way to recover from denial-of-service attacks and network failures. Consequently, we agree that the design of the voting system does address this class of risks in a responsible and effective way.

  However, this is not the only risk associated with networking voting systems:

- *Endpoint security:* Connecting a device to a communication network may introduce the potential for an attacker to attack that device and try to subvert or compromise its integrity.

[6] California Elections Code §19250 (g) states: "A direct recording electronic voting system shall not be permitted to receive or transmit official election results through an exterior communication network, including the public telephone system." It is our understanding that the election results transmitted from the regional results center to county headquarters would normally be considered unofficial election results, not official election results.

Whether such attacks are possible depends upon whether the software that interacts with the network contains any exploitable vulnerabilities. However, history has taught us that connecting software of any significant complexity to a network poses a non-negligible risk. The restriction to transmitting only unofficial election results does not help against this class of threats. For instance, when GEMS is set up to receive unofficial election results by modem on election night, this poses a substantial risk that an attacker might be able to dial in, pretend to be a unit in the field, and exploit some vulnerability or configuration error in GEMS or in the modem server or RAS server.

Similarly, in counties that use regional processing, the client GEMS at the regional return center connects to the host GEMS at county headquarters using some unspecified communication link, which could be a modem, a dedicated point-to-point link (e. g., a private T1 line), a connection to the county intranet, a VPN over the public Internet, or even a public Internet connection. Depending on how this is implemented, this may create a possibility for an attacker to subvert this communication link (perhaps as a man-in-the-middle, or perhaps by spoofing one party to the other) and try to compromise one of the two endpoints.

One of the greatest reasons to be concerned about these kinds of attacks is the size of the population of potential attackers who might have the opportunity to mount modem- or network-based attacks. When a voting system is connected to the public telephone network or to any shared network, there is a risk that a hacker anywhere in the world, not necessarily on U.S. soil and not necessarily subject to U.S. law, could attack the voting system from afar. Given the current state of computer security, such attacks might be essentially untraceable: it is often effectively infeasible to trace network-based attacks back to their source. There are many parties who might have both the motive and the opportunity to attack voting systems remotely, and there are few effective deterrents against such attacks.

We can see three strategies for defending against attacks on endpoint security. The first and most natural strategy is to completely avoid use of shared or public networks. This risk avoidance strategy is effective but it denies officials the ability to benefit from the administrative and efficiency advantages of telecommunication networks, so it may or may not be satisfactory. The second strategy is to design software that is impervious to attack. Unfortunately, this is difficult to achieve, and it is difficult to know when you have succeeded. As our analysis of the code illustrates (see Chapter 5), other parts of the code are not vulnerability-free, so there is no particular reason to expect that the code exposed to remote attack is necessarily vulnerability-free. The third strategy is to restrict the adversary's access to the communication link, to make it harder for an attacker to inject malicious data onto the communication channel. For instance, one might use a dedicated point-to-point link or a VPN or use cryptography to authenticate all traffic sent across the communication channel. This is a wise risk reduction strategy. In the case of the Diebold voting system, there are limits to how much one can restrict the attacker's access to these communication channels: because AV-OS and AV-TSX units are sent out to the field and are left unattended without strong physical security, an attacker who is capable of tampering with them while they are left unattended could obtain all of the cryptographic secrets needed to communicate with GEMS. The safest approach is probably to apply all three strategies to the best of one's abilities, including limiting communication over shared networks to the minimum necessary.

A thorough analysis of the risks requires access to working equipment configured in a way that is representative of how counties use the system in practice. We neither had access to working equipment nor information about how communication equipment is normally configured in practice, and the Diebold system documentation provided little guidance on how to configure modems or externally accessible networks to be secure.

Ultimately, we were not able to gain confidence in the security of the Diebold software if it is connected to modems or shared communication networks. The voting system seemed to rely primarily on the unofficial nature of results transmitted over modems for security, but as discussed above, that alone is not sufficient to prevent or detect subversion of the voting system.

4.2 Implementation

4.2.1 Input Validation

Input validation is one of the most important practices that developers of security-critical software must follow. Some experts estimate that approximately half of all software vulnerabilities can be attributed to failure to properly validate inputs from untrusted sources. The best practice is to establish a discipline to ensure that all inputs are validated, for instance, by checking all inputs against a template or whitelist as soon as they are read from any untrusted source and before they are used for any purpose.

We did not find a consistent pattern or discipline of input validation in the source code. Untrusted inputs are occasionally compared against a whitelist or template describing expected values but are more frequently not checked at all. Integers read from untrusted sources are sometimes bounds-checked immediately after being read but sometimes not. Strings are not usually checked for null-termination and are rarely matched against a whitelist or regular expression.

4.2.2 Defensive Programming

Defensive programming is another recommended practice. It involves checking all data provided by other software components just before using the data. Even if one expects that the source of the data has already verified the correctness of the data, each recipient also redundantly checks the data. For instance, pointers are verified to be non-null before being dereferenced, indices are confirmed to be within bounds before being used, and so on. The philosophy is that the program should be constructed to be robust against unexpected inputs and should fail gracefully even if other components contain unexpected bugs.

The use of defensive programming in the Diebold source code was variable. In a few places, the source code was written defensively, carefully checked all inputs, and appeared to be reasonably robust. In other places, the code made unchecked assumptions about the data it used, was not written defensively, and did not appear to be as robust as it could have been. We noticed that the latter appeared more frequently in places where the programmer might not have been expecting malicious or erroneous inputs (e. g., some of the code that handles data read from the election database or other files on the memory card) and in non-core code (e. g., debugging or logging code, code that is used only to print reports, or code for system administration tasks). In some cases, the absence of defensive checks did not lead to any bugs: even though the callee failed to defensively check all necessary preconditions on the inputs, all of the callers happened to establish those preconditions anyway. In other cases, the absence of defensive checks led immediately to security vulnerabilities. In all cases, the absence of local defense against buffer overflows and other error conditions creates a software maintenance hazard: as the code evolves there is a risk that a developer might, without realizing it, add a new call site that violates the implicit preconditions and thereby creates a serious vulnerability.

In many places, the failure to program defensively appeared to be of no particular import. However, in some cases, the failure to program defensively led to serious, exploitable security vulnerabilities. The reason that security engineers often recommend applying defensive programming to all code, not just code that is known to be exposed to an attacker, is that programmers often make unjustified assumptions and fail to anticipate the ways that attackers might be able to provide unexpected inputs. The failure to consistently apply defensive programming techniques probably contributed to the number of exploitable implementation-level vulnerabilities that we found.

4.2.3 Choice of Programming Languages and Libraries

The choice of programming language can have an influence on the frequency of implementation-level vulnerabilities. The Diebold system uses assembly languages, C, and C++. These programming languages are known to be prone to several common types of security vulnerabilities,

including buffer overflows, format string vulnerabilities, and integer overflows. We found instances of all these vulnerabilities in the source code we analyzed.

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, until recently, buffer overflows were consistently the number one publicly reported vulnerability [6]. Memory-safe languages, like Java or C#, effectively eliminate buffer overflow vulnerabilities, while programs written in older languages like assembly, C and C++ are known to be at risk for these vulnerabilities. Because it is so easy to make a catastrophic mistake in these older languages without realizing it, many security practitioners recommend that, all else being equal, projects where security is critical should consider using a more modern, memory-safe programming language. We do not mean to suggest that systems written in languages like C or C++ are necessarily insecure. However, programming securely in those languages requires more attention to detail and more experience with secure programming. It appears that the necessary level of care was not taken in the construction of the Diebold voting system.

The use of older programming languages can be partly mitigated by appropriate selection of libraries and other programming platforms. In this respect the AV-TSX and GEMS code can be credited with frequently using safer libraries, which partially reduces the risk associated with their use of the C++ programming language.

For instance, the AV-TSX and GEMS source code often uses the Microsoft Foundation Class (MFC) `CString` class to manipulate strings. This C++ idiom is inherently safer than using C-style strings, because `CString` was designed to avoid many of the common pitfalls associated with C-style strings (e. g., `CString` performs its own memory management and thus tends to prevent buffer overflows). However, the AV-TSX and GEMS code is not consistent in its use of these safer libraries. For instance, it also frequently uses C-style strings, and occasionally uses them incorrectly in a way that creates security vulnerabilities.

The MFC class `CString` provides the function `Format` which works similarly to `sprintf`. Unlike `sprintf`, however, `CString::Format` always ensures it has allocated the memory it needs before writing to its internal buffer. This makes `CString::Format` safe from buffer overflow vulnerabilities and much safer to use than character buffers (however, `CString::Format` does not prevent format string vulnerabilities).

Comments in Diebold's code indicate that developers were aware of the benefits of `CString` over character buffers:

uploaddlg.cpp:340

```
340  Assumes buf is large enough for a token
341  This would be better if it delt[sic] with CStrings
342  rather than with fixed buffers. Gems implemented
343  this improvement at one point.
```

Had the safer `CString` functions been consistently used, nearly all buffer overflow vulnerabilities would have been prevented. For example, in one particular function, both a `CString` and a character buffer are used. The character buffer usage leads to a vulnerability while the `CString` usage does not. See private appendix Issue 4.1 for more information.

In contrast, the AV-OS uses the standard C libraries and so remains susceptible to all of the risks associated with the C programming language.

4.3 Engineering Practices

Our interview with Talbot Iredale [1] provided useful insight into Diebold's general software engineering practices. Overall, Diebold's practices seem to be similar to those of most small- to medium-sized software development firms. These practices may be sufficient for ordinary commercial software, but they are inadequate for meeting the rigorous security requirements of voting software.

No Formal Threat Model or Security Plan The first step in designing a secure system is to write a formal *threat model*, a document which clearly identifies the assets that the system must protect and the threats that the system could face. A threat model also attempts to catalog the potential types of attackers, their capabilities, and their likely motivations. The goal of a threat model is clarify the system's security requirements so that there will be a basis for evaluating the security of potential designs of the system. We present a generic threat model for large electronic voting systems such as the Diebold system in Appendix A.

Once a threat model has been devised, the next step is to design a security plan aimed at countering the potential threats to the system and to evaluate its strengths and weaknesses in light of the threat model. The security plan should be a formal document and should clearly state how it deals with each of the threats that have been identified.

In our interview, Mr. Iredale stated that Diebold has neither a formal written threat model nor a formal security plan for its voting systems. Indeed, we found no evidence in the source code that systematic analysis of threats had been performed. Instead, the security measures that are in place appeared to be *ad hoc*. For example, the same threat often receives inconsistent treatment: the AV-TSX uses cryptography to protect some of the data on its memory cards, but the AV-OS does not — even though the threats to both types of memory cards are largely the same. Similarly, GEMS restricts what an operator can do through the GEMS user interface, but the system does not effectively control what an operator can do using the underlying Windows interface.

No Formal Security Training Diebold has about 25 developers that work on electronic voting systems, including those who focus on documentation, testing, and hardware. When new developers arrive at the company, they do not receive any kind of formal security training. Mr. Iredale states that some developers have security backgrounds but no one is dedicated to handling security issues. They have two small groups of quality assurance testers of approximately four people each, but none of them are dedicated specifically to security or red-team testing.

Weak Source Code Review Process Diebold uses standard versioning software (CVS) to manage the development of their source code. Any developer can check code into CVS and the code is not reviewed by other developers before it is committed into the repository. Mr. Iredale states that every CVS check-in causes an e-mail to be sent to developers who are responsible for reviewing the code. Initially, they do "random checks" on most of the code and do a "closer review" of the more critical portions. Although Mr. Iredale claims that 100% of the code is reviewed by another Diebold employee within a few weeks, there seems to be no formal procedure for assigning code to other employees for review. It seems possible that, without formal procedures, some source code could remain unreviewed before release. Issue 5.2.24 suggests that this is the case.

No Unit Testing or Red Team Testing There is no formal requirement to develop a set of unit tests that correspond to each piece of code checked into CVS — the option of doing this is strictly up to individual developers. The testing group will later check for correctness based on standard test plans.

This manual testing methodology seems fragile and inappropriate for security-critical software. Often, a developer who modifies one part of the code will inadvertently break another part of the code. It would be very difficult for individual manual reviewers and testers to reliably notice these latent errors. Unit tests written concurrently with the actual source code by the initial developer would be a more thorough strategy. The developer can better construct a set of unit tests that check boundary conditions, pre-conditions and post-conditions of each block of code she writes. These unit tests can then be run systematically at any time by anyone to check for errors. Unit tests should be combined with other automatic systems-level tests that are run regularly when code is checked in.

Diebold also lacks any formal procedures for "red team" testing, where the testers play the role of attackers and attempt to break into the system. This type of testing can detect different types of bugs that "white box" unit and system tests might not catch, such as illegal input handling and failure recovery.

Our analysis has led us to conclude that the design and implementation of the Diebold software does not meet the requirements for a security-critical system. We identified a number of systemic issues that were pervasive throughout the source code or that reflected flaws in the design of the voting system. We also found that the Diebold system's code fails to consistently follow sound, generally accepted engineering practices for secure software. Moreover, we have determined that these systemic weaknesses led to specific flaws that can be exploited by attackers (see Chapter 5).

Fixing these specific flaws without addressing the underlying systemic weaknesses that caused them is unlikely to render the systems secure. Systems that are architecturally unsound tend to exhibit "weakness-in-depth"—even as known flaws in them are fixed, new ones tend to be discovered. As a result, we must conclude that without sweeping changes to its architecture and to the practices under which it is developed, the Diebold system will likely continue to pose a risk to election integrity.

CHAPTER 5

Selected Specific Issues

In this chapter, we detail specific weaknesses we found in the AccuVote-OS, the AccuVote-TSX, and the GEMS election management systems. We discuss the issues, the requirements to exploit the issues, their implications on system security, and the observations that led us to our findings. This chapter is by no means a complete catalog of issues that might exist on these systems. Exact references to specific problems, such as source code excerpts and line numbers, are included in the private appendix.

5.1 AccuVote-OS

There are two types of Diebold AccuVote-OS machines: the AV-OS Precinct Count and the AV-OS Central Count. Since the AV-OS Central Count machine is not used in the field, the attack surface for the Central Count machine is significantly smaller than that of the AV-OS Precinct Count machine. The risk of attack for the AV-OS Central Count is diminished because it is physically located at the county headquarters and not used to process memory cards from the field. Therefore, we focused our efforts on the AV-OS Precinct Count machine. The following comments about the AV-OS machine pertain only to the AV-OS Precinct Count machine.

Three published reports previously exposed serious security vulnerabilities in the AV-OS:

1. Hursti first described critical AV-OS vulnerabilities in his July 2005 report, "Critical Security Issues with Diebold Optical Scan Design" [17]. He analyzed an earlier version of the AV-OS (1.94w) than we studied in this report (1.96.6).

2. A study entitled "Security Analysis of the Diebold AccuBasic Interpreter" [33] was published in February 2006 by Wagner, et al. The authors were charged by the California Secretary of State with reviewing the security implications of the AccuBasic subsystem of the AV-OS and AV-TSX in response to Hursti's findings. The study covers the same version of the source code that we were provided for this study.

3. Kiayias, et al. discovered new vulnerabilities in the AV-OS in October 2006 and reported their findings in "Security Assessment of the Diebold Optical Scan Voting Terminal" [23]. The study was supported by the Office of the Connecticut Secretary of the State. They were able to find these security holes solely by experimenting with the machine and without any access to the AV-OS source code.

The combination of security vulnerabilities in these three reports provides an attacker with numerous vectors to breach the integrity of an election run on AV-OS machines. We confirmed that many of these previously reported holes still exist in the current version of the AV-OS source code. Because of the highly exploitable nature of these bugs already found and the short time frame of this study, we spent less time looking for new attacks on the AV-OS itself and instead concentrated our efforts on other areas of the system. Our other findings, especially those concerning the

GEMS server, present additional vectors for attacking the AV-OS that build on these existing vulnerabilities.

In this section, we describe the problems we confirmed about the AV-OS Precinct Count machine.

Issue 5.1.1: *Data on the AV-OS memory cards is unauthenticated.*

The AV-OS does not use any strong authentication mechanisms to confirm that the data on the inserted memory card originates from a legitimate source. The machine always assumes that the memory card data is trustworthy and does not sufficiently validate the data before use. Thus, an attacker who can arbitrarily write to the memory card can easily modify its entire contents without being detected.

Each memory card contains the full state of the current election for that machine. All data on the memory card is stored in unencrypted form (other than the supervisor PIN[1], which while obfuscated, can easily be deciphered — see Issue 5.1.8). The memory card contents include:

- Memory card header: firmware revision number, card size, election status, counting mode (absentee or not), master card copy password, global counters (number of total ballots counted, number of election uploads, number of ballot tests, etc.)
- Election header: voting center name and number, download version number, obfuscated supervisor PIN, election title and date, election type, party code table, election configuration flags, and data checksums
- Election definition and data: lists of precincts, ballot cards, races, candidates and voting positions; race counters and candidate counters
- Audit log
- Compiled AccuBasic scripts
- Memory card heap

In order to take advantage of this vulnerability, an attacker would need to have write access to the memory card. One way to gain write access is to have temporary physical control of the card. According to the Diebold AccuVote-OS Hardware Guide documentation [9], the AV-OS memory card is stored behind a locked retaining door. The lock can easily be picked in a short time using paperclips [23]. The memory card is sealed in the machine by procedure, but attacks may be able to bypass the seal to access the card without detection. Once past the seal, the memory card can be removed from the machine and modified. To modify the card, the attacker needs an Epson RBC compatible memory card reader-writer device, which is publicly available for purchase [17]. Another possibility is for the attacker to bring a new Epson smart card from home that contains his arbitrarily-chosen election data. The attacker-controlled card can be inserted into the machine.

An election insider or an intruder who gains temporary access to the machine before or after the election could exploit this vulnerability. A well-prepared attacker would probably need only a minute or two to swap the AV-OS memory card. It is unlikely that this attack could be conducted by a voter on election day. This particular avenue of memory card access takes time, would look suspicious, and would be easily visible to others, since the AV-OS has no physical privacy screens and is often in plain view of both poll workers and other voters.

Another way to obtain write access to the memory card is through GEMS or a GEMS-impersonator. This avenue would not require physical access to the memory card. See Issue 5.1.2 for the details of this attack.

Three implemented mechanisms attempt to detect tampering of the memory card data: memory card checksums, the internal audit log, and the memory card signature. None of these mechanisms provide adequate protection and are easily be subverted by an attacker.

[1] The supervisor PIN is used by election officials to access machine setup functionality.

The primary problem is that all of these mechanisms store their values on the memory card, but an attacker can modify these values to conceal the attack. Each mechanism is described in further detail below in Issue 5.1.3, Issue 5.1.4, and Issue 5.1.5.

The lack of authentication on the AV-OS memory card data is a fundamental security flaw that cannot be overstated. This vulnerability is a stepping stone to highly exploitable attacks that make it much easier for an attacker to compromise the integrity of the election. See Issue 5.1.6, Issue 5.1.7, Issue 5.1.9, Issue 5.1.10, and Issue 5.1.11.

Issue 5.1.2: *The connection between the GEMS server and the AV-OS is unauthenticated.*

When the AV-OS is booted with an empty memory card inserted, the machine will prompt the user to initialize the card by downloading data from the GEMS server. The connection channel between GEMS and the AV-OS, which can be established over either a serial link or a modem line, is neither encrypted nor authenticated. An attacker can learn or reverse engineer the protocol that GEMS uses to communicate with the AV-OS. He can subsequently impersonate a GEMS server to the AV-OS to write into important memory card data fields.

If the memory card is not initially empty, the attacker can first use Issue 5.1.8 to learn the supervisor PIN to reach the option of clearing the existing memory card. A subsequent reboot will prompt the attacker to connect to GEMS.

Neither the AV-OS nor the GEMS server requires any form of authentication token (e. g., a password) in order to connect. An attacker can connect a GEMS clone (e. g., a laptop implementing the reverse-engineered protocol) directly using either a serial cable or a modem. The attacker can also arbitrarily change the phone number that the AV-OS uses to connect to GEMS again using Issue 5.1.8. Once connected, the attacker can control:

- The entire election header
- The entire election definition and initial data
- The compiled AccuBasic script

Writing to the memory card by impersonating GEMS is slightly weaker for the attacker than exploiting Issue 5.1.1. Whereas the previous vulnerability allows writing to any bit of the memory card, the attacker here cannot write to the memory card header, the audit log or the card's memory heap section. Still, it is possible for the attacker to use this exploit to compromise the entire machine. See Issue 5.1.9, Issue 5.1.10 and Issue 5.1.11.

Even if the connection between GEMS and the AV-OS were securely authenticated, it would still be possible for an attacker with access to GEMS to compromise the AV-OS. No longer would the attacker be able to impersonate the GEMS server, but vulnerabilities we found in GEMS could allow an attacker to modify the originating memory card data. See Issue 5.3.2, Issue 5.3.4 and [3].

After the election, the AV-OS connects again to the GEMS server to upload the results of the election. Again, no encryption or secure authentication is used. This allows an attacker to impersonate a legitimate AV-OS to the GEMS server. An imitation AV-OS would be able to upload bogus results to the GEMS server if the legitimate AV-OS had not already uploaded results. When the legitimate AV-OS attempts to upload, the GEMS server will reject the results as already uploaded.

Issue 5.1.3: *The memory card checksums do not adequately detect malicious tampering.*

The AV-OS uses non-cryptographic checksums to detect errors in the memory card, but the checksums are weak and do not protect against malicious tampering. The checksums are stored on the memory card with the data. The checksum mechanism does not provide a cryptographic guarantee of either data integrity or authenticity. It is only effective to detect non-adversarial changes to the card contents, such as transmission errors or hardware faults. A malicious attacker who modifies data on the card can easily calculate a new valid checksum to avoid detection.

In the election header of the memory card, there are eleven different 16-bit checksums values that cover most, but not all, of the data fields on the card. The checksums are checked immediately when the machine is turned on, when the machine enters pre-election mode, and when the machine enters post-election mode. During the election, the checksums are updated as ballots are processed, but the checksums are not checked.

The eleven checksums are divided into three categories:

- Header checksums covering: the election header, precinct headers, precinct cards, ballot card headers, contest (race) headers, candidate headers, and ballot card voting positions
- Counter checksums covering: contest (race) counters, candidate counters, and ballot card counters
- A text checksum covering (most) strings on the memory card, such as the name of the voting center, the election date, and the names of races and candidates. This does *not* include the compiled AccuBasic script or the audit log.

Although there is an allocated slot in the election header for a twelfth checksum named `auditLogChecksum`, it is never assigned or accessed. We assume the audit log checksum was planned for but never implemented.

The header and counter checksum algorithms are very primitive. For each checksum, the machine calculates the arithmetic sum of the corresponding data members on the memory card. For example, the candidate counter checksum is the arithmetic sum of the values of each candidate's total number of votes. If Alice has 36 votes and Bob has 23 votes, the candidate counter checksum is 59.

The text checksum algorithm is slightly more complex but still insecure. An integer counter is first initialized to zero. For each character of each string covered, the counter is incremented and the bits of the character are XORed with the integer counter. These XORed values are then arithmetically added together to create the text checksum. We can more precisely express this algorithm using the following pseudocode. (Note: This is *not* an except from the actual source code, but rather a simplified description of what it does written in an imaginary programming language.)

```
def text_checksum():
  int checksum = 0;
  int counter = 0;
  for each string s:
    for each char c in s:
       checksum += c ^ counter;
       counter++;
  return checksum;
```

The checksums are only effective for detecting accidental errors in the memory card data. The checksums are completely ineffective against a malicious attacker attempting to tamper with the card data fields. An attacker can easily change the checksum stored on the card to reflect the changes made to the data. Moreover, an attacker can simply ignore the checksum if he makes two or more changes that offset one another. See Issue 5.1.9 for an example of this.

The checksums are all generated locally by the AV-OS machine. Initial checksum generation happens after the memory card contents are downloaded from GEMS — the checksum is not sent with the data by GEMS. If there are transmission errors between GEMS and the AV-OS, the checksum will be calculated on the erroneous memory card data. Similarly, if a man-in-the-middle attacker tampers with the link between GEMS and the AV-OS, the checksum provides no defense.

Issue 5.1.4: *The audit log does not adequately detect malicious tampering.*

The audit log is another security measure that the AV-OS uses to try to detect machine abnormalities. Under normal operation, all major events that occur on the machine are recorded in the audit log. Each event transaction records the event type and the event time. Only two types of events, audit log initialization and memory card insertion, record the date as well. In all, there are 30 different types of log transaction events. The log has capacity for 512 transaction entries and wraps around to overwrite the first entry when the log reaches capacity. It is stored in plaintext on the memory card.

Like the checksums, the audit log is stored on the memory card and can be modified if the attacker has control of the card (see Issue 5.1.1). The attacker can arbitrarily add, modify, or remove log transactions so that the log will be consistent with the other data on the memory card after an attack. Thus, the audit log is completely insecure and should not be trusted by election authorities as evidence that no attack has occurred.

Issue 5.1.5: *The memory card "signature" does not adequately detect malicious tampering.*

The AV-OS uses a struct called `MemCardSignature` to check whether the memory card was switched while a ballot card was being scanned. The `MemCardSignature` is not a signature in the cryptographic sense, but rather a weak check on the consistency of three counters on the memory card. It consists of three integer counter values: the total number of ballots, the number of absentee ballots, and the number of non-absentee ballots counted so far. The sum of the latter two values should equal the first value. The signature covers no other parts of the memory card. The signature is read from the memory card immediately before a ballot is scanned and the saved signature is checked against the signature of the memory card immediately after the scanning is complete. If the signatures differ, counting is aborted and the machine halts.

Since all three counters are stored on the memory card, this vulnerability allows an attacker to swap the memory card in the AV-OS with his own malicious memory card as a ballot is being scanned. For example, if the attacker is the first voter of the day, the three counters would all be zero. As long as the new memory card has the same signature (i. e., counter) values, the attacker can modify other parts of the memory card (e. g. the number of votes for each candidate) without detection.

Issue 5.1.6: *Buffer overflows in unchecked string operations allow arbitrary code execution.*

There are at least four buffer overflow vulnerabilities in the code used to upload election results to the GEMS server. When the machine prepares the memory card data to be sent to GEMS, it uses the unsafe `sprintf` function to write data to a buffer string `buf`. When one element of the format string argument to `sprintf` is a string that comes from the memory card, an attacker may be able to overflow `buf` on the stack. If the attacker controls the memory card (see Issue 5.1.1), he can delete the null-termination character of the string element to extend its length to be larger than the size of `buf`. The code here is not defensively-programmed and incorrectly assumes that the strings on the card are never longer than expected.

We believe that each of these vulnerabilities can be used by an attacker who controls the contents of the memory card to execute arbitrary code on the AV-OS. One of the four overflows occurs in the code immediately before uploading the memory card's election header to GEMS. If the attacker overflows `buf` and takes control of the machine at this point, he can modify both the election results on the card and the results to be uploaded to GEMS.

See private appendix Issue 5.1.6 for more information.

Issue 5.1.7: *Integer overflows in the vote counters are unchecked.*

The AV-OS does not check for integer overflow in the counters that keep track of candidate votes. Each candidate vote counter is a 16-bit unsigned integer, which can hold up to 65,535 votes. If the vote counter reaches its maximum value and more votes are added, the counter

will wrap around to zero and continue counting. The counter will overflow without warning. Because of the unchecked overflow, a large counter value is equivalent to a small negative counter value. For example, a counter value of 65,526 is equivalent to a counter value of -10 because adding 10 votes to either value results in a counter value of 0.

This vulnerability could allow an attacker to undetectably switch a predetermined number of votes from one candidate to another. This attack was first demonstrated by Hursti [17], and we confirm that the AV-OS remains vulnerable. To see an example of this attack, see Issue 5.1.11.

Issue 5.1.8: *The machine does not adequately protect the supervisor PIN.*

The PIN used by election supervisors to administer an election is stored in obfuscated form on the memory card. The obfuscation procedure is a linear function that can easily be reversed without special knowledge. This was first shown by Kiayias et al. [23], who had physical access to an AV-OS but no access to the source code. The obfuscated form obf of a supervisor PIN pin is as follows:

$$\mathsf{obf} = \text{encode}(\mathsf{pin}, \mathsf{key}) = \mathsf{pin} + (\mathsf{key} \times \mathsf{MAGIC}_1) + \mathsf{MAGIC}_2$$

where MAGIC_1 and MAGIC_2 are hard-coded 20-bit magic constants. The value key[2] is stored in cleartext at a fixed location on the memory card adjacent to obf. To recover pin, an attacker could compute:

$$\mathsf{pin} = \text{decode}(\mathsf{obf}, \mathsf{key}) = \mathsf{obf} - (\mathsf{key} \times \mathsf{MAGIC}_1) - \mathsf{MAGIC}_2$$

Anyone with access to the machine for a few minutes can reverse-engineer both the magic constants and the formulas used to encode and decode the PIN [23]. The same magic constants are apparently used on every AV-OS machine.

This privilege escalation vulnerability allows anyone with read access to the memory card to learn the supervisor PIN. An attacker can use the PIN to access supervisor functions, such as changing system setup parameters (the GEMS dial-up phone number, feeding options, etc.), duplicating the memory card, or clearing the memory card.

See private appendix Issue 5.1.8 for more information.

Issue 5.1.9: *Votes can be swapped or neutralized by modifying the defined candidate voting coordinates stored on the memory card.*

The Kiayias report identified an attack against the AV-OS that can be mounted by anyone who can tamper with the AV-OS memory card [23]. The memory card contains a map of the layout of the paper ballot, listing for each candidate the location where that candidate's bubble can be found. A mark in that location will be counted as a vote for that candidate. Locations are identified by $(row, column)$.

The report points out that it is possible to modify those locations on the memory card to ensure that marks for a candidate will not be counted. For instance, if the attacker modifies the voting coordinates for a particular candidate to point to some other location on the ballot where no marks are likely to occur, then it is likely that the machine will never detect any votes for that candidate.

As the Kiayias team discovered, the checksum on the contents of the memory card does not prevent this attack. For example, if the attacker modifies a candidate's voting coordinates from (*row*, *column*) to ($row - 1$, $column + 1$), the ballot card voting position checksum stays constant. Their report also reveals that it is possible to swap two candidates, so that a mark

[2] As an aside about code quality, the key used in the above pseudocode is actually named `publicKey` in the source code. It is a 16-bit unsigned integer stored in the election header. It is clear that `publicKey` does not refer to a cryptographic public key as computer security experts would use the term. Rather, it is just a random nonce used in the obfuscation code that resides on the memory card. The terminology used is non-standard and can be confusing or misleading.

for Smith is counted as a vote for Jones and vice versa. In general, an attacker who can tamper with the contents of the memory card can rewrite the map of the ballot and cause ballots to be miscounted by the AV-OS in a controllable way.

We confirmed that this vulnerability is present in the version of the AV-OS source code that we examined. The Red Team has also confirmed this attack [3].

Issue 5.1.10: *Multiple vulnerabilities in the AccuBasic interpreter allow arbitrary code execution.*

Diebold uses a scripting language called AccuBasic to customize the format of reports printed on the AV-OS, such as the Election Zero report (printed before the election) or the Election Results report (printed after the election). An AccuBasic script is stored on the memory card, and the AV-OS software interprets the script from the card. An earlier study commissioned by the California Secretary of State revealed that the AccuBasic interpreter software on the AV-OS contains security vulnerabilities that allow a malicious AccuBasic script to compromise the integrity of the AV-OS [33]. In particular, a carefully crafted AccuBasic script can, when it is executed, exploit bugs in the AccuBasic interpreter to inject malicious code into the AV-OS and then begin executing that malicious code. Consequently, an attacker who can tamper with the AccuBasic script can take control of the AV-OS machine and cause it to misbehave in any way that the attacker chooses (e. g., misrecording votes).

We confirmed that the flaws discovered earlier exist in the AV-OS (we examined the same software that was examined earlier). We summarize briefly the impact of these vulnerabilities. See the earlier report for full details [33].

The AccuBasic script on the memory card is written by GEMS. If an insider with access to GEMS is malicious, he could replace the legitimate AccuBasic script with a malicious script to take over the machine. This means that the security of GEMS is critical: any compromise of its security could affect every AV-OS machine in the field.

Also, it would be possible for an individual with unsupervised access to the memory card to tamper with its contents and introduce a malicious AccuBasic script. Because the data on the memory card is not cryptographically protected (see Issue 5.1.1), anyone who has access to the memory card can modify its contents or swap it for a prepared "evil twin" card. That could allow someone who had unsupervised access to the memory card to compromise the software on the AV-OS machine that it is inserted into.

This vulnerability could help a virus to spread. If GEMS is infected, it can infect every AV-OS memory card that is written. Inserting an infected memory card into an AV-OS machine will allow execution of malicious code on the AV-OS once the operator prints any report. Also, any individual who has physical access to a memory card can infect it. This vulnerability alone is not enough to create a virus that spreads from AV-OS to AV-OS, but it could be one building block that a virus writer might use, in conjunction with other vulnerabilities. See Section 3.1 for further details.

Issue 5.1.11: *A malicious AccuBasic script can be used to hide attacks against the AV-OS and defeat the integrity of zero and summary tapes printed on the AV-OS.*

We confirmed that many of the vulnerabilities identified by Hursti in his July 2005 report [17] remain present in the current version of the AV-OS.

For instance, Hursti identified an attack that combines integer overflows (Issue 5.1.7), memory card tampering (Issue 5.1.1), and a malicious AccuBasic script to cause the AV-OS to transfer votes from one candidate to another. We briefly outline the attack here. Suppose that the attacker wants to fraudulently transfer ten votes from Brown to Smith. The attack proceeds as follows:

1. The attacker sets all the vote counters on the memory card to zero, except that Brown's counter is set to $65,526$ (namely, -10) and Smith's is set to 10. The ballot box has now been pre-stuffed: Brown starts out at a disadvantage, and Smith starts out at an advantage.

2. The attacker writes a malicious AccuBasic script onto the memory card. This script provides a custom format for the Election Zero report so that the tape will show zero votes for Brown and Smith even though the vote counters on the memory card are non-zero. However, the script for printing the Election Result report is left unmodified, so that the report accurately prints the values of the vote counters on the memory card.
3. When the AV-OS is powered up on election morning with this memory card inserted, it will print a fraudulent Election Zero report showing all zeros (even though the memory card's electronic ballot box has been pre-stuffed).
4. As voters vote, the vote counters will be incremented accordingly. On the 10th vote for Brown, the vote counter for Brown will overflow and wrap around and become zero, though there will be no way for anyone to notice that this has happened.
5. When the polls are closed at the end of the day, the AV-OS will print a Election Result report showing the contents of the vote counters on the memory card. If voters cast 56 true votes for Brown and 44 votes for Smith, the memory card will show 46 votes for Brown (because it was pre-loaded with -10 votes for Brown) and 54 votes for Smith, so the report will incorrectly show that Smith is leading Brown 54 to 46. When poll workers compare the number of voters who signed in against the total number of voters in that contest, they will not find any discrepancies, since this attack did not alter the total number of votes cast: it only shifted votes from Brown to Smith.
6. When the memory card is returned to county headquarters, GEMS will read the (fraudulent) tallies on the memory card, namely, Smith: 54, Brown: 46. When county officials perform the official canvass, there will be no discrepancies between the Election Zero report, Election Result report, and unofficial electronic tallies.
7. If the paper ballots in this precinct are selected to be manually counted during the 1% manual tally or during a 100% manual recount, the fraud will be revealed. However, this is the primary way that the attack can be discovered. If attackers try to tamper with the results in hundreds of precincts, it is likely that the 1% manual tally will detect the fraud in at least one precinct. However, if attackers only tamper with a few polling places, the attack is unlikely to be detected.

This attack requires the ability to make unauthorized changes to the contents of the memory card. Anyone with unsupervised physical access to an AV-OS memory card could attack the machine in this way (see Issue 5.1.1). A malicious GEMS operator could make these changes. Likewise, a security breach/infection of the GEMS PC could allow someone to mount this kind of attack (see Issue 5.1.2).

Hursti also identified the possibility for malicious AccuBasic scripts to print malicious messages to the LCD display or cause the machine to crash under certain circumstances. Those attacks remain possible but appear to have less severe consequences.

Issue 5.1.12: *The physical paper ballot box deflector is under software control.*

The Diebold AV-OS Hardware Guide documentation [9] states that the physical ballot box has two compartments: a primary compartment and a secondary compartment. There is a physical ballot card deflector that determines into which compartment a ballot card is deposited after it is scanned. The direction of the deflector changes depending on programmable sort conditions that are set for each election. For example, ballots with write-in candidates can be deflected into the secondary compartment so election officials can tally these manually after the election closes.

An attacker who controls the machine (see Issue 5.1.1 and Issue 5.1.2) can arbitrarily move the deflector for each ballot card. This can potentially disenfranchise voters if, for example, ballots with write-in candidates are not deflected to the secondary compartment for manual review. Voter privacy can also be affected by using the secondary bin to single out ballots of interest.

5.2 AccuVote-TSX

These are some of the specific issues we identified in the AccuVote-TSX system:

Issue 5.2.1: *The AV-TSX automatically installs bootloader and operating system updates from the memory card without verifying the authenticity of the updates.*

The AV-TSX includes a software update mechanism that allows new bootloader and operating system software to be installed from a memory card inserted into the machine. When the machine boots, it searches the memory card for specially named files. If it finds files named `eboot.nb0` or `nk.bin`, it replaces the bootloader software or Windows CE operating system image, respectively, stored in its internal flash memory with the contents of the files. These filenames have been previously published on the Internet [18].

Hursti was the first to discover that the AV-TSX has no mechanism for checking the authenticity of these software updates [18]. While the machine does employ a simple checksum to make sure the files have not been garbled in transmission, it fails to utilize a digital signature or other mechanism that would prevent an attacker from using the software update feature to install malicious code.

This means that an attacker can create malicious software updates containing arbitrary code, and the software update mechanism will not be able to distinguish these from legitimate upgrades. An attacker who has temporary physical access to a memory card — or control of any machine into which a memory card is inserted — can place his own malicious software update files on the card, and this software will be installed on any AV-TSX machine that is booted with that card in place.

Alternately, an attacker with unsupervised physical access to the machine for as little as a minute could replace the installed memory card with one containing a malicious software update prepared earlier, boot the machine to install the update, and then reinsert the original memory card. This attacker would need to bypass the lock on the memory card door, but we were notified by the Red Team that this can be accomplished quickly using only a ball-point pen. Tamper-evident seals might be used to deter attacks, but creating a seal that requires a sophisticated attack to defeat remains beyond the state of the art [20].

Furthermore, the AV-TSX machine installs bootloader and operating system updates without asking the local user for confirmation. While it does display a message indicating that an update is taking place, this message is displayed in a small font and could easily be overlooked by poll workers. The lack of confirmation increases the odds that this issue could be used to spread voting machine viruses, which we discuss in Section 3.1. The machine does not maintain a log of software updates installed via this mechanism.

Feldman, et al. demonstrate how an identical update mechanism used by the AccuVote-TS model DRE could be used to spread malicious software that would alter the electronic vote records and totals [14]. We believe that similar attacks are possible on the AV-TSX.[3]

One mitigation strategy that has been suggested in the past is to seal the memory card inside the voting machine before the machine is delivered to the polling place. This might make it more difficult for an attacker to access the memory card without being detected. However, it also ensures that the memory card will be installed in the machine when the machine boots. As a result, if the central office AV-TSX that initializes the memory card is compromised so that it stores a malicious software update on each card while initializing it, the AV-TSX machines in the polling places will automatically install the malicious update when they boot on election day.

[3] Diebold has removed another service feature that has been criticized by previous reports [18, 14]. In older versions of the AV-TSX, the system would boot into Windows Explorer instead of the BallotStation application if a file named `explorer.glb` was present on the memory card. This would allow an attacker with physical access to the machine to install malicious software manually. Since this feature has been disabled, attackers would need to exploit issues such as Issue 5.2.1 or Issue 5.2.2 to access Windows Explorer.

Issue 5.2.2: *The AV-TSX automatically installs application updates from the memory card without verifying the authenticity of the updates.*

A second software update mechanism operates after the AV-TSX boots into Windows CE. The Windows kernel launches a program called `taskman.exe`, which eventually starts the BallotStation application. Before running BallotStation, `taskman.exe` searches the memory card for files with the extension `.ins`. These are software update packages in a Diebold proprietary format. Each `.ins` file contains instructions for installing one or more files onto the machine's file system, along with the data to be placed in those files. (The machine's root file system, \, is backed by RAM, so its contents disappear when the machine reboots. The internal flash memory is mounted in a subdirectory called \`FFX`, and removable memory cards are mounted in a subdirectory called \`Storage Card`.)

Like the bootloader-based update mechanisms described above, the `.ins` update mechanism does not attempt to verify the authenticity of the updates, and it does not maintain a log of software updates that have been performed [18]. Unlike the other mechanisms, it does ask the user to confirm each update by touching a button on the screen. However, the system trusts update files to accurately describe their contents. Malicious updates could inaccurately describe their purpose, possibly fooling operators into consenting to their installation.

Attackers could use this mechanism to install arbitrary code onto the voting machine, such as by replacing the BallotStation executable file with an altered version. They could also replace other files on the system or destroy data by replacing it with garbage. Conducting such attacks would require the ability to write the update file onto a memory card as well as the operator's consent; however, Issue 5.2.3 describes a way to avoid the need for consent.

Issue 5.2.3: *Multiple buffer overflows in `.ins` file handling allow arbitrary code execution on startup.*

This issue was first reported by Feldman, *et al.* in their study of the AccuVote-TS [14]. The code in `taskman.exe` responsible for installing the application updates described in Issue 5.2.2 contains multiple buffer overflows in the way it parses `.ins` files that could allow an attacker to execute arbitrary code. For example, a malicious `.ins` file could modify files in the machine's filesystem or launch programs, and it could do these whether or not the user consented to its installation.

To exploit these previously-known vulnerabilities, an attacker would need the ability to write files onto the memory card. Since the machine does not verify the authenticity of `.ins` files, no secret keys would be required.

See private appendix Issue 5.2.3 for more information.

Issue 5.2.4: *Setting a jumper on the motherboard enables a bootloader menu that allows the user to extract or tamper with the contents of the internal flash memory.*

As previously disclosed by Hursti, the motherboard inside the AV-TSX contains a jumper header marked "Debug" [18]. When a jumper is installed here, the machine's bootloader displays a service menu over its serial port when the machine boots. A feature of the service menu called the "mini monitor" allows arbitrary memory locations to be read or written over the serial port. Since the system maps its internal flash memory into the address range 0xA0000000–0xA4000000, the mini monitor can be used to extract or alter the data stored there.

To access this feature, an attacker would need physical access to the inside of the voting machine. The plastic case of the AV-TSX can be removed by unscrewing a small number of screws. After opening the case, an attacker would need to close the jumper circuit by placing a paperclip or small wire against the marked terminals on the motherboard while booting the machine [3]. To interact with the menu, the attacker would need to attach another computer to the machine by connecting a serial cable to the "VIBS Keypad" port (in the rear of the machine). Broken security seals might provide some evidence of the attack, but the machine does not maintain any log files that would show that the service menu was accessed or which commands were issued.

An attacker could use the mini monitor to alter any part of the voting machine's software, including the bootloader, operating system, and BallotStation application. The mini monitor can also be used to create a perfect copy of the internal flash memory, including the software that runs the machine, records retained from past elections, and cryptographic keys stored there. This would be useful in constructing future attacks. Reading or writing the entire flash memory using this method would require several hours of continuous access to the machine, but an attacker could install small software changes or read short memory items like keys in seconds. This method could be used to extract the secret keys from the machine, facilitating other attacks that rely on modifying signed or encrypted files.[4]

Issue 5.2.5: *Keys used to secure smart cards and election data are not adequately protected.*

The AV-TSX uses security keys for various security purposes, including authenticating election definition files, encrypting and authenticating ballot result files, validating voter smart cards, and generating ballot serial numbers. Older Diebold machines used hardcoded keys set when the software was compiled. The version of the AV-TSX that we studied retains those hardcoded keys but also allows county election officials to change the keys that the machine uses. Officials can set three keys, the 64-bit *Smart Card Key*, the 16-bit *Smart Card Magic Number*, and the 128-bit *Data Key*. (Internally, these are labeled `SCKey`, `SCMagic`, and and `DESKey`, respectively, though the system no longer uses the DES cipher.)

The machine stores the Smart Card Key, Smart Card Magic Number, and Data Key in a file in its internal flash memory. This file, `bs-security.cf`, resides in the same directory as `BallotStation.exe`. BallotStation encrypts the contents of the file (using AES128-CBC) with a third key called the *System Key*. However, the value of the System Key is not a secret — rather, it is the hash of the machine's serial number. The serial number is stored in the machine's registry (`HKLM\Software\Global Election Systems\AccuVote-TS4\MachineSN`), displayed in the user interface (the parameter "SN" at the bottom of every screen), and printed on the Results Report and other printouts.

As a result, any party with the ability to read data from the machine's internal flash memory can learn the values of Smart Card Key and Data Key. For example, an attacker with temporary physical access to the inside of the machine's case could exploit Issue 5.2.4 to read the contents of the `bs-security.cf` file and the registry key containing the machine serial number. The attacker could compute the System Key from the serial number then use it to decrypt the other keys.

Furthermore, malicious code running on a machine can automatically obtain the System Key, Smart Card Key, Smart Card Magic Number, and Security Key using this method. An attacker who can inject malicious code into the machine can use it to discover the keys without opening the machine's case. Alternatively, the attacker could program his malicious code to obtain the keys and use them directly to carry out another attack.

This attack may be particularly damaging because the design of the Diebold system makes it difficult to use different keys on different machines. Consequentially, all machines within a particular county most likely share the same Smart Card Key and Data Key. An attacker who can extract the keys from a single machine can therefore use them to attack all of the machines and memory cards in the county.

A malicious party who obtains these cryptographic keys can perform a number of attacks, which are described in detail later in this section. These attacks include:

- Tampering with election data files, and viewing or tampering with the electronic records of election results and system logs (Issue 5.2.6)
- Creating voter cards, supervisor cards, and election administrator smart cards (Issue 5.2.7)

[4] It is possible that an attacker with access to the inside of the machine's case could also read and write the internal flash memory using the motherboard's JTAG interface.

- Determining the order in which ballots were cast based on their serial numbers (Issue 5.2.21)

Issue 5.2.6: *Malicious code running on the machine could manipulate election databases, election resources, ballot results, and audit logs.*

Four types of data files are used in elections: election databases, election resource files, ballot results files, and audit logs. These files are stored on the removable memory cards, with backups stored in the machine's internal flash memory.

Election database files (`.edb` files) contain information about races and candidates as well as other ballot text. They are not encrypted, but they are authenticated using a kind of message authentication code (MAC). The MD5 hash of the body of the file is encrypted with the AES128-ECB algorithm using the Data Key (see Issue 5.2.5).[5] The machine requires this encrypted hash to match a value in the header of the ballot definition.

Election resource files (`.xtr` files) contain RTF strings, AccuBasic scripts, audio, images, and other data used during the election. The resources in these files are not encrypted, but each resource is individually authenticated using a MAC like the one used for the election database.

Ballot results files (`.brs` files) contain a record of the votes for each ballot cast in the election. The record of each vote is individually authenticated as in the election resource files. Each vote record is also encrypted with the AES128 algorithm[6] in CBC mode using the same Data Key.

Audit logs (`.adt` files) store a partial record of BallotStation's operation. They are authenticated and encrypted using a similar format as ballot results files, except that they use the System Key (see Issue 5.2.5) in place of the Data Key.

An adversary who obtained access to the Data Key as described in Issue 5.2.5 could carry out various attacks on these files. If the attacker had access to the memory card prior to the election, he could tamper with election files and election resource files in order to exploit other security vulnerabilities in BallotStation. Some vulnerabilities, such as Issue 5.2.15 and Issue 5.2.13 below, could allow the attacker to execute arbitrary code on the voting machine during the election. The attacker could also attempt to alter the ballots in subtle ways that would confuse voters or otherwise disrupt the election.

An adversary who knows the Data Key and has access to the memory card after the election can carry out other attacks by defeating the encryption and authentication of the ballot result files. If the attacker accesses the memory card before the votes on it are loaded into GEMS, then he can tamper with the ballot results file in order to exploit security vulnerabilities in the BallotStation terminal connected to the GEMS server (see Issue 5.2.16). Using this technique, he might be able to infect a large number of voting machines with a voting machine virus, as described in Section 3.1.

An adversary with knowledge of the Data Key and access to the memory card or voting machine after the election could defeat the encryption on the ballot result files to discover the time at which each vote was cast, potentially compromising voter privacy (see Issue 5.2.20).

An adversary with only access to the memory card or the files on the voting machine during or after the election could decrypt or tamper with audit logs that are secured with the System Key, which is not a closely guarded secret (see Issue 5.2.5). By tampering with the logs, the attacker could remove evidence of his activities. The audit logs also contain sensitive debugging information, such as stack traces recorded during errors in the software, that could help a malicious party develop further attacks.

Attacks like the ones described above could be carried out automatically by malicious code installed on the voting machine. In that case, the attacker would not need to have physical access to each machine or memory card being attacked.

[5] This non-standard MAC construction was first disclosed publicly in [33].
[6] This is consistent with Diebold's public statements (see [8]).

In the final days of this study, we learned of a recent security analysis of the AV-TSX conducted by Kiayias, et al. [24]. They found several cryptographic-related vulnerabilities in the AV-TSX that we overlooked. For instance, they found that incorrect MAC values on a record cause the record to be silently ignored but processing continues, rather than halting operation with an error message, allowing an adversary to effectively remove candidates from the ballot. Also, they found that, if a pair of candidates have names of the same length, an attacker with access to the memory card can swap the candidates by swapping their corresponding RTF-string resources, causing votes for Smith to accrue to Jones and vice versa. Due to flaws in the cryptography, these attacks do not require knowledge of the cryptographic keys. They were able to find these security holes solely by experimenting with the machine and without any access to the AV-TSX source code.

Issue 5.2.7: *The smart card authentication protocol can be broken, providing access to administrator functions and the ability to cast multiple votes.*

The AV-TSX authenticates smart cards using a two step protocol involving two secrets: the 64-bit *Smart Card Key* and the 16-bit *Smart Card Magic Number*. (Both can be set by election officials using Security Key Cards, as described in Issue 5.2.5.) When a smart card is inserted into the machine, the machine issues the ISO 7816-4 SELECT command with file ID 0x3D40 as data, then it issues the VERIFY command with the Smart Card Key as data. Conceptually, the file ID and key act like a username and password, which the voting machine uses to prove its identity to the smart card. The smart card will not allow the machine to read it unless the key the machine provides matches a key programmed into the card earlier.

If the voting machine provides the correct file ID and key, the smart card allows the machine to read the data stored on it, which vary depending on the type of card being used. Smart cards used by the AV-TSX include voter cards (which may be enabled or disabled depending on whether they have been used to cast a vote), central administrator cards, and supervisor cards. The latter two kinds of smart cards enable various supervisory functions. Part of the data stored on these smart cards is a field called the *magic number*. The voting machine requires the magic number in the smart card data to match the Smart Card Magic Number programmed into the AV-TSX machine by election officials. Thus, the magic number acts like a password that the smart card uses to authenticate itself to the voting machine.

This architecture allows a malicious voter who is given an authorized voter card and allowed to vote on an AV-TSX to steal the Smart Card Key and Smart Card Magic Number. To carry out the attack, the attacker would use a commercial programmable smart card such as a Java Card [32] to create a smart card that logs all commands sent to it. When the logging card was placed in a AV-TSX, the machine would attempt to authenticate itself to the card, and the card would record the file ID and Smart Card Key sent by the machine. Next, the attacker would place the authorized smart card in his own smart card reader and send it the file ID and Smart Card Key captured by the logging card. The authorized card would respond by granting access to its data, including the Smart Card Magic Number. This attack could plausibly be carried out by a voter on election day, since the voter will receive a legitimate voter card from poll workers and would be able to insert a logging card into the AV-TSX.[7]

With knowledge of the Smart Card Key and Smart Card Magic Number, a malicious party could carry out a variety of attacks involving counterfeit smart cards. Kohno, et al. [26] first described variations on these attacks in 2003, and they still appear to be feasible despite modifications to the smart card protocol.

For example, an attacker could use the Smart Card Key to reactivate voter cards that had already been used. Or, with slightly more information, the attacker could create new voter cards from blank Java Cards. The necessary data, which includes the election and polling place IDs, could be extracted from the voting machine by various methods, read from the

[7] Alternatively, an attacker could learn the Smart Card Key and Smart Card Magic Number using the attacks discussed in Issue 5.2.5.

election database on the removable memory card if the attacker had temporary access to it, or copied from a stolen legitimate voter card.

Furthermore, an attacker could construct a forged voter card that refuses to deactivate itself. Such a card could be used to cast multiple votes. Normally, the AV-TSX deactivates the voter card when the voter casts their ballot. This process involves rewriting the data on the card using standard ISO commands. A voter card created using a programmable Java Card could be designed to ignore those commands, or to report to the machine that the commands had succeeded while actually doing nothing. While each voter card contains a voter serial number that is supposed to be unique, the machine does not record this serial number for normal votes, so the duplicate votes would not be automatically discarded.

Finally, an attacker could use a variant of this attack to create counterfeit supervisor cards. To access supervisory functions, a poll worker needs to insert a supervisor card into the machine and key in a PIN associated with the card. Creating a forged supervisor card would require even less information than creating a forged voter card, since supervisor cards are not tied to a particular machine serial number or election ID. The PIN presents no additional challenge to the attacker. The system stores the PIN on the smart card itself, so the attacker can set it to a known value when he constructs the card.

Issue 5.2.8: *Security key cards can be forged and used to change system keys.*

Election officials use a specially formatted smart card called a *Security Key Card* to update the Data Key, Smart Card Key, and Smart Card Magic Number used by the AV-TSX. The ability to change these keys is a substantial improvement over earlier Diebold designs, which were criticized for using hardcoded security keys [26]. Standard procedures call for the keys to be changed frequently. Normally, Security Key Cards would be safeguarded by election officials and used only within the central election facility. However, a flaw in the way these cards are processed leaves them vulnerable to forgery.

Unlike other smart cards, which are authenticated with the Smart Card Key and Smart Card Magic Number set by election officials, Security Key Cards use a hardcoded key and magic number for authentication. All such smart cards apparently use the same hardcoded key, which is labeled in the source code as the "manufacturer's factory default" key. This key is the easily guessable sequence of bytes 0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08. The cards also use a hardcoded 16-bit magic number to authenticate themselves to the machine. Elections officials do not have the ability to change this key or magic number.

The key and magic number are hardcoded in the source code and are apparently the same for every AV-TSX machine in the nation. An attacker who has been given unsupervised access to an AV-TSX machine could easily extract these values from the machine. Alternatively, an attacker with prolonged physical access to a voting machine might be able to guess these values.

With these values, an attacker can use a programmable smart card to create a fake Security Key Card with arbitrary Data Key, Smart Card Key, and Smart Card Magic Number values. These values are stored on the smart card in plaintext form, so no additional secrets would be needed to construct a valid Security Key Card.

Before a Security Key Card can be loaded into the AV-TSX, the machine must be placed into supervisor mode. Normally this would require a supervisor smart card, but several attacks (see, e. g., Issue 5.2.7 and Issue 5.2.9) could allow an attacker to access supervisor mode without a legitimate supervisor card.

An attacker who is able to forge Security Key Cards and load them into the voting machines could use this attack to disrupt the election. By setting the security keys in a machine to random values, the attacker could prevent the machine from loading and saving election data (if the Data Key were changed) or from accepting supervisor and voter smart cards (if the Smart Card Key or Smart Card Magic Number were changed). At that point, the machine would be effectively non-functional: it could not be used to accept votes. A group of attackers

with forged cards could possibly reprogram the security keys on a large number of machines immediately before the election or even with voter access to the machine. Diagnosing these problems on election day might be difficult and lead to lengthy delays for voters [2].

To fix these problems, county officials would need to visit each affected machine and reprogram it with a legitimate Security Key Card. However, this would require the valid Security Key Card to be taken out of the secure central election facility, exposing it to a greater risk of theft or copying. An attacker who had momentary access to the legitimate Security Key Card could use a smart card reader and knowledge of the hardcoded "factory default" key to learn the security keys stored on the card.

See private appendix Issue 5.2.8 for more information.

Issue 5.2.9: *A local user can get to the Main Menu/System Setup menu without a smart card or key.*

Normally, the AV-TSX requires the insertion of a supervisor card or central administrator card before granting access to certain program menus. These include the "Main Menu" and the "System Setup" menu, from which the user can change hardware settings, reset the system clock, and perform other administrative functions. When the machine boots, if it detects a hardware failure in the smart card reader or the printer, it enters a fail-safe mode that allows the user to access these menus without a smart card.

The Red Team notified us that they discovered a way for a voter in the voting booth, using only a paperclip, to trick the machine into sensing a smart card reader hardware failure. This method could be used by a malicious party to access sensitive menus as part of other attacks.

Issue 5.2.10: *The protective counter is subject to tampering.*

For security purposes, BallotStation maintains a "protective counter" that is intended to reflect the total number of votes ever cast on the machine. However, as reported by Feldman, et al. [14], the counter is just a 32-bit binary value written to a unprotected file in the machine's internal flash memory (the filename is `\FFX\AccuVote-TS\system.bin`).

Attackers could change the value of the protective counter using a variety of methods. With access to the inside of the voting machine, they could alter the internal flash memory using the methods described in Issue 5.2.6. Or, by using any of malicious code injection methods described in this section, an attacker could install software on the machine to manipulate the counter. For these reasons, the protective counter is not a reliable defense against electronic attacks.

Issue 5.2.11: *SSL certificates used to authenticate to GEMS can be stolen and have an obvious password.*

Some counties use modems or other network connections to transmit unofficial election results from AV-TSX machines in polling places to the GEMS server. Optionally, these connections can be protected using the SSL protocol. In the Diebold system, the AV-TSX machines and the GEMS server have cryptographic certificates that include private and public keys, which are intended to provide strong encryption and authentication of the SSL connection.

On the AV-TSX, the certificate is stored in a file called `client.pem`, and in GEMS it is stored in a file called `server.pem`. Default certificate files ship with both systems, and it is unclear whether counties have procedures for changing them. It is likely that the same default certificates ship with most or all Diebold systems, so counties should not depend on them as a security measure. An attacker could steal the certificate file by exploiting problems like Issue 5.2.4 or by installing malicious code onto the machine.

Furthermore, while the default certificate files do use passwords to protect their private keys, both files use an obvious password—"diebold". Given that Diebold has used other obvious default passwords in the past [26], this likely would be among an attacker's first few guesses. Even without guessing, an attacker could learn this password by examining the Windows registry of a GEMS server or the application software of an AV-TSX, since the password is

stored without encryption in both places. (Access to the data on a GEMS server or an AV-TSX would usually be required to obtain the password-protected certificate file in the first place.)

Though this default password provides no meaningful security, there does not appear to be an easy mechanism for officials to change it to a secure value, since it is hardcoded into BallotStation (though not into GEMS). Thus, even if counties did replace the default SSL certificates with secure ones, they would be unable to switch to a secure password without upgrading to a new version of the Diebold software.

These problems limit the usefulness of SSL for protecting election transfers.

Issue 5.2.12: *OpenSSL is not initialized with adequate entropy.*

BallotStation uses the OpenSSL library to establish SSL connections. To operate securely, OpenSSL needs to be initialized with unpredictable data, which it later uses to generate pseudorandom numbers for use in security protocols.

When the AV-TSX boots, it initializes OpenSSL using two values — the current contents of the screen, and a deterministic function of the number of milliseconds since Windows CE started.[8] Both of these would be easy for an attacker to guess, especially if the attacker had access to machine. The contents of the screen will almost always be the same during this part of the startup process, and the time since startup should not vary by more than a small number of milliseconds. Consequently, the seed that is provided to OpenSSL's pseudorandom generator is likely to be guessable.

In a normal SSL connection, the client provides all of the secret entropy used to protect the data. Since the entropy provided by the AV-TSX is easy to guess, an attacker who can intercept the encrypted SSL traffic can use an off-line attack to decrypt it. If an attacker used this technique to obtain the ballot results file as it is uploaded to GEMS, he could try to discover how individual voters voted using the techniques discussed in Section 3.4.

More powerful attacks might also be possible. An attacker who can interpose himself between the AV-TSX and the GEMS server (for instance, by splicing into the telephone line), might be able to launch a man-in-the-middle attack in real time. With a modern computer, an attacker could guess about 1,000 possibilities for the initialization data every second. The seed provided to OpenSSL has so little entropy that an attack like this appears likely to succeed before the transmission would time out. This would give the attacker the ability to change the contents of the ballot results file in transit to GEMS and to exploit any vulnerabilities in the way GEMS processes that data.

In both these cases, attacking the SSL transmission is an alternative to obtaining access to the memory card. For further discussion of attacks on modem transmissions, see Section 4.1.8.

The GEMS application contains almost identical source code for seeding OpenSSL's pseudorandom number generator. The difference is that GEMS invokes the Microsoft Windows `CryptGenRandom()` function to generate the seed that is provided to OpenSSL. This is an excellent approach. The source code in the AV-TSX is an almost-identical copy of the source code in GEMS, except that in the AV-TSX source code the call to `CryptGenRandom()` (which is not available on Windows CE, the operating system used by the AV-TSX) is commented out and replaced by an insecure seeding process that operates as described above. In both cases, the code is immediately preceded by a comment that identifies this code as critical and warns that the OpenSSL pseudorandom number generator must be seeded properly.

Issue 5.2.13: *Multiple vulnerabilities in the AccuBasic interpreter allow arbitrary code execution.*

An earlier report revealed flaws in the AccuBasic interpreter in the AV-TSX [33]. We confirmed that those flaws remain present in the AV-TSX. This is to be expected, as we examined the same software that was examined earlier. The AV-OS contains similar

[8] The function is a kind of linear congruential pseudorandom number generator, the purpose of which might be to fool OpenSSL's entropy estimation system.

vulnerabilities, with a similar impact. See Issue 5.1.10 for more details about these flaws and their impact.

Issue 5.2.14: *Tampering with the memory card can result in code execution during voting.*

BallotStation reads the file `assure.ini` when it loads. The current election database is defined by whatever database is specified as "current" in the `assure.ini` configuration file. BallotStation assumes no line in the `assure.ini` file is longer than a fixed number of characters. If a line is longer than the relatively short buffer BallotStation has allocated for it, excess characters will be written past the end of the buffer. This would allow an attacker to crash the machine and (most likely) execute arbitrary code.

This attack is especially serious because `assure.ini` is neither encrypted nor authenticated. An attacker could modify this file without knowledge of the Data Key. An attacker only needs to be able to insert a malicious memory card into a machine, or modify a legitimate memory card that will later be inserted into a machine, to successfully compromise an AV-TSX machine.

See private appendix Issue 5.2.14 for more information.

Issue 5.2.15: *A malicious election resource file on the memory card could exploit multiple vulnerabilities to run arbitrary code.*

1. The AV-TSX displays text in the election resource file on the memory card at various stages of the election. This text is stored in language-specific RTF files. One such file contains the text used to show the current page number at the bottom of every ballot page (excluding the "cast ballot" screen). This text includes two `%d` characters which are replaced with the current page number and the total number of pages in the current ballot using `sprintf`. For example, the text "Page %d of %d" would be replaced with "Page 5 of 10" on page five of a ten-page ballot. The adversary could instead provide a string like "Page %d of %d%s%s%s%s%s%s%s%s%s%s", which would very likely crash the machine, or other strings to potentially run malicious code.

   See private appendix Issue 5.2.15 for more information.

2. The AV-TSX uses bitmap images to display information to the voter in a number of places. One such image is displayed to the voter before he inserts his card to vote. Since the image contains the language-specific message "Please Insert Your Card", it is included with other language files on GEMS and sent to the AV-TSX as part of the election database file. This vulnerability exists in this code and also in other places bitmaps are displayed to the user.

   A bitmap file begins with header structures describing its data. These headers contain a variety of information, including the size of the bitmap file and the size of the bitmap data. The AV-TSX, assuming these values will be equivalent, creates a buffer large enough to hold the bitmap data and then reads the entire bitmap *file* into that buffer. A malicious bitmap file could be constructed that would lie about its size, claiming to be smaller than it is. The AV-TSX's buffer would be too small to hold the data in the bitmap file, and a buffer overflow would occur.

   See private appendix Issue 5.2.15 for more information.

To exploit this vulnerability, an attacker would need either control of GEMS, control over the relevant RTF or bitmap files stored on GEMS, or access to the memory card. In the latter case, the attacker would need to know the Data Key. See Issue 5.2.5 for more information.

Issue 5.2.16: *Malicious election database files can cause arbitrary code execution on the AV-TSX when uploading elections to GEMS.*

When an AV-TSX uploads election information to GEMS, it calls a function that prints out an "election ticket" containing information drawn from the election database. There are three

separate vulnerabilities in this ticket-printing function that allow an attacker who is able to modify the election database to crash the machine and possibly to execute malicious code.

1. The variables containing election attributes are combined using `sprintf` into a buffer `buf` that is 512 bytes long. An attacker able to modify either of these variables could overrun `buf`.
2. The very next statement uses `buf` as an argument to an `sprintf`-style formatting function belonging to BallotStation's printer class. This formatting function contains a format string vulnerability that can be exploited by an attacker who is able to modify either election attribute.
3. Later in the ticket-printing function, a similar formatting function is called every time a file is uploaded. This formatting function contains a format string vulnerability that can be exploited by an attacker who is able to modify the a different election attribute.

See private appendix Issue 5.2.16 for more information.

An attacker able to compromise an AV-TSX machine could use this vulnerability to spread a virus to the central office AV-TSX when election results are uploaded to GEMS.

See Section 3.1 for more information on virus propagation.

Issue 5.2.17: *A buffer overflow in the handling of IP addresses might be exploitable by voters.*

A voter can potentially gain access to a screen used to set an IP address used by the AV-TSX machine. This address is copied to a fixed-size stack-allocated buffer (256 characters) and then stored in the registry using the Windows API function `RegSetValueEx`. If the IP address is longer than 256 characters, this results in a buffer overflow. The IP address is validated, but the validation function only checks that each segment of the IP address is not greater than 255 when interpreted as a 32-bit signed integer. Therefore, entering a string such as `000...0001.0.0.1` where the first segment of the IP address is more than 255 characters will crash the AV-TSX.

We do not know if it is possible to exploit this problem to execute malicious code. The input validation of the IP address entered into the system setup screen severely limits the degrees of freedom available to the attacker, so we suspect this vulnerability may not allow execution of malicious code.

To exploit this vulnerability, a malicious user needs access to the system setup screen. Issue 5.2.9 illustrates one way of obtaining this access.

See private appendix Issue 5.2.17 for more information.

Issue 5.2.18: *A malicious GEMS server can cause a crash on election download.*

When an election is downloaded from GEMS, the AV-TSX prints a label to be attached to the memory card. The fields printed on the label are treated improperly, resulting in format string vulnerabilities. An attacker with control of GEMS could place `printf` format specifiers (e. g., "%s%s%s%s%s%s%s%s%s%s...") in those fields to crash the AV-TSX machine. We do not know if it is possible to exploit this problem to execute malicious code.

See private appendix Issue 5.2.18 for more information.

Issue 5.2.19: *Ballot results files store votes in the order in which they are cast.*

The AV-TSX records votes in an encrypted ballot results file (`.brs`) stored on the removable memory card with an identical backup held in internal flash memory. The ballot results file consists of a series of records, one for each ballot cast. Each record includes a 32-bit ballot serial number, a 32-bit timestamp (in UNIX format), and a representation of the voter's selections. After each vote is cast, the machine simply appends another record to the end of the ballot results file.

This design potentially threatens the secrecy of voters' ballot selections, as first noted by Kohno, et al. [26]. An attacker with access to the removable memory card containing the ballot result files (or with access to the voting machine before the backup of the ballot results file is removed) could potentially decrypt the file to learn the exact sequence of votes cast. The attacker would need to know the Data Key used to encrypt the file, which could be obtained by exploiting the problems discussed in Issue 5.2.5, for example.

See Chapter 3 for a more complete discussion of attacks on ballot secrecy.

Issue 5.2.20: *Stored votes and VVPAT barcodes include a timestamp.*

As mentioned in Issue 5.2.19, ballot results files include a 32-bit UNIX-style timestamp with the record of each cast ballot. This information could allow an attacker who can access the ballot results file and who knows the Data Key to compromise ballot secrecy. For example, if the time when each voter checks in is recorded in the poll log book, an attacker with access to the log book could correlate this data with the timestamps to determine how voters voted. Alternatively, observers in the polling place could note the time when target voters cast their votes and find the corresponding vote records in the ballot results file.

The machine also encodes the time when each vote was cast as part of the barcode that is optionally printed on each VVPAT record. If election officials have enabled printing of barcodes, this design provides another opportunity for attackers with insider access to match votes with voters' identities. Procedures for handling and auditing the ballots could mitigate or exacerbate this vulnerability. In particular, if the ballot barcodes are scanned as part of the audit process, access to the resulting data must be carefully safeguarded. Fortunately, GEMS provides a configuration option to disable printing of barcodes on VVPAT records, in which case the timestamp will not be included on VVPAT records but will still be included in the ballot results file records.

Issue 5.2.21: *Ballot serial numbers are chosen using an insecure method, which may allow attackers to discover the order in which ballots were cast.*

The AV-TSX assigns a serial number to each ballot cast. The serial number is recorded as part of the election results file, optionally printed on the paper ballots in barcode form, and displayed in various parts of the AV-TSX user interface. To protect voter privacy, the AV-TSX attempts to hide the order in which ballots are cast by assigning ballot serial numbers using a cryptographic pseudorandom function.

Kohno, et al. [26], in their study of an earlier version of Diebold's software, discovered that the mechanism used to assign these pseudorandom serial numbers was insecure and could allow an attacker to recover the true order of the votes. Diebold has since updated the software to use a different method, but attacks are still possible in spite of the changes.

Today, BallotStation assigns serial numbers by (essentially) running a custom 20-bit block cipher in counter mode. The block cipher is based on a Feistel network, using AES as the round function. (Comments in the source code refer to section 14.10 of Bruce Schneier's *Applied Cryptography*.) The network consists of 8 rounds, where the ith round computes, for a 10-bit half x_i, $c = \text{AES}(x_i)$ and extracts the 10-bit substring beginning at bit $16 \cdot i$. Serial numbers are encrypting a 20-bit counter using this block cipher. The software requires serial numbers to be no greater than 1,000,000, and it skips over ones that do not satisfy this constraint.

For this scheme to be secure, the encryption key for the AES cipher must remain secret. BallotStation generates this key by taking a value called the *BRS Signature* and encrypting it with the Data Key. It generates the BRS Signature when the ballot results file is created by taking the MD5 hash of several values from the results file header, such as the machine serial number, plus the system's tick count (the number of milliseconds since the machine booted).

Most of the values from the election header are printed in election logs or displayed in the voting machine user interface, so an attacker could launch a brute force attack to guess the

system tick count. However, an easier attack may be possible, since BallotStation uses the BRS signature value as part of the filename for the ballot results files. This means that an attacker who could obtain the Data Key (see Issue 5.2.5) and determine the name of the ballot results file could quickly calculate the list of ballot serial number in the order in which they were cast. With this information, an election insider might be able to match the VVPAT records to individual voters, or an attacker with a copy of the ballot results file could determine the order in which all votes were cast, even if Issue 5.2.19 and Issue 5.2.20 were fixed.

Issue 5.2.22: *Files on the voting machine are not securely erased when they are deleted.*

BallotStation deletes files from the memory card or the machine's internal flash memory at various times during the election process. For instance, it allows election officials to delete the backup copies of old ballot results and audit logs that are stored on the machine. Deleting old these files is important for safeguarding the secrecy of the ballot, since, as described above, the contents of the files may reveal how voters voted.

However, when BallotStation delete files, it does so using the standard Windows `DeleteFile()` API. This function removes the file from directory listings, but it does not securely erase the contents from the memory card. Even after being deleted, the data will remain in the memory card until it are overwritten with other data, which may take multiple election cycles. An attacker with access to the memory card, unsupervised physical access to the machine, or the ability to run malicious software on the voting machine might be able to recover the contents of these files even after BallotStation deletes them.

Issue 5.2.23: *Logic errors may create a vulnerability when displaying bootloader bitmap images.*

When the machine boots, it uses incorrect code to display a bitmap image. Since this code only loads images from the bootloader memory, an attacker would need to be able to modify the bootloader (for instance, using Issue 5.2.1) in order to exploit this issue. However, an attacker with the ability to modify the bootloader could simply insert malicious code directly, without having to exploit the bitmap display error. Therefore, the impact of this vulnerability is low, but we describe it in detail as an example of code quality problems in the bootloader and because it underscores the challenges of thorough code review and testing.

The following is a short except of the bootloader bitmap display code:

graphics.c:253

```
253  void GlibPutPixel(UINT xx, UINT yy, Pixel_t Color)
254  {
255      // Check for library not initialized or (x,y) out of range
256      if(FrameBuffer != FALSE || (xx < USER_X) || (yy < USER_Y))
257      {
258          // Compute the frame buffer offset and write the pixel
259          FrameBuffer[FB_OFFSET(xx,yy)] = Color;
260      }
261  }
```

The `GlibPutPixel` function is used to write pixels from a bitmap into a buffer. Line 256 was intended to check that the data being written lies within the boundaries of the buffer. However, the programmer mistakenly employed logical "or" operations where logical "and" operations are required. The correct code differs in a subtle but crucial way:

```
if(FrameBuffer != FALSE && (xx < USER_X) && (yy < USER_Y))
```

The result is that the boundary checks have no effect. As long as the `FrameBuffer` condition holds, the `if` statement will succeed. As constructed, `GlibPutPixel` would allow a specially crafted bitmap file (embedded into the bootloader by an attacker) to overwrite portions of memory, possibly leading to the execution of malicious code.

Issue 5.2.24: *AV-TSX startup code contains blatant errors.*

startup.cpp:287

```
287   TCHAR name;
288   _stprintf(&name, _T(''\\Storage Card\\%s''), findData.cFileName);
289   Install(&name, hInstance);
```

Here, `name` is not a character array but a single character in memory. The `_stprintf` function expects its first parameter to be a character array, so the programmer had to use the & operator to get the address of `name`, rather than its value. The result is an obvious buffer overflow. A string that includes the filename, which could be under an attacker's control, gets copied over whatever data resides in the memory region following `name`.

That this code works at all seems purely accidental. Memory corruption occurs even when legitimate .ins files are used. An attacker who included a file with a long name or a name containing particular characters might be able to crash the program or, possibly, execute malicious code.

This bug sheds light on the vendor's software engineering practices, because it is a very unusual error for an experienced C++ programmer to make. Characters and character arrays are very different constructs in C++. Students using the language for the first time might confuse the two, but experienced programmers who understand basic concepts like pointers would be unlikely to confuse them. The probability that an experienced C++ programmer would make such a mistake or overlook it during even a cursory review of the code is exceptionally low. This suggests to us that after this code was written it was not reviewed by any other engineers at Diebold.

5.3 GEMS

In analyzing GEMS, we focused on several interfaces that serve as points of entry for external data. These interfaces allow for:

- Communication with AccuVote-TSX machines via TCP connections
- Communication with AccuVote-OS machines via raw byte streams
- User input via the graphical user interface
- Interaction with the database

There are other interfaces, such as data importation, that we were unable to examine within the limited time of our review. Therefore, other problems may exist.

Many of the most dangerous vulnerabilities that we found in GEMS relate to its use of and interactions with the database. In general, data from the database is fully trusted by GEMS. The developers apparently made an implicit assumption that malicious parties would be unable to subvert GEMS itself or to modify the database outside of GEMS. On the GEMS server that the Red Team received from Diebold, such trust does not appear to be warranted. According to the Red Team, Diebold representatives indicated that the configuration of the GEMS server that we received matches what a county typically would receive [3].

We identified the following specific issues with GEMS:

Issue 5.3.1: *GEMS uses the Microsoft Jet data layer.*

Essentially, a data layer is the component that directly modifies the underlying data store of a database. A data layer may be combined with an application layer, such as Microsoft Access, that allows users to more easily interact with that data layer. In practice, the term "Access" typically refers to the combination of the Access application layer with the Jet data layer [7].

Microsoft offers two popular data layers, Jet and SQL Server, to meet the varied needs of diverse users and organizations. According to Microsoft documentation "The strength of Access [Access/Jet] is its ease of use, rapid application development environment, and simplistic distribution... The strength of SQL Server is its more robust data integrity,

scalability, security, and manageability." [7] While Jet may be appropriate and even desirable in a wide variety of cases, other options seem more appropriate for a critical election scenario. For example, Microsoft recommended against the use of Jet in Cuyahoga County, Ohio due to the large scale of the county's election [12]. Ryan and Hoke [30] discuss additional issues with the use of Jet.

Even if GEMS used Jet with extreme caution and implemented any additional necessary features above the data layer, Diebold's choice of Jet is questionable when alternative solutions exist that seem more appropriate. Barring a compelling reason, Diebold should switch to SQL Server or another appropriate alternative.

Issue 5.3.2: *Anyone with access to the GEMS server's local disk can modify the GEMS database.*

Anyone with access to the database files could use Microsoft Access to modify them, even while GEMS is running. GEMS does not lock the entire database while it runs, so modification of any value in the database seems possible at nearly any time. This allows malicious insiders to bypass the access control and other restrictions in GEMS.

We have successfully modified a database using a hex editor, so Access is not even necessary.[9] Given access to the disk, a malicious user could create custom software to carry out elaborate attacks against the database automatically (e. g., see Issue 5.3.7). Using exploits discussed in the Red Team report, an attacker may even be able to hack into the GEMS server over the local area network and modify the database [3].

We have tested modification of election results, audit logs, candidate names, user names, user passwords (in combination with Issue 5.3.8), user access levels, and a variety of other data using programs other than GEMS.

Issue 5.3.3: *GEMS trusts the graphical user interface (GUI) to safeguard data and enforce security constraints.*

In a number of cases, GEMS relies on the state of widgets to enforce when and how users may modify data in the underlying database. For example, if a user without administrator access attempts to edit her user options, she will see a window in which the "Administrator" checkbox is blank and disabled. The disabled checkbox is the only safeguard that prevents the user from giving herself administrator access. Using a small, freely available program, we successfully enabled disabled widgets, and GEMS automatically updated the database to match those changes.

Were users without administrator access able to install similar code on a GEMS server (for example, by installing a CD), they could give themselves administrator access. Through a series of steps, a malicious user without administrator access would even be able to delete the primary administrator account.

The ability to subvert GUI-enforced controls in GEMS allows for a variety of other attacks. For example, vote totals are reported based on numeric identifiers that map to candidates. An attacker could change the mappings even after the election has begun, meaning that the GEMS server would have different mappings than the voting machines. While we have not tested this attack, we believe that it would cause vote totals to be reported for incorrect candidates (the Red Team has tested a similar attack for us — see Issue 5.3.5).[10]

Since this attack subverts the GEMS application's own user interface to modify election data, it is simpler than other ways of tampering with data in memory, and it could be carried out by an attacker with relatively little technical skill.

Issue 5.3.4: *Procedures described in Diebold system documentation place too much trust in third-party transcription and translation services.*

[9] For our tests, we needed to close GEMS prior to modification, but it may be possible to modify the database while GEMS is running using programs other than Access.

[10] Changing identifiers can change the order in which candidates appear on results reports, but election officials would need to know that the order is incorrect to catch this.

GEMS and the AV-TSX support multiple languages through language-specific text (RTF) files. Chapters 5.4.2.2 and 5.4.2.3 in the GEMS Election Administrator's Guide [10] describe the process of translating ballot information from English into a foreign language using a third-party translation agency. These processes appear to place inappropriate trust in the translation agency.

Two processes are documented:

1. (5.4.2.3) Send the entire GEMS database to the translation agency. The translation agency modifies the database directly using its own copy of GEMS. The agency then returns the modified GEMS database, which is considered to be the official election database.

   If the translation agency is malicious, it could embed malicious data into the database that will compromise GEMS (Issue 5.3.7), all AV-TSXs, and all AV-OSs in the county. The Election Administrator's Guide does state that this option "requires that the translation agency. . .not compromise the integrity of the database and the election." The guide does not note, however, that a malicious translation agency can subvert all GEMS and AV-TSX software in the county. This level of trust in translation agencies is dangerous.

2. (5.4.2.2) Send the RTF files to a translation agency by invoking the GEMS "Export RTF" function. The translation agency edits the RTF files and returns files containing translations. Officials can use the GEMS "Import RTF" function to load the translated files into GEMS.

   The Election Administrator's Guide says nothing about trust in the translation agency when following this process. This could lead a security-conscious election administrator to believe that sending only RTF files to a translation agency would result in no security problems. Unfortunately, a malicious translation agency with access to only the RTF files could crash or possibly run malicious code on the AV-TSXs used for voting (see Issue 5.2.15).

Similar issues apply to the use of third-party audio recording agencies.

Issue 5.3.5: *Race and candidate labels may be changed after GEMS has been "set-for-election."*

GEMS allows election officials to change race and candidate labels after the election data has been transferred to the voting machines. Comments in the code indicate that this feature is desired to allow for correction of spelling errors, but the comments acknowledge that this raises a "big security issue." Results from voting machines are reported by numbers mapped to candidates and race names, not by the names themselves. Thus, swapping the names of two candidates or races will cause votes to be attributed to the wrong candidates or races in GEMS.

An attacker would have to solve several challenges to exploit this weakness. The attack would need to occur on the master GEMS server — otherwise, the candidate numbers would map to the correct names on the machine that compiles results. Also, party labels cannot be swapped (we ignore the possibility of changing party affiliations using Issue 5.3.3). Since swapping names could therefore cause mismatched party affiliations, swapping two candidate names alone might arouse suspicion if results include both candidate name and party. In addition, swapping yes/no answers for ballot issues would result in a no/yes ordering that might look suspicious.

Some of these challenges do not apply in certain cases, such as party primaries, and others can be overcome. For example, by changing both candidate and race names, an adversary could cause the results:

```
Mayor
   Candidate A (Rep)  550
   Candidate B (Dem)  450
```

```
District Attorney
   Candidate C (Rep)  400
   Candidate D (Dem)  600
```

to instead be reported as:

```
District Attorney
   Candidate C (Rep)  550
   Candidate D (Dem)  450

Mayor
   Candidate A (Rep)  400
   Candidate B (Dem)  600
```

Candidate A should have won the mayoral race, and Candidate D should have won the district attorney race. Based on these values, however, Candidate B would win the mayoral race, and Candidate C would win the district attorney race.

These attacks can be easily detected during the official canvass by comparing summary tapes printed at the polling place against the official results produced by GEMS.

A better solution for spelling errors might be to allow officials to append notes to reports rather than allowing officials to change actual labels. This would also keep officials from using this flaw to accidentally or purposely cover up spelling or labeling errors that could have confused voters during the election.

Although this vulnerability is unfortunate, honest documentation of potentially dangerous design decisions and the corresponding rationale is tremendously useful for both reviewers and future developers and should be encouraged. We emphasize that the problem here is not the presence of the source code comment quoted earlier, and the solution is not to delete that comment from the code. Difficult design decisions are sometimes necessary and always warrant justification. The problem here is that, even when potential issues were documented, quality assurance processes failed to produce less dangerous alternatives.

Issue 5.3.6: *GEMS fails to filter some user input before using it in SQL statements.*

When requesting log-in credentials from a user, GEMS places the username input string directly into a SQL statement without filtering that string. This would allow an attacker to conduct a minor SQL injection attack against the database. A carefully crafted username string would allow a malicious user to gain additional database information. For example, assume that GEMS executes the query:

```
SELECT password FROM users WHERE username='username';
```

(where *username* is the user input) and compares the user's password (from the database) to the typed-in password to determine whether the user should receive access.[11] If the user already has an account, the user is able to ask true/false questions about the contents of any table in the database. For example, suppose that a malicious user `mallory` wishes to learn whether `alice` has administrator access to GEMS. Rather than entering `mallory` as her username, she could enter:

```
mallory' AND (SELECT COUNT(*) FROM users WHERE
username='alice' AND access='admin')=1 AND
'1'='1
```

This would cause GEMS to execute the SQL statement:

[11] This is a simplified version of the GEMS login process.

```
SELECT password FROM users WHERE username='
  mallory' AND (SELECT COUNT(*) FROM users WHERE
  username='alice' AND access='admin')=1 AND
  '1'='1';
```

If `mallory` enters her correct password and can log in, she learns that `alice` must be a user with administrator access. Otherwise, she learns that `alice` does not have administrator access.

While this particular vulnerability does not appear to allow modification, addition, or removal of data, this is only due to the apparent constraints of Jet. Based on our tests, Jet does not allow statements that can modify the database to appear inside `SELECT` queries, and it does not allow multiple SQL statements to be sent in one SQL query. However, relying on these limitations of the database engine is bad practice, as they may change in future versions.

Issue 5.3.7: *In several cases, GEMS trusts data from the database not to be malformed.*

Multiple buffer overflows are possible if an adversary is able to modify data in the GEMS database. Data including text preferences and district information is trusted to be less than a given length. All identified buffer overflows appear to impact data on the stack, meaning that a straightforward "stack smashing" attack would be possible.

In one case, GEMS copies data pertaining to election districts from the database into a `CString`, resulting in a `CString` that may be arbitrarily long.[12] GEMS then creates a constant-sized buffer on the stack and copies a slightly modified version of the `CString` character-by-character into the buffer — and, if the string is too long, beyond the end of the buffer. In our tests, we are able to crash GEMS by changing text in the relevant field of the database. We expect that an adversary with access to the database could use the vulnerability to execute arbitrary code on the GEMS server.

In another case, GEMS copies a table entry relating to various formatting preferences from the database to a `CString`. It then parses that string, copying an arbitrary-length substring pertaining to text formatting into a constant-sized buffer. More than one problematic path to the flawed code exists, but we believe that the buffer is on the stack in all known cases. We were able to crash GEMS by modifying any one of multiple entries in the database and believe that an adversary could exploit the flaw to execute arbitrary code.

See private appendix Issue 5.3.7 for more information.

Issue 5.3.8: *Attackers can create a valid "encrypted" password from any desired user password, without needing to know any cryptographic keys.*

To encrypt user passwords for storage in the database, GEMS calls the OpenSSL function `DES_crypt` with the plaintext password and a two character salt as parameters. To generate the salt, the program relies on the `rand` function, seeded by the present time. `DES_crypt` uses the salt to introduce disorder in the DES encryption algorithm, and it uses content from the first eight characters of the password as a key. The algorithm uses the key to encrypt a static string, and `DES_crypt` returns a string containing the two-character salt followed by the algorithm result [34, 5]. GEMS writes the returned string to the database. While GEMS calls this "encryption", it would be more accurate to describe this as hashing the password and salt using a deterministic hash function.

This process is problematic for several reasons. First, any user with access to the database can choose a secret password and salt, create an encrypted password, and replace the password in the database with the new value known. If this new password is known only to the user, this could be used to take control of other users' GEMS accounts.

[12] While Access/Jet has length limits depending on the field type, the attacker can simply change the field type to permit arbitrary-length values. This attack does not rely on changing the field type, however, as the present field type allows sufficiently long values.

Second, a user with a valid GEMS account but without access to the database could use Issue 5.3.6 to rapidly learn the salt of another user. Even a naïve approach of guessing the first character then the second character would require 128 or fewer tries, and a user could cut this to twelve or fewer using binary-search techniques. The expected number of guesses grows linearly with the length of the salt. The same techniques could be used to learn the other user's encrypted password. Given the salt and encrypted password, a malicious user would have everything necessary to guess the corresponding password without additional interaction with GEMS or modification of the database. If the password is relatively weak, this process may be almost trivial with an off-line dictionary attack [25]. If not, the process would be far more difficult: an attacker could be reduced to brute-force guessing eight random characters. Nevertheless, such attacks are not unprecedented [27], and an adversary able to obtain special-purpose hardware could conceivably learn the password in time for an election within months or fewer after learning the salt.[13] Changing passwords regularly or using both longer passwords and an encryption function utilizing longer keys might help mitigate any potential for password cracking. That said, the impact of this weakness appears to be relatively minor, compared to other issues described here (e. g., Issue 5.3.2, Issue 5.3.7).

Issue 5.3.9: *In several cases where GEMS converts signed integer values to strings, GEMS writes them into buffers that are too short.*

In several cases, GEMS copies signed integer values into ten-byte character buffers. In addition to the ten digits that a 32-bit signed integer may contain, its string representation requires a terminating character and may require a negative sign, making a buffer of twelve bytes necessary. While an attack could potentially overwrite the buffer by up to two bytes, the executable is compiled with DWORD alignment and, consequently, the data following the buffer will start at least two bytes past its end in each of these cases. This means that the flaw probably is not exploitable. Nevertheless, as pieces of code tend to migrate elsewhere in a project, these errors should be corrected. In addition, this code may eventually be used in an application without DWORD alignment.

We have not confirmed that making the integer value negative or large enough to cause a buffer overflow is possible in every one of these cases but we still consider this code dangerous.

See private appendix Issue 5.3.9 for more information.

[13] The COPACOBANA system, utilizing approximately $10,000 of special-purpose hardware, reportedly is capable of cracking DES in under nine days on average [27].

CHAPTER 6

Procedural Safeguards and their Limitations

In this chapter, we discuss the extent to which changes in election procedures can compensate for the security shortcomings of the Diebold software. Although we believe that some of the procedures that we describe below may help reduce the risk posed by these vulnerabilities, we stop short of endorsing any of them because we are not confident that anything short of a redesign of the Diebold system can provide an acceptable level of security.

6.1 Logic and Accuracy Testing

Logic and accuracy testing provides little defense against software-based attacks. Malicious software running on the machines can detect whether officials are performing logic and accuracy tests and can force the machine to behave normally until the testing completes.

6.2 Commercial Virus Scanners

Commercially available virus scanners provide little defense against the kinds of attacks described in this report. They normally are only able to recognize PC viruses that have been observed in the wild on many computers. However, they cannot detect new attacks never seen before, and they are not designed to detect virally propagating malicious code that targets voting equipment and voting software.

6.3 Stricter Chain of Custody Measures

We are not optimistic that stricter chain-of-custody controls will prove effective in addressing the vulnerabilities identified in this report. We were not able to identify any realistic procedures that would ensure that voting equipment and memory cards remain under two-person control at all times. Leaving voting machines unattended overnight in a polling place breaks the chain of custody and creates an opportunity for an attacker to tamper with the machines. Sending voting equipment home with the chief poll worker allows that person unsupervised access to the equipment; since in many counties essentially any registered voter who volunteers can become a poll worker, it is difficult to prevent an attacker from becoming a poll worker. Since it might take only one compromised machine to spread a virus to all the county's voting machines, the prospects for devising chain-of-custody rules that will meet the necessary level of perfection in practice seem dim.

6.4 Tamper-Evident Seals

We do not expect that tamper-evident seals will be effective at detecting tampering with voting equipment while it has been left unattended. First, the Diebold polling place equipment does not appear to have been designed to meet this threat model. We understand that the Red Team has identified several ways that a voter might be able to tamper with an AV-TSX machine while in the process of voting [3]. Second, most, if not all, tamper-evident seals have known vulnerabilities that could allow an attacker to break them and then replace them or restore them to a condition where the tampering is unlikely to be detected [21]. Third, it is challenging to devise protocols that make it likely that poll workers will detect and respond appropriately to tampering; few poll workers have prior training as a seal inspector, and it is not practical to provide the kind of training that would be needed in the already-rushed training that poll workers receive. Fourth, the false alarm rate (where seals are broken or unverifiable for innocuous, ordinary reasons) is so high that election workers may become inured to these issues; it may be difficult to ensure that broken seals are consistently taken seriously enough. Since it only takes one compromised machine to infect the entire county's voting machines, we do not believe that tamper-evident seals can prevent introduction of virally propagating malicious code. See Section A.5 for a further discussion of tamper-evident seals.

6.5 Forensics

Forensics performed after election day may be helpful to determine the cause and nature of attacks. However, procedures to govern forensic analysis should be in place before any problems are detected. Viruses and other malicious software could be designed to remove traces of their activities from the voting machines at the end of the election, so workers need to collect and preserve evidence even before they suspect an attack. Ideally, some number of voting machines and memory cards should be randomly selected and set aside, unused, so that any attack software present will be preserved for analysis.

6.6 Parallel Testing

Parallel testing is another partial mitigation to consider. Parallel testing involves selecting a random sample of DRE machines, taking them aside, and running a mock election on election day using the equipment. By preparing a known voting slate, one can compare the results from those machines against the inputs that mock voters entered. Typically, parallel tests are videotaped so that it is possible to go back and review any discrepancies. Parallel tests are one way to detect bugs or malice in DRE software, if the faulty software is widespread enough that the random sample is likely to pick at least one DRE that exhibits incorrect behavior.

The reliability of parallel testing at detecting malicious code appears to be open to debate. The effectiveness of parallel testing is heavily dependent upon the details of how the testing is done. If malicious software can distinguish when it is being tested from normal operations, for instance by looking for mistakes that inexperienced voters would make but officials performing tests would not, then the malicious software can evade detection by behaving correctly when it is under test.

Ultimately, parallel testing becomes an arms race between attack designers and officials who plan realistic parallel tests. The defenders attempt to design testing procedures that mimic real elections as closely as possible, while we must assume the attackers will try to design methods to detect when they are being tested. It is not clear who has the advantage in this race. The problem with this kind of arms race is that it is difficult to know who is winning. Thus, there is a risk that an attacker might develop a secret way to defeat parallel testing, leaving the defenders with a false sense of security about election integrity.

Another way to thwart parallel testing would be to use a secret knock (a series of inputs known only to the attacker that would be unlikely to happen by chance) to control activation of the vote-stealing code [4]. A secret knock could be used to activate the virus, though this would require the

virus author to have conspirators who could access each of the voting machines where votes would be stolen. Alternatively, a secret knock might serve to deactivate the vote-stealing code, though this would require the help of an insider in the parallel testing process.

Parallel testing only defends against malicious software on the AV-TSX DREs. It does not defend against malicious software at county headquarters, such as malicious software on GEMS or on the central-office AV-TSX/AV-OS units.

Parallel testing is more effective at detecting attacks than at preventing them from disrupting the election. Suppose testing reveals that a small number of votes are recorded for the wrong candidate. If the test is conducted on or close to election day, there may not be enough time to determine the cause. As described above, it may be difficult or impossible to determine the correct vote totals if attack code is running on the machines on election day. Denial of service attacks present an even greater challenge. Officials have few recovery options if they discover shortly before the election that the machines will fail the next time they are used, and parallel testing on election day provides no advance warning of such a failure.

Finally, unless parallel testing is performed on a very large number of machines, it will have a low probability of uncovering attacks that are directed only at specific precincts or election conditions. Other mitigation strategies must be applied to control these risks.

Despite these limitations, parallel testing may still have value at detecting viral attacks and human factors attacks on the VVPAT, like those discussed in Section 3.3. We leave it to others to analyze the cost-effectiveness and appropriateness of parallel testing.

6.7 Voter-Verifiable Paper Records

One of the critical security mechanisms in the Diebold voting system is the voter-verifiable paper trail, namely, the paper ballots for the AV-OS and the VVPAT records for the AV-TSX. The idea is that, in case an attacker manages to replace the certified software on the AV-OS or AV-TSX with malicious software, the paper trail will provide a way to detect misbehavior by the malicious software. Any strategy to mitigate the Diebold system's technical problems must take into account the limitations of the paper trail system. Therefore, we discuss the constraints these limitations would impose on any solution.

Voter-verifiable paper records (paper ballots and VVPATs) are perhaps the best defense against vote-stealing attacks; however, as discussed in Chapter 3, they may not be adequate to detect and recover from attacks that change only a small number of votes. The design of the paper audit trail greatly influences its effectiveness. Voters should be strongly encouraged to review the contents of the VVPAT record and to report any discrepancies to poll workers. Discrepancies should be logged and reported to election officials and centrally tracked on election day to monitor for signs of a widespread problem.

VVPATs provide little defense against most kinds of denial-of-service attacks, since the machines cannot print VVPAT records if they are not operational. Attackers may also target the VVPAT directly, for instance, by programming the machine to exhaust the supply of paper.

6.8 Ballot Secrecy Protections

The ballot secrecy problems identified in this report are difficult to mitigate. After-the-fact procedural controls seem inherently inferior to technological measures that randomize the electronic records at the time the vote is cast. We recommend that the AV-TSX software be fixed to ensure that the electronic records retain no trace of information that might reveal voter identity and to ensure that the electronic cast vote records are independent of the order in which voters voted.

Until the software can be fixed, there may be no fully satisfactory solution, but we can identify several stop-gap steps that election officials could consider adopting if they must use the AV-TSX:

- Do not record voter names in the sign-in roster sheet in the order that voters sign in. For instance, one might use roster sheets that have voter names pre-printed in alphabetical order.

- Do not use e-pollbooks that record or transmit any information about the order in which voters signed in.
- Consider introducing procedural mechanisms to ensure that county staff who are present at a polling place are not given access to the electronic or VVPAT records for that polling place.
- Limit the number of individuals with access to the GEMS network to the minimum necessary, and ensure that they can be trusted. Do not give temporary workers access to GEMS, the network that GEMS is connected to, or other devices that are connected to GEMS.
- Limit or prohibit access to the raw ballot results files stored on the memory cards.
- Use the Key Card Tool to change the cryptographic key on every AV-TSX machine in the county to a secret, county-specific, unguessable key, and establish strict two-person control over all AV-TSX memory cards that contain electronic records of voted ballots. These measures make it harder for a malicious poll worker, who is entrusted with transporting the memory card back to county headquarters, to make a copy of the electronic results files for later analysis.
- Offer every voter the opportunity to vote on a paper ballot. Optically scanned paper ballots are not subject to the ballot secrecy risks in the AV-TSX. Unfortunately, if the county does not have an AV-OS scanner in every polling place, voters who vote on paper ballots do not receive the benefit of overvote notification and thus may suffer from a higher rate of lost votes. Counties in this position should adopt procedures to minimize the rate of lost votes, such as screening all centrally counted paper ballots cast at the polls for overvotes or marginal marks and manually examining those ballots for voter intent.

These mitigations are incomplete. Even if all of them were adopted, they still would not suffice to completely address the ballot secrecy shortcomings of the AV-TSX.

The risks to ballot secrecy only apply to voters who vote on the AV-TSX. Therefore, voters who use paper ballots are not subject to these risks. The impact of the AV-TSX ballot secrecy issues will be proportional to the number of voters who use the AV-TSX; counties who use a hybrid model and encourage most voters to use paper ballots will be affected less than counties where all voters vote on the AV-TSX.

6.9 Minimizing Use of Modems and Shared Networks

Modems Modems pose a risk to election integrity. The risk is that someone may be able to dial into the modems and compromise GEMS. If someone is able to do that, they may be able to introduce virally propagating malicious code onto the server, which will then be able to infect all the voting machines in the county in the next election. We are concerned that it would be easy for software flaws or misconfiguration associated with modems to make it possible for someone to mount such an attack. See Section 4.1.8 for further analysis.

The safest countermeasure is probably to simply avoid any use of modems. Under this approach, modems would not be used for any purpose, not even to communicate unofficial results on election night, and GEMS would never be connected to any modem or to the public telecommunication system (or to any machine or network that is directly or indirectly connected to such) at any time. The advantage of this approach is that it eliminates the possibility of a physically remote attacker dialing in and hacking the voting system. The disadvantage of this approach is that it may slow down the reporting of election results, particularly for geographically distributed counties.

Regional processing The use of regional processing also poses similar risks, because it inherently involves connecting GEMS to modems or shared networks.

Shared networks pose a risk that is analogous to that posed by modems. The degree of risk may depend upon many factors, such as how many people have authorized access to the network,

how easy it may be to gain unauthorized access to the network, and what else that network is used for. At one extreme, connecting any component to the public Internet is very dangerous, because it creates the opportunity for anyone anywhere in the country to attack the system. At the other extreme, a physically secured point-to-point communication link may pose little risk. County intranets and other shared networks may fall somewhere between these extremes and the risk they pose depends upon the specific circumstances.

We mention several potential responses to this risk, in order of decreasing security:

1. The most secure countermeasure is to avoid any use of modems or other shared networks and to avoid using regional processing. This eliminates the possibility for a physically remote attacker to dial in and breach the security of GEMS.

2. A closely related variant is to use regional processing but avoid any use of modems or networks. Instead, unofficial results could be transported from regional return centers to county headquarters on write-once media (e. g., CD-R or DVD-R).

3. Another possibility is to use a physically secured point-to-point communication link, such as a dedicated T1 line leased from the phone company. This may partially reduce but not eliminate the risk. There remains the risk of configuration errors, attacks by phone company insiders, breaches of security within the phone company, and breaches of physical security at either endpoint.

4. A third approach is to use a commercial virtual private network (VPN) product to emulate a point-to-point link. This carries the risk of configuration errors as well as the risk of software flaws or cryptographic weaknesses in the VPN product. Unfortunately, configuring VPN products securely can be tricky, and determining whether a VPN product has software flaws is difficult. Therefore, this option may not be as safe as a dedicated point-to-point link.

5. Using modems is the riskiest approach. Because of the risk of configuration errors and software flaws, we would not recommend this option even to expert system administrators.

Counties may wish to re-examine the security risks associated with regional processing and assess whether the benefits outweigh the risks.

6.10 A Segregated Dual-GEMS Architecture

Another potential approach that is worth investigating involves deploying two separate GEMS installations at county headquarters, a permanent GEMS and a sacrificial GEMS. The permanent GEMS installation would be used for laying out the ballot, defining the election, and writing to memory cards before the election. The sacrificial GEMS installation would be used for reading memory cards, accumulating and tabulating results, and producing reports. The latter installation can be reformatted after the election and is never used to write memory cards, so if it is infected by a virus, at least the virus will not be able to spread to every other voting machine in the county.

This architecture is motivated by the observation that the key step in the propagation of the virus of Section 3.1 is when an infected central-office unit is used to write many memory cards destined for the field, infecting all of them. This step is what causes the virus to spread so rapidly. If we can ensure that no infected central-office machine unit is ever used to write memory cards, then we can prevent the rapid viral spread of Section 3.1.

The approach In more detail, we would have two entirely separate, isolated installations of GEMS. Each would be a complete installation of GEMS and accompanying equipment, complete with its own Ethernet network, port server, and central-office AV-TSX and AV-OS systems. The

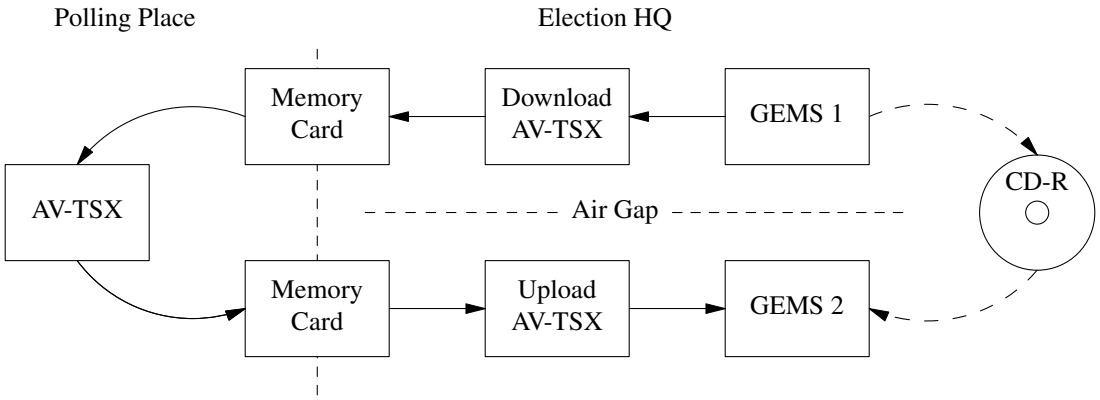

Figure 6.1: Adopting a **segregated dual-GEMS architecture** would help protect against certain kinds of viruses. Officials use one GEMS server and set of central-office voting machines to create memory cards before the election. They use a second, physically separate GEMS server and set of voting machines after the election to tabulate results. The GEMS database is transferred from the first GEMS server to the second GEMS server using a write-once medium, such as a CD-R.

two systems would be carefully segregated and air-gapped[1] to ensure that there are no cross-connections. The sacrificial GEMS installation would be treated as presumed-to-be-infected, so any machine or equipment that is ever connected to the sacrificial GEMS system must never again be connected to the permanent GEMS installation. Strict procedural controls must be applied to ensure that any media that has been connected to the sacrificial GEMS installation is securely erased or reformatted before being used with the permanent GEMS installation.

Before the election, system administrators would reformat and reinstall all the machines and software on the sacrificial GEMS installation, to bring up a clean sacrificial installation. County staff would use the permanent GEMS installation to lay out the ballot, define the election, and program all of the AV-OS and AV-TSX memory cards. Then county staff would write a backup of the GEMS database from the permanent GEMS installation onto write-once media (e. g., CD-R or DVD-R), carry the media by hand to the sacrificial GEMS installation, and install that GEMS database onto the sacrificial GEMS. After this point, the permanent GEMS installation would not be used for the remainder of the election.

On election night, as memory cards or other equipment are returned from the field, they would be taken to the sacrificial GEMS installation (*not* the permanent GEMS installation). Memory cards would be read using the central-office AV-TSX and AV-OS units that are part of the sacrificial installation. The sacrificial GEMS would be used to accumulate and tabulate election results, produce reports, and calculate the official election results.

Finally, after the election is over, all memory cards would be erased and reformatted using a separate laptop (not connected to either GEMS installation) that is used only for this purpose. This ensures that if the memory cards were carrying data infected with a virus, they have been returned to a clean uninfected state. Some mechanism would have to be devised to securely reformat the AV-OS memory cards.

System administrators could optionally reformat all devices that are part of the sacrificial GEMS installation, including the GEMS PC. System administrators could then reinstall all of the software on the sacrificial GEMS installation, in preparation for the next election. This ensures a clean copy of the GEMS software. Unfortunately, there seems to be no reliable way to clean the sacrificial AV-TSX machines, so reformatting the sacrificial GEMS PC may not be worth the effort.

[1] This is a term applied when two networks are kept physically separate to ensure that data cannot flow from one network to the other. In particular, one ensures that no device attached to the first network is connected (directly or indirectly) to the second network.

Security analysis This architecture prevents viruses introduced by outsiders or poll workers from spreading rapidly over the course of two elections, as described in Section 3.1. While every device in the sacrificial GEMS installation can be easily infected by viruses, this does not help the virus to spread further. The sacrificial GEMS installation is only used for tabulating votes, an operation that can be easily cross-checked during the official canvass by reference to the summary tapes printed in the polling site. The architecture makes it difficult for a virus introduced by an outsider to infect the permanent GEMS installation.

This architecture does not address viruses introduced by insiders. It also does not address slowly spreading viruses. The architecture would need to be supplemented with some additional practices, such as:

- No machine that has ever been used in the field or with the sacrificial GEMS installation should ever be connected to the permanent GEMS installation. Several AV-TSX and AV-OS units should be set aside for use as central-office units to be connected to the permanent GEMS installation, and they should never be used for any other purpose.
- Memory cards and other media would have to be tracked very carefully to ensure that they were never inadvertently inserted into the permanent GEMS installation, as doing so even just once could permanently infect the permanent GEMS installation. For instance, one might institute color-coded labelling and locate the permanent GEMS installation in a physically remote location to reduce the likelihood of accidental procedural lapses.
- If accumulation within a polling site is used, the assignment of AV-OS and AV-TSX machines to each polling place should be permanent. If two machines are assigned to the same polling place in one election, election administrators should avoid assigning them to separate polling places in another election, as that can enable the spread of viruses. If polling-site accumulation is used, then the presumption should be that if any machine in the polling site is infected, malicious code can propagate to infect all of the machines in the polling site. A static assignment of machines to polling sites limits the spread of viruses.
- If modems are used, they should be connected to the sacrificial GEMS installation. No component of the permanent GEMS installation should ever be connected to modems or any other shared network.

6.11 The Alternative: A Voting System that is Secure by Design

Given the costs of designing a new voting system, leaving the Diebold software largely unmodified and relying on procedural changes to mitigate the threats that we describe may seem attractive to policymakers. We consider this to be a risky approach, however, because we are not convinced that it is possible to fully resolve the security problems in the Diebold system through procedural means. We are concerned that, because the Diebold system is vulnerable in so many ways, the procedures needed to protect it would be extensive, complex, and hard to follow. We worry that despite the best efforts and intentions of election officials, the procedures would not be followed perfectly every time and the system would sometimes be left open to attack. As a result, we believe that rather than attempting to retrofit security onto a flawed system, it is safer to reengineer the Diebold system so that it is secure by design.

Building a secure voting system requires making security a central part of the design process from the start. It also demands the involvement of election administrators, experienced software architects and developers, and experts in software security and physical security. Such a system would need to use design techniques appropriate for security-critical systems, such as threat modeling, attack surface reduction, defense in depth, and privilege separation. It would need to apply sound, generally accepted engineering practices for secure software, including input validation, defensive programming, and security testing and assessment. Designing a secure voting system is an expensive proposition that requires a long-term commitment, but the ultimate benefit of doing so is increased confidence in the electoral process.

CHAPTER 7

Conclusion

Our study of the Diebold source code found that the system does not meet the requirements for a security-critical system. It is built upon an inherently fragile design and suffers from implementation flaws that can expose the entire voting system to attacks. These vulnerabilities, if exploited, could jeopardize voter privacy and the integrity of elections. An attack could plausibly be accomplished by a single skilled individual with temporary access to a single voting machine. The damage could be extensive — malicious code could spread to every voting machine in polling places and to county election servers. Even with a paper trail, malicious code might be able to subtly influence close elections, and it could disrupt elections by causing widespread equipment failure on election day.

We conclude that these problems arose because of a failure to design and build the system with security as a central focus, which led to the inconsistent application of accepted security engineering practices. For this reason, the safest way to repair the Diebold system is to reengineer it so that it is secure by design.

We discussed a number of limited solutions and procedural changes that may improve the security of the system, but we warn that implementing any particular set of technical or procedural safeguards may still be insufficient. Similarly, fixing individual flaws in the system — even all of the issues identified in this report — may not yield a secure voting system because of the possibility that unidentified problems will be exploited. We are also concerned that future updates to the system may introduce new, unknown vulnerabilities or fail to adequately correct known ones. We urge the state to conduct further studies to determine whether any new or updated voting systems are secure.

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 only to consider the security of voting systems proper, not that of California's entire election system, and therefore many possible attacks are outside the scope of this report.

A.1 Reference Model

In order to simplify our analysis, we assume a common reference model, which distills the essential features of all the voting systems involved in this study. The reference model consists of the following components.

In the polling place:

- Management stations (MS)
- Direct recording electronic (DRE) voting machines, attached to Voter-Verifiable Paper Audit Trail (VVPAT) printers
- Paper ballot optical scanner (optical scan) machines

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, ignore 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 close, 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.

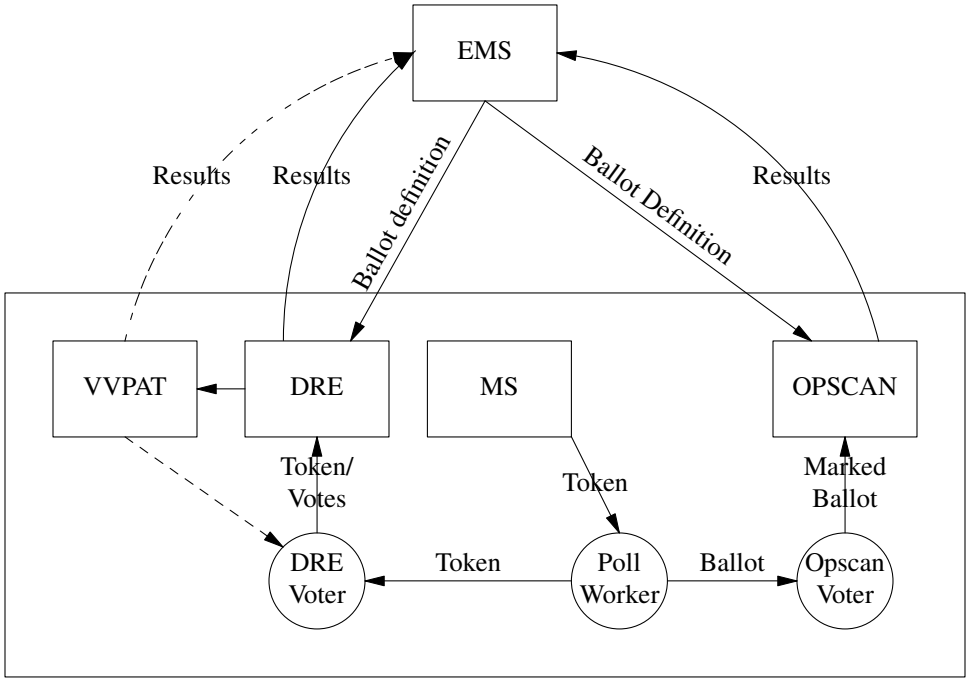

Figure A.1: Reference architecture

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 load the election definitions
- Distribute the local voting equipment to the polling places

Voting systems vary to a degree, but generally an EMS is used to create ballot definition files. These are then loaded onto some memory cards/cartridges and/or directly onto the voting machines. The vote counters in the machines are reset and the internal clocks are set to the correct time before the machines are shipped out to the local polling places or provided directly to poll workers. To protect ballot definition files from tampering, memory cards/cartridges are generally distributed with physical security measures. These consist of either 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 metal or plastic loops which, once installed, can only be removed by cutting them.

A.1.2 Voting

The exact details of the voting phase differ with each technology and manufacturer, but there are common aspects of each technology (DRE or optical scan).

Optical scan machines Optical Scan voting can be thought of as machine-counted paper ballots. When a voter enters the polling place and receives permission to vote, he is given a blank paper ballot. The voter 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 at the precinct or centrally at the county

election headquarters. When done locally, the precinct has a scanner that counts the ballots as they are inserted. Generally, the voter inserts the ballot directly into the scanner. Precinct-based optical scanners can detect "overvoting" and reject such ballots, giving the voter an opportunity to correct the error. When the election ends, the scanner's electronic records are 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 are sent back to the county's election headquarters in their original ballot box where they are tabulated with a high-speed scanner under the supervision of election officials. Central- and precinct-based tabulation may be used together in a variety of ways. Central tabulation is better suited for absentee ballots but 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 differs fundamentally from optical scan voting. Instead of entering their vote on paper, voters use a computer-based *graphical user interface* (GUI). Once each vote has been "cast," an electronic record is stored locally, in the DRE machine and possibly in the memory card/cartridge or in another attached system. At the end of the day, these electronic records may either be exported from the DRE machines to memory cards or transmitted via modems. Alternatively, the DREs themselves may be transported to the county's election headquarters. In any case, the EMS will collect the electronic records from each precinct and tabulate them electronically.

In DRE voting (as with optical scan voting), the voter enters the polling place and establishes her eligibility to vote. However, because there is no paper given to the voter, authorized voters must be prevented from casting more than one vote. In every DRE system we analyzed in this study, the poll worker uses an administrative device to issue the voter some type of token. The voter takes this token to any voting machine and is allowed to vote only once. The Diebold and Sequoia systems use a smart card as a token, while the Hart system uses a four-digit "Access Code."

Once the voting machine is activated, a voter can page through each contest and select her choices. DRE machines do not allow overvotes (too many votes cast in a contest) but do allow undervotes (too few choices cast in a contest). The voter is then presented with an opportunity to review the ballot and then commits to it ("casts" it), at which point it is recorded to local storage.

California requires DREs to use a *voter-verifiable paper audit trail* (VVPAT). On all the machines we studied, this takes the form of a sealed printer attached to the DRE that contains a continuous spool of paper. Before the voter confirms the ballot, the machine prints out a paper ballot record and displays it to the voter. Once the voter accepts or rejects the ballot, an appropriate indication is printed on the VVPAT record. When the voter casts his ballot, the record is marked as accepted and scrolled out of sight. Because the paper is displayed behind a glass panel, a voter cannot easily "stuff" additional ballots into the machine or take the record of their vote home as a "receipt."

When the election is over, the local results are transmitted to the county election headquarters, typically by shipping a removable memory device from the voting machine. The VVPAT paper rolls, perhaps still sealed in their printers, are also sent to the county for use in audits.

Typical deployments There are two common models for deploying this equipment in the polling place, DRE-only and a hybrid model. In the DRE-only model, every polling place contains one or more DRE machine and most or all voters vote on the DREs. The polling place contains no optical scan machines. In the hybrid model, every polling place contains one optical scan machine and one or more DREs. Voters either have the option of voting on paper ballots or using the DRE, or the DRE may be reserved for voters with disabilities while all others vote on paper ballots. In California, each county determines which model is most appropriate for its 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 optical scan 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

It's important to note that the second task is typically performed based on electronic records. In the most common case, precincts send back memory cards containing election results which are added directly to the tally without any reference to the paper trail (except, of course, for centrally tabulated optical scan and absentee ballots).

The 1% manual recount compares the paper records of a given set of votes to the reported vote totals. In the case of optical scan ballots, this means manually assessing each optical scan ballot. In the case of DREs, it means manually assessing the votes on the VVPAT records. Note that while in principle the optical scan 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 target specific types of voters or the populace at large. For instance, an attacker might target voters who are registered to a particular party, or voters who live within a certain geographic area for attack. The ability to determine how an individual voted also 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

Whether or not specific attacks are feasible depends on attackers' 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") and voter suppression. An attacker might:

- Arrange for some subset of machines to malfunction

- 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 attacks are selectively targeted and could be used to influence the outcome of an election. The third, more global attack is primarily useful for invalidating an election, but could also potentially be used for extortion.

A.2.3 Violating Ballot Secrecy

An attacker unable to influence voting directly might still be able to determine how individuals or groups voted. He might engage in either:

- Vote buying and voter coercion
- Information gathering

In a vote buying attack, the attacker pays (or threatens) individual voters to vote in a specific way. In order for this attack to be successful, the attacker needs to be able to verify how the voter voted. Note that the attacker does not need to be absolutely certain how the voter voted—he must only make the voter believe that the voter's vote has a good chance of being verified.

Another reason an attacker might violate voter secrecy might be to gather information about a large group of people. For instance, an attacker might wish to determine which voters were sympathetic to a particular political party and target them for investigation or surveillance. Alternately, an attacker might wish to publish a prominent voter's votes in order to influence public opinion. In either case, ballot secrecy is required to block this attack.

A.3 Attacker Types

Different kinds of attackers can attack the voting system in different ways. We consider the following broad classes of attacker, 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 — an attacker might have the capabilities of more than one category.

A common focus of security analysis is *privilege escalation*. In many cases, one participant in the system is forbidden from performing actions which can be performed by another participant in the system. A key feature of a secure design is enforcing such restrictions. For instance, a voter should only be allowed to vote once, but poll workers are able to authorize new voters to access the system.

A voter able to use his access to the voting terminal authorize new voters would be an example of someone obtaining privilege escalation. Similarly, poll workers are entrusted with maintaining the integrity of their polling place. If a poll worker were able to use his 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 has no authorized access to any piece of voting equipment. He may be physically separated from the system or present but unable to physically touch the equipment. An outsider has limited capabilities in the context of this review. They might, for instance, 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. [28].

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, either directly or indirectly, an attacker might manage to infect the systems and alter the software running on the machines. If so, any individual with an Internet connection would 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 the night before the election. While polling places may be locked overnight, most polling places are low-security locations. For example, they may be located at a school, a church, a public building, or a citizen's garage. Consequently, an attacker able to break into the polling place can likely obtain unsupervised physical access to the voter equipment for several hours.

An outsider may also be able to impersonate other roles in the system, such as a vendor representative or an election official. For example, an outsider might mail a CD containing a malicious software upgrade to the election official in packaging closely resembling the official packaging from the vendor.

A.3.2 Voters

An attacker able to register to vote would be able to launch voter attacks. Unlike an outsider, a voter has physical access to voting machines for at least a short period of time. Because this access is partially supervised, 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 machines and DREs is relevant here.

Optical scan machines In optical scan voting, the voter marks the ballot, which is a special type of paper, herself. The voter then inserts the ballot into the optical scan machine, typically under the supervision of a poll worker. Therefore, the voter is likely unable to 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 an attack more difficult. The most likely avenue of voter attack is by the ballot itself. For instance, a voter might mark malicious patterns on the ballot intended to subvert the scanner or might attempt to vote multiple times. Such specific patterns could also be used to trigger a dormant "Trojan horse" to cause a compromised machine to begin cheating.

DRE machines In DRE voting, by contrast, the voter has mostly unsupervised access to the voting terminal for a short period of time. The front of the terminal is hidden by a privacy screen in order to protect voters' secrecy, allowing voters to mount a variety of attacks. Any aspect 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 data into the system via the normal user interface. It may be possible to use this interface to compromise the machine.

In all the systems studied here, the voter is 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 smart card and in the Hart system it is a four-digit access code. As the voter has access to these tokens, the tokens are also a potential target of attack as the voter might attempt to substitute a counterfeit token or subvert the smart card.

A.3.3 Poll Workers

Local poll workers have a significant capability that voters do not: legitimate access to the management functions of the equipment. For instance, the poll worker has the ability to authorize voters. Note that although in principle this may give the poll worker opportunities for malfeasance, this risk may be mitigated by procedural controls. For instance, 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. Poll workers must perform their duties in public view, so any 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, officials would investigate the cause of the discrepancy. Purely technical means may, alone, be insufficient to prevent such attacks, but procedural mechanisms may be sufficient to address the 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 instance, some counties provide the chief poll worker at each polling place machines to store and deliver to the polling place. Even counties that deliver machines by commercial transport may provide the chief poll worker with other equipment (smart cards, smart card activation devices, management consoles, etc.) before the election. Even if equipment is stored in a secured polling place, 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.

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 will be able to be a poll worker.

A.3.4 Election Officials

County election officials and staff have three significant capabilities that poll workers do not:

- 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 kinds 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

instance, 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 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. Experience with computer security shows that an attacker who has physical access to a general purpose computer can almost always take control of it. This may be achieved in a number of ways ranging from software attacks to directly compromising the system hardware. 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 mainly by procedural controls and the honesty of 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 opportunities to subvert the system. He might, for instance, insert malicious code into the election software, insert exploitable vulnerabilities or back doors into the system, or 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 or bugs as those are extremely common in large software projects. It may be very difficult to distinguish good-faith mistakes 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 instance, the employee might be present at Election Central during or after the election to help election officials, either by answering questions or by 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 instance, 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. Also, 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, a failure can potentially diminish public confidence. 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.

- *Wholesale vs. retail.* One can distinguish attacks that attempt to tamper with many ballots or affect many voters' votes ("wholesale" attacks) from attacks that attempt to tamper with only a few votes ("retail" attacks). For instance, 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 does not have an impact large enough to change the outcome of an 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.* Certain attacks require little technical knowledge, sophistication, advance planning, resources, or access. For instance, 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.

We can also categorize defenses:

- *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 instance, we attempt to prevent or deter ballot box stuffing by placing the ballot box in the open where it can be observed. 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.

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 [14] and Nedap/Groenendaal [15] 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 simple tools may be able to clone the serial numbers from old wire loops to new ones without detection [31].

Tamper-evident tape: Adhesive tape can be printed with numbered labels in the same fashion as loop seals. 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 [29].

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" [20]. 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 [20].

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.

Bibliography

[1] Interview with Talbot Iredale, Software Development Manager, Diebold Election Systems.

[2] Voting machine company apologizes to Kern County. *KERO 23 Bakersfield*. June 13, 2006. Available at http://www.turnto23.com/news/9366552/detail.html.

[3] ABBOT, R. P., DAVIS, M., EDMONDS, J., FLORER, L., PROEBSTEL, E., PORTER, B., AND STAUFFER, J. Security evaluation of the Diebold voting system, July 2007.

[4] BRENNAN CENTER TASK FORCE ON VOTING SYSTEM SECURITY. *The Machinery of Democracy: Protecting Elections in an Electronic World*, June 2006. Available at http://www.brennancenter.org/dynamic/subpages/download_file_36343.pdf.

[5] BURREN, D. man crypt(3) (FreeBSD 6.2). Available at http://www.freebsd.org/cgi/man.cgi?query=crypt&sektion=3, Jan. 1997. Multiple editors.

[6] CHRISTEY, S., AND MARTIN, R. A. Vulnerability type distributions in CVE, version 1.1. Available at http://cwe.mitre.org/documents/vuln-trends/index.html, May 2007.

[7] CHUNG, L. Microsoft Access or Microsoft SQL Server: What's right in your organization. Available at http://download.microsoft.com/download/5/d/0/5d026b60-e4be-42fc-a250-2d75c49172bc/Access_Whats_Right.doc, Dec. 2004.

[8] DIEBOLD ELECTION SYSTEMS. Frequently asked questions. Available at http://www.diebold.com/dieboldes/faq.asp.

[9] DIEBOLD ELECTION SYSTEMS. AccuVote-OS hardware guide revision 6.0, Feb. 2005.

[10] DIEBOLD ELECTION SYSTEMS. GEMS 1.18 election administrator's guide 8.0, June 2005.

[11] DILL, D., AND WALLACH, D. Stones unturned: Gaps in the investigation of Sarasotas disputed congressional election. Available at http://www.cs.rice.edu/~dwallach/pub/sarasota07.html, Apr. 2007.

[12] DRIEHAUS, B. Audit finds many faults in Cleveland's '06 voting. *The New York Times*. April 20, 2007.

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

[14] FELDMAN, A., HALDERMAN, J. A., AND FELTEN, E. W. Security analysis of the Diebold AccuVote-TS voting machine. In *Proc. 2007 USENIX/ACCURATE Electronic Voting Technology Workshop (EVT '07)*.

[15] GONGGRIJP, R., AND HENGEVELD, W.-J. Studying the Nedap/Groenendaal ES3B voting computer: A computer security perspective. In *Proc. 2007 USENIX/ACCURATE Electronic Voting Technology Workshop (EVT '07)*.

[16] HOGLUND, G., AND BUTLER, J. *Rootkits: Subverting the Windows Kernel*. Addison-Wesley Professional, 2005.

[17] HURSTI, H. Critical security issues with Diebold optical scan design. Available at http://www.blackboxvoting.org/BBVreport.pdf, July 2005.

[18] HURSTI, H. Critical security issues with Diebold TSx (unredacted). Available at http://www.wheresthepaper.org/reports/BBVreportIIunredacted.pdf, May 2006.

[19] HURSTI, H. Diebold TSx evaluation: Supplemental report, additional observations (unredacted). Available at http://www.wheresthepaper.org/BBVreportIIsupplementUnredacted.pdf, May 2006.

[20] JOHNSTON, R. G. Tamper-indicating seals. *American Scientist 94* (November-December 2006), 515–523. Reprint available at http://ephemer.al.cl.cam.ac.uk/~rja14/johnson/newpapers/American%20Scientist%20(2006).pdf.

[21] JOHNSTON, R. G. Tamper-indicating seals. *American Scientist 94* (Nov. 2006), 515–523.

[22] JONES, D. W. Voting system transparency and security: The need for standard models, September 2004. Testimony before the EAC Technical Guidelines Development Committee, National Institute of Standards and Technology, Gaithersburg, MD.

[23] KIAYIAS, A., MICHEL, L., RUSSELL, A., AND SHVARTSMAN, A. Security assessment of the Diebold optical scan voting terminal. Available at http://voter.engr.uconn.edu/voter/OS-Report_files/uconn-report-os.pdf, Oct. 2006.

[24] KIAYIAS, A., MICHEL, L., RUSSELL, A., AND SHVARTSMAN, A. Integrity vulnerabilities in the Diebold TSX voting terminal. Available at http://voter.engr.uconn.edu/voter/OS-TSX-Report_files/TSX_Voting_Terminal_Report.pdf, July 2007.

[25] KLEIN, D. V. Foiling the cracker: A survey of, and improvements to, password security. In *Proc. 1990 USENIX Workshop on Security*.

[26] KOHNO, T., STUBBLEFIELD, A., RUBIN, A., AND WALLACH, D. Analysis of an electronic voting system. In *Proc. 2004 IEEE Symposium on Security and Privacy*, pp. 27–42.

[27] PELZL, J. Cryptanalysis with a cost-optimized FPGA cluster (presentation slides). UCLA IPAM Workshop IV: Special purpose hardware for cryptography: Attacks and Applications. Available at https://www.ipam.ucla.edu/publications/scws4/scws4_6560.pdf, Dec. 2006.

[28] ROHDE, D. On new voting machine, the same old fraud. *The New York Times*. April 27, 2004. Available at http://www.nytimes.com/2004/04/27/international/asia/27indi.html.

[29] RUBIN, A. D. My day at the polls — Maryland primary '06. Available at http://avi-rubin.blogspot.com/2006/09/my-day-at-polls-maryland-primary-06.html, Sept. 2006.

[30] RYAN, T. P., AND HOKE, C. GEMS tabulation database design issues in relation to voting systems certification standards. In *Proc. 2007 USENIX/ACCURATE Electronic Voting Technology Workshop (EVT'07)*.

[31] SHAMOS, M. Oral testimony, Technical Guidelines Development Committee (TGDC), public data gathering hearings. Available at http://vote.nist.gov/PublicHearings/9-20-94%20Panel%202%20SHAMOS.doc, Sept. 2004.

[32] SUN MICROSYSTEMS. Java card technologies. Available at http://java.sun.com/products/javacard/.

[33] WAGNER, D., JEFFERSON, D., AND BISHOP, M. Security analysis of the Diebold AccuBasic interpreter. Available at http://www.ss.ca.gov/elections/voting_systems/security_analysis_of_the_diebold_accubasic_interpreter.pdf, Feb. 2006.

[34] YOUNG, E. des(3). Available at http://www.openssl.org/docs/crypto/des.html.