Part II

Analysis of the Election Systems & Software, Inc. Voting Systems

CHAPTER 4

ES&S EXECUTIVE SUMMARY

This part of the EVEREST report evaluates the ability of the ES&S Unity EMS, iVotronic DRE, and M100/M650-based optical scan voting systems to conduct trustworthy elections. The review team was provided access to the ES&S source code and election equipment. The reviewers studied these materials in order to identify security issues that might be exploited to affect an election. As part of that analysis, the reviewers were asked to identify, where possible, practices that limit or neutralize the impact of discovered issues.

Our analysis suggests that the ES&S Unity EMS, iVotronic DRE and M100 optical scan systems lack the fundamental technical controls necessary to guarantee a trustworthy election under operational conditions. Exploitable vulnerabilities allow even persons with limited access – voters and precinct poll workers – to compromise voting machines and precinct results, and, in some cases, to inject and spread software viruses into the central election management system. Such compromises render the election result subject to subtle manipulations – potentially across election cycles. These vulnerabilities arise from several pervasive, critical failures of the ES&S system:

- *Failure to protect election data and software* – The firmware and configuration of the ES&S precinct hardware can be easily tampered with in the field. Virtually every piece of critical data at a precinct – including precinct vote tallies, equipment configuration and equipment firmware – can be compromised through exposed interfaces, without knowledge of passwords and without the use of any specialized proprietary hardware.

- *Failure to effectively control access to election operations* – Access to administrative and voter functions are protected with ineffective security mechanisms. For example undocumented "quality assurance" hardware tokens that bypass password checks are easily forged using inexpensive commodity devices such as palmtop computers. Central back-end election management software is vulnerable to attacks that exploit coding and design errors and that can be triggered from data sent from the field.

- *Failure to correctly implement security mechanisms* – Many of the most serious vulnerabilities in the ES&S system arise from the incorrect use of security technologies such as cryptography. This effectively neutralizes several basic security features, exposing the system and its data to misuse or manipulation.

- *Failure to follow standard software and security engineering practices* – A root cause of the security and reliability issues present in the system is the visible lack of sound software and security engineering practices. Examples of poor or unsafe coding practices, unclear or undefined security goals, technology misuse, and poor maintenance are pervasive. This general lack of quality leads to a buggy, unstable, and exploitable system.

We believe the issues reported in this study represent practical threats to ES&S-based elections as they are conducted in Ohio. It may in some cases be possible to construct procedural safeguards that partially mitigate some of the individual vulnerabilities reported here. However, taken as a whole, the security failures in the ES&S system are of a magnitude and depth that, absent a substantial re-engineering of the software itself, renders procedural changes alone unlikely to meaningfully improve security.

Nevertheless, we attempted to identify practical procedural safeguards that might substantially increase the security of the ES&S system in practice. We regret that we ultimately failed to find any such procedures that we could recommend with any degree of confidence.

A particular challenge in securing the iVotronic DRE terminal is the large number of precinct-based attack vectors whose exploitation must be prevented. Effective procedures that accomplish this, even if they existed, would be arduous indeed, and would likely substantially hamper poll workers in their duties, reduce the ability to serve voters efficiently, and greatly increase the logistical challenges of running an election.

The security failings of the ES&S system are severe and pervasive. There are exploitable weaknesses in virtually every election device and software module, and we found practical attacks that can be mounted by almost any participant in an election. For this reason, the team feels strongly that any prudent approach to securing ES&S-based elections must include a substantial re-engineering of the software and firmware architecture to make it "secure by design."

It may be possible to deploy a reduced subset of the ES&S hardware and software that excludes components that present the greatest risks. For example, a system that uses only centrally-counted optical scan hardware eliminates many of the threats of precinct-based attacks. We defer to the expertise of the Ohio election officials to determine whether it is possible to use a version of the ES&S system with reduced functionality in a way that presents an acceptable level of risk to the integrity of their elections.

CHAPTER 5

ES&S SYSTEM OVERVIEW

The following chapters detail the ES&S portion of the EVEREST review. Readers are directed to Part I of this report for a discussion of the review's goals and methodologies. Those readers not experienced in information security or Ohio election practices are also encouraged to read the threat model in Chapter 3. The content in that chapter is instrumental in gaining a detailed understanding of the substance and impact of the issues identified throughout.

The ES&S source code analysis and penetration testing exercises reported here were conducted in Philadelphia, PA by the University of Pennsylvania team and in Santa Barbara, CA by the WebWise Security, Inc. team. Team personnel are identified as authors in the front matter of this report. We frequently consulted with the project's procedural and document consultants and with the Pennsylvania State University team.

The issues and commentary described in this section of the report are *reflective of our analysis of the ES&S system only*. None of the included material should be considered to apply to either the Hart or Premier systems, nor should any statement be construed to be making any quantitative or qualitative comparisons of the different systems. Such comparisons are expressly outside the scope of the review, and we have not developed any opinions on the relative merits of any one system with respect to the others. Nothing in this review should be considered as a positive or negative commentary on electronic voting in general.

The remainder of this part of this report is structured as follows. We begin in the next section by giving a brief overview of the ES&S system and its use in Ohio elections. Chapter 6 broadly characterizes the security and reliability of the ES&S system by highlighting the most serious architectural and systemic issues encountered in the review. Chapter 7 catalogs various specific software vulnerabilities that we discovered. Chapter 8 outlines software quality and software engineering risks arising from the system's design and implementation. Chapter 9 describes a number of attack scenarios that demonstrate how the identified issues may be used in concert to subvert an election.

5.1 Architectural Overview

The election management system from Election Systems & Software (ES&S) prepares, collects, and tabulates elections using either paper ballots or touchscreen terminals (or a combination of both). It contains software for managing all stages of an election, as well as special hardware interfaces for configuring and retrieving results from the election devices. The high level architecture of the ES&S system is shown in Figure 5.1.

Unity is the election management software suite. It is comprised of a number of individual programs which interact with one another through shared data files (collectively referred to as the "*database*"). **Election Data Manager** is used to initialize the database with jurisdiction, voter, and candidate information. **ES&S Image Manager** and **iVotronic Image Manager** are used to design the appearance of paper and touchscreen ballots. The **Election Reporting Manager** is used to collect and tally election results, and

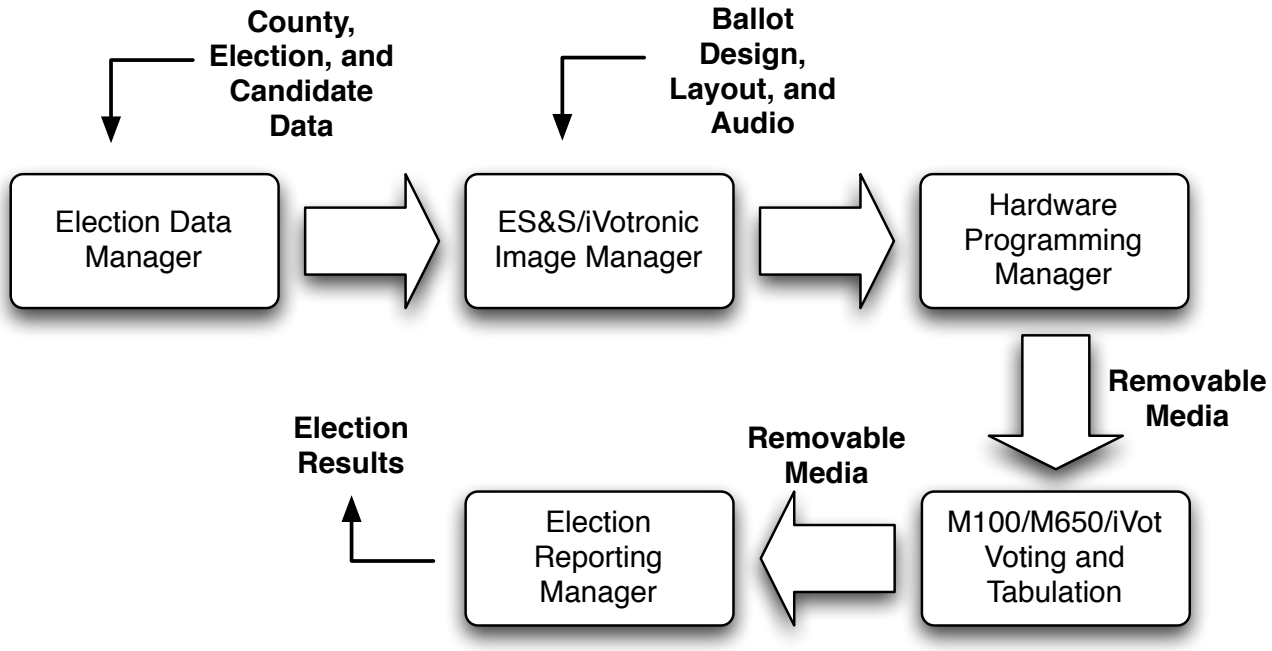

Figure 5.1: The high level overview of the ES&S Unity voting system with user input to each stage of the election.

Audit Manager is used to verify election results using audit data. All interaction between Unity and voting hardware is controlled by the **Hardware Programming Manager** program.

The **iVotronic** is a touchscreen direct recording electronic voting terminal (DRE). There are two distinct types of iVotronic terminals, distinguished by colored inserts along the sides: red *supervisor terminals* and blue *voter terminals*. Both types of iVotronic terminals are activated using special hardware tokens called **Personalized Electronic Ballots (PEBs)**, which are also used to store ballot definitions and election results. PEBs are typically programmed via a supervisor terminal at the start of an election, and read using either a supervisor terminal or a dedicated **PEB Reader** connected to the machine running the Election Reporting Manager at the end of an election. A PEB can be used in multiple iVotronics as long as they are qualified for the same election and polling place.

The voter iVotronics also use **Compact Flash** cards to store large ballots, audio ballots, and election result audit files. Connected to the iVotronic in Ohio is a printer which provides a voter-verified paper audit trail, known as the **Real-Time Audit Log** printer in ES&S documentation and source code. A separate **Communication Pack** is connected to iVotronic terminals at the start and end of elections to print zero count tallies and precinct results on a separate printer with removable paper.

The **Model 100** is a machine for scanning and validating/tallying paper ballots at a polling location. The Model 100 uses **PCMCIA Memory Cards** to hold ballot definitions and tallies.

The **Model 650** is a machine for batch scanning and tallying paper ballots at a central election office. The Model 650 uses **Zip Disks** to hold ballot definitions and election tallies.

The **AutoMARK** Voter Assist Terminal is used by disabled voters to fill out a paper ballot without additional assistance. The AutoMARK uses Compact Flash cards to load ballot definitions.

5.2 System Components

In this section we describe the different components of the ES&S voting system in greater detail. This background on the architecture and implementation will aid in understanding the vulnerabilities and attacks in subsequent chapters.

The ES&S system is made up of a number of custom and off the shelf general purpose computers. On top of these computers run several hundred thousand lines of source code in various pieces of software. It is important to keep in mind that, although they do not appear the same as your typical desktop or laptop computer, all the components of the ES&S system are fully programmable computers capable of running arbitrary software stored in easily modifiable memory. Therefore use of the term "firmware" to refer to the software controlling the hardware components of the ES&S system is somewhat misleading. The code running on the iVotronic or Model 100 is in no way less susceptible to bugs, tampering, or co-option than any other part of the Unity system. When discussing the code comprising the ES&S system we will use the term "software" to refer to code running both on desktop PCs and the specialized voting hardware.

5.2.1 Unity

Unity is the Windows-based software suite for managing elections. It contains tools for creating and managing election databases (Election Data Manager), designing the appearance of ballots (ES&S and iVotronic Image Managers, AutoMARK Information Management System), tabulating and reporting results (Election Reporting Manager). Additionally, there is a tool to audit the use of the other components of Unity (Audit Manager), and a tool for abstracting programming and communicating with the various hardware components used by several Unity components (Hardware Programming Manager). The various components of Unity communicate with each other indirectly through common files stored on the Windows filesystem.

Election Data Manager

The Unity Election Data Manager (EDM) is a Windows XP application which is used for creating and updating the election database used by the other software components of Unity. The database for each county is stored in a single file in the Windows filesystem known as the "Ballot Data File" (BDF).

Each BDF is logically made up of two different databases: the "County Database" which contains tables of data which does not change from election to election, and the "Election Database" which contains the data defining a single election. Reusing the County Database from election to election is encouraged through an import feature which copies from a prior election's BDF to a new one, and the initial County Database can be filled in from another county's BDF as well. The remaining data entry is done either through the Windows GUI, or by importing text files in a number of formats.

In addition to specifying the races, candidates, proposals and other data that will appear on the ballot for an election, EDM is also used to select the equipment which will be used to cast votes in the election and specify any policies particular to the jurisdiction which influence ballot design (e.g. candidate position rotation). Finally, centralized configuration of the iVotronic passwords (see below) is performed from within EDM.

Other modules of Unity read and parse the BDF directly, as well as using "Intermediate Interface Files" (IFF) and "Ballot Set Collection" (BSC) files produced by EDM in a process referred to as "merging the election database". While the ballot data file is updated automatically when any changes are made in EDM, the other files must be re-generated by merging the election again.

ES&S Image Manager

ES&S Image Manager (ESSIM) is a C++ Windows page-layout tool used to design optical scan paper ballots. It is also sometimes referred to by an older name, Ballot Image Manager (BIM), in documentation and source code. ESSIM uses the IFF and BSC files exported from EDM to obtain the information to be displayed on ballots, including both information for the voter (candidates, races, proposals, etc.) and information for the back-end optical scanners (ballot codes, etc.). The layout specific data is stored in a "Ballot Layout File". When ballot design is finalized, ESSIM exports another IFF file (with a different file extension) to be used in the Hardware Programming Manager, as well as PDF ballots to be sent to a printing company.

iVotronic Image Manager

The iVotronic Image Manager (iVIM) is a Java Windows application for designing text, graphical, and audio ballots for the iVotronic DRM. iVIM relies on a MySQL database server to store its settings, which can either be installed on the same machine as iVIM or accessed remotely. The main input to iVIM is an XML file exported by EDM, along with graphical templates bundled with iVIM. Once layout is complete, iVIM exports the ballots as an iVotronic Election Definition file and a folder containing bitmap images and file hierarchy for the iVotronic Compact Flash card. Audio ballots for ADA voting are not managed within iVIM. Instead, an HTML document listing the necessary recordings and filenames to save them as is produced along with the other output files.

Hardware Programming Manager

Hardware Programming Manager (HPM) is used to transfer ballot configurations from files produced by ESSIM and iVIM to the removable media used in the M100, M650, and iVotronic. It consists of a Windows application written in COBOL and a number of helper applications and shared libraries (DLLs) written in C and C++. The COBOL code focuses on overall program logic and the creation of a new election database, while the C and C++ code focuses on communicating with the PEB and memory card readers and performing specialized operations (e.g. cryptography). Additionally, HPM serves as the bridge between the Election Data Manager (where elections are defined) and the Election Reporting Manager (where elections are tallied and reported), since the latter is also a COBOL based application.

The election configuration in HPM is imported from the IFC file produced by ESSIM. Adjustments to the election database can be made within HPM prior to writing the removable media to be used in the iVotronic or ballot scanner. These changes are only seen within HPM and the Election Reporting Manager, and not reflected in EDM. If changes are made in EDM they must be propagated through ESSIM to HPM and be imported again. Finally, device-specific settings for the M100 and M650 optical scanners can also be made from within HPM.

HPM copies files from ESSIM directly to the PCMCIA SRAM cards and Zip disks for the M100 and M650 scanners. For the iVotronic, the Compact Flash folder produced by iVIM is copied using a helper application. Before the PEB can be programmed, a binary ballot file is created using another helper application. This file can either be encrypted or unencrypted depending on options set by the user. The PEB is then programmed using another command-line tool which copies this binary ballot file along with the EQC block to a unused Supervisor PEB (see Section 5.2.4). A red Supervisor iVotronic is connected to the computer running HPM over a serial port and used to communicate with the PEB for this programming.

Election Reporting Manager

The Election Reporting Manager (ERM) is used to collect and report results from an election. Like HPM, it is a COBOL Windows program which relies on several helper applications and libraries written in other programming languages to access the removable media containing votes. It uses the same directory of data files generated by HPM upon importing an IFC file, and creates a new election database file used to store results.

Election results are collected from M100 and M650 scanners directly using an external PCMCIA card reader and Zip drive. iVotronic results can be collected either from the Compact Flash card of each machine, or from the Master PEB used to close several iVotronics. The CF card is read directly using a USB card reader, while the Master PEB can be read using either a Supervisor iVotronic or PEB Reader connected to the serial port of the machine running ERM using the same command-line tool used by HPM.

Once retrieved from memory devices, the COBOL code processes the results files and records the official election results. ERM has features for producing election reports per contest, per precinct or summary results. Additional details that can be reported include the totals for each type of ES&S tabulation device.

Audit Manager

The Election Data Manager and ES&S Image Manager (as well as Audit Manager itself) contain a mechanism for logging use and modification of the Unity election database files to a Microsoft Access database. Audit Manager provides a way of viewing these logs.

Other components of Unity developed in languages other than Microsoft Access (iVIM, HPM, ERM) have no support for recording actions in the Audit Manager log. Additionally, Audit Manager relies on EDM and ESSIM to dutifully update the Access database; Audit Manager is incapable of detecting changes to critical files performed outside of Unity tools.

AutoMARK Information Management System

The AutoMARK Information Management System (AIMS) is a standalone Windows application used to configure the AutoMARK Voter Assist Terminal to read and mark optical scan ballots produced in Unity with EDM and ESSIM. Although bundled as part of the Unity suite, the AutoMARK and AIMS were not developed by ES&S, and are not tightly coupled with any of the other Unity programs or files.

AIMS contains an entire election database of its own, which can either be imported from EDM's BDF or re-entered using the AIMS interface. Documentation recommends the later approach be taken to avoid errors in the conversion. Once the election database has been created, visual, audio, and translated ballots are designed for each type of printed ballot. The final configuration is copied to a Compact Flash card to be inserted in the AutoMARK device.

5.2.2 iVotronic

The iVotronic is a touchscreen DRE based around an Intel 386 processor with 1 MB of SRAM. There are four internal flash memory devices. One of these holds the iVotronic firmware, and is memory-mapped directly into the address space of the CPU to avoid using the limited amount of RAM to hold instruction code. The other three flash devices provide redundant storage for ballot and vote data. At power-up, and periodically during operation, the contents of the three memories are compared to one another to detect any corruption.

The iVotronic does not run anything resembling a modern operating system. There is a single process started at boot which runs in an event (interrupt) driven loop. Multiple threads, memory protection, dynamic memory allocation, exception handlers and similar constructs are unavailable.

The primary user input device to the iVotronic is an LCD touchscreen which is wired to the CPU as a serial device. There are two other serial ports on the iVotronic, one connected to a standard DB9 serial port at the top of the iVotronic, and the other to an infrared transceiver. The external serial port is used to connect to the RTAL printer (see Section 5.2.3) or a Communications Pack (see Section 5.2.7) in the field, and to a computer running Unity HPM or ERM in the central election office.

The infrared serial port is used to communicate with the Personalized Electronic Ballot hardware tokens (see Section 5.2.4). The left side of the iVotronic case has a molded socket to hold a PEB allowing IR communication as well as activation of the iVotronic power switch through a magnetic reed switch. There are two IR transceivers, one on the inside of the PEB socket, and a second on the outer edge of the iVotronic case. These two transceivers are multiplexed onto the single serial port, and only one can be used at a time.

In addition to the internal solid-state flash memory, a Compact Flash slot is located next to the printer serial port. The inserted CF card is accessed using a third-party library which implements the necessary filesystem code.

There are two types of iVotronic terminals used in elections: red "Supervisor iVotronics" are used by poll workers or elections officials to administer the election, and blue "Voter iVotronics" are used by voters to cast their votes. Every iVotronic has a serial number which is programmed into a PIC microcontroller during manufacturing. Except for this serial number and the outward appearance, all iVotronics are functionally identical. The same firmware is used on both Supervisor and Voter iVotronics, with a serial number check at boot to determine which mode is used. In the next two subsections the unique aspects of each type are introduced.

iVotronic Supervisor Terminal

The red Supervisor Terminal is used to manage PEBs and the contents of their flash memory before, during and after elections. It plays a far more significant role in the Voter Activated Voting mode (which is not used in Ohio elections), where at least one Supervisor Terminal must be at every polling place.

In the Poll Worker Activated Voting mode used in Ohio, a Master PEB for each polling location is created from HPM using a Supervisor Terminal connected via null modem cable. Afterwards, each Master PEB is then cloned several times using a Supervisor Terminal (standalone from Unity) in order to produce the Supervisor PEBs needed while the polls are open (see Section 5.3). At this point, the supervisor terminal is no longer required for opening, closing, or tallying the election.

iVotronic Voter Terminal

The blue Voter Terminal is the iVotronic used by voters to cast their ballot. When a qualified Supervisor PEB is inserted the iVotronic prompts the poll worker to select the correct ballot to be voted on. The poll worker then removes the PEB and leaves the voter to make their decisions. Once the voter casts their ballot, the Voter Terminal goes into a low-power state and waits for the Supervisor PEB to be inserted again to cast another ballot.

If the poll worker inserts the Supervisor PEB while holding the "Vote" button above the touchscreen, a service menu appears. This menu allows various settings of the terminal to be adjusted, and also provides the interface for opening and closing the polls. While in the service menu, actions performed are logged to the RTAL printer.

5.2.3 Real-Time Audit Log Printer

The Real-Time Audit Log Printer (RTAL) is a continuous feed thermal printer manufactured by FutureLogic, Inc. specifically for use in DRE voting systems. It performs the function of VVPAT on iVotronic machines.

It is connected to the iVotronic by a standard 9-pin RS232 serial cable, and mounted behind a plexiglass window next to the iVotronic. The RTAL supports both ASCII text using several built in fonts and encodings and a bitmap mode used to print barcodes or other non-text content.

Unlike the printer in the Communications Pack (see Section 5.2.7), the RTAL collects the output internally. The RTAL only has a motor on the collection spool, making it impossible to rewind the audit log without jamming. A movable carrier holds the paper mechanism independently of the print head and display window, allowing for the appearance of a small backwards movement (approximately .5") and the ability for the print head to "back up" a few lines.

5.2.4 Personalized Electronic Ballot

The Personalized Electronic Ballot (PEB) is a palm-sized device containing a PIC microcontroller, 2MB of flash storage, a bi-directional infrared (IR) transceiver, and battery. The PEB is activated by a magnetic reed switch, and contains a magnet to activate the corresponding reed switch in the iVotronic PEB socket.

In addition to the flash storage, the PIC microcontroller contains a small amount of non-volatile storage which is "burned" to the PIC during the PEB manufacturing process. This area contains the PEB firmware, PEB firmware version number, PEB hardware revision number, PEB serial number, and 'PEB Kind' variable.

The microcontroller firmware implements the passive half of a very simple command/response protocol between the PEB and host over the PEB IR port using IrDA SIR (Serial Infrared). The host sends the PEB one of several commands along with an address and fixed mount of data depending on the command, and the PEB performs the command and returns a response and command-determined amount of data. The primary operation is reading and writing 128 byte blocks of the PEB's flash memory, or verifying the integrity of blocks using a cyclic redundancy check (CRC) stored with each block. In addition, the serial number, PEB kind, battery voltage can all be retrieved but not modified by the host device.

While all PEBs are internally identical in construction, they are discernible from one another by the read-only information burned in the PIC: their serial number, and more importantly by their PEB Kind (both read-only values once the PIC inside is burned). The two documented PEB Kinds are *supervisor* and *voter*, which are visually differentiated by a red and blue band in the casing. In Ohio, only supervisor PEBs are used in the documented election procedures. There is a third (undocumented) PEB Kind recognized in the iVotronic source code.

The first block of a Supervisor PEB further identifies it as having been configured as a regular Supervisor PEB or as a special-purpose Supervisor PEB which can perform a single iVotronic administrative function without needing to go through multiple menus. These special-purpose PEBs are commonly referred to by the name of the function they access (e.g. a "Clear and Test PEB" jumps immediately to the Clear and Test menu of the iVotronic it is inserted in).

In a regular Supervisor PEB, the remaining blocks hold the ballot (identical to the `.bin` or `.ebn` produced in HPM), followed by a fixed size "vote results structure" (the `.vot` file read by ERM), and finally the remainder of the PEB (except for the last block) holds write-in ballots. The last block of the PEB is known as the Election Qualification Code block, and serves to authenticate the PEB to the iVotronic.

5.2.5 PEB Reader

The PEB Reader is a standalone cradle used by devices with standard RS232 serial ports to communicate with PEBs. It acts as a media converter between the RS232 serial connection and the IrDA connection to the PEB. The PEB Reader does not contain any functionality in hardware other than this media conversion; all protocol implementation for communicating with a PEB must be done in software on the device connected

to the PEB Reader. Despite its name, there is nothing preventing a PEB Reader from being used to issue commands to write to or erase a PEB if controlled by suitable software.

5.2.6 Compact Flash Cards

Standard Type I Compact Flash cards are used by Unity and the iVotronic to hold files too large to fit in the PEB flash storage as well as audio ballots and audit and results data. Cards contain a standard FAT16 Windows filesystem and are accessed through a dedicated slot in the iVotronic and an external USB reader on the Unity PC. In Windows, these are mounted to the desktop and accessible to any Windows application with no special libraries.

The ballot data is accessed by the iVotronic on demand, but the presence of the CF card is checked periodically and the iVotronic will not boot without its presence. That same CF card must be present to close the polls. An audit log can be saved to the card when the polls are closed. It is a raw dump of the internal flash memory, compressed and encrypted before saving.

5.2.7 Communication Pack

The Communication Pack is a briefcase containing a thermal printer, a modem (not used in Ohio), space to store PEBs for transport, and a serial cable to connect to an iVotronic. A switch is used to toggle between off, printer, and modem modes, and the case contains batteries for when a wall outlet is unavailable.

The communication pack is used by disconnecting the RTAL (if present) from an iVotronic (supervisor or voter) and connecting the communication pack's serial cable when prompted by the iVotronic. The iVotronic then controls the printer or modem in the communication pack and prompts the user when to disconnect the communications pack and reconnect the RTAL.

5.2.8 Model 100

The Model 100 (M100) is a voter-operated optical scan ballot counter intended for use at the polling location. It is mounted on top of a secure ballot box which holds the accepted (and counted) ballots and provides physical security for the M100. The M100 we were provided with used one set of identically keyed locks for physical protection of the M100 and ballots, and a second differently keyed lock for selecting the mode of the M100.

The M100 contains an Intel 286 microprocessor with 2MB of RAM, and runs the QNX embedded operating system from an internal 512KB flash memory. A thermal receipt printer and 40x4 character LCD text display are built-in, and a standard parallel printer port is available for connecting an external printer (when a parallel port connection is detected the M100 automatically switches all printed output to use it instead of the built-in thermal printer). In addition to the optical scanner, the inputs to the M100 are 4 programmable buttons positioned below the LCD display for user input, an RS232 serial port, and two PCMCIA Type slots for loading ballot images and firmware updates, and storing counted ballots.

The M100 performs a self test of its operating system on power-up, and checks for a properly formatted flash card in the PCMCIA slot. The PCMCIA card is mapped directly into memory, and portions of it are copied into RAM for faster access. For the remainder of the data on the PCMCIA card, the M100 builds an index in memory to data structures on the card. The card is written to after each ballot is scanned, recording either the votes cast for a valid ballot or the error for an invalid ballot.

There are two modes for the M100, selected using a key: Open/Close Polls and Vote. When set to Open/Close, a number of administrative and diagnostic options are available which print summaries of the contents of the PCMCIA card. Once done, a poll worker selects the "open polls" or "reopen polls" menu option, and turns the key to Vote when prompted. In Vote mode, the M100 operates in an automated fashion,

only needing user interaction on invalid ballots. At any point the M100 can be switched back to Open/Close mode using the key.

5.2.9 PCMCIA Memory Cards

Unity (via the HPM) and the M100 use specially formatted PCMCIA SRAM flash storage cards for all communications. The cards are formatted so that a small header can be loaded into the M100's RAM which contains pointers into the SRAM for ballot definitions and results counters.

The PCMCIA memory card is also used to upgrade the firmware of a M100. When a correctly formatted card is present at power-up, the M100 will prompt the user to copy a new firmware image to the internal flash memory.

Unlike the Compact Flash cards and Zip disks used by other equipment, the PCMCIA card does not contain a recognizable filesystem and cannot be accessed directly through Windows. Instead, a library for the OmniDrive USB reader is required to read or write data.

5.2.10 Model 650

The Model 650 (M650) is a centralized high-speed optical ballot counter intended for use at a central elections office. It scans batches of ballots, possibly from multiple precincts, and tabulates results to be transferred to Unity.

The M650 contains an Intel Pentium processor and runs the QNX real-time operating system off of an internal solid-state hard drive. There are two standard parallel printer ports which must be connected to printers for operation to begin. Additionally, an Iomega Zip 100 drive is used to transfer ballot definitions and perform firmware updates to the M650, and carry results from the M650 to Unity. The M650 has a control panel on its front with buttons to start and stop scanning as well as set the mode of operation. An LED display provides feedback to the operator.

Inside the chassis, the motherboard and daughter cards are accessible behind a locked panel. There are two RS232 serial ports and a PS2 keyboard/mouse port located on daughter cards which are not accessible from outside the locked chassis.

5.2.11 Zip Disks

The M650 uses FAT32 formatted 100MB Zip disks to load ballot configurations and store tallies of counted ballots. The files on the disk are copied by the Hardware Programming Manager. In Windows, these disks are mounted to the desktop and accessible to any Windows application with no special libraries.

5.2.12 AutoMARK Voter Assist Terminal

The AutoMARK Voter Assist Terminal (VAT) is a combination scanner/printer used by the voter to mark and verify optical scan ballots (to be tabulated by the M100 or M650). It is intended to be used by disabled voters as required by the Help America Vote Act. The VAT runs Windows CE (now called Windows Mobile) with a custom application developed in .Net for the interface.

The VAT is operated by inserting a paper ballot which is then scanned and matched to a pre-defined ballot style configured during election setup. Translations into various languages as well as audio ballots may be configured for a particular ballot style which the user then uses to complete or verify their ballot. Once the user confirms their choices, an inkjet print head contained within the VAT marks the inserted ballot (if it was initially blank) and returns it to the voter. There is a single Compact Flash slot used to program ballot styles, and several jacks for audio headsets and input devices. The AutoMARK does not store or tabulate votes itself, and no data is read from the AutoMARK into Unity.

5.3 Election Procedures

This section describes how the ES&S components discussed in this overview are used in an actual election. We draw our information from the ES&S documentation, and assume that elections follow these general procedures, though some counties may differ on specific procedures.

5.3.1 Preparation

Preparation for an election begins by setting up complete county and election databases using the EDM. This is an involved and complicated process requiring an expert user. The EDM stores all county, precinct, jurisdiction, ballot styles, contests, registered voter totals, and whatever other information is necessary for the election. The set up of this database is so complicated that many counties rely on a service from ES&S to create the database for them. In addition to the election database, ballots must be designed using either ESSIM for paper ballots or the iVIM for iVotronic electronic ballots. Per-election passwords for iVotronic DREs are also set at this point.

Once the election database has been created, and ballot images prepared, an election administrator uses the HPM to program all the media required by the different ES&S voting hardware.

5.3.2 Touchscreen (DRE) Voting

The election process for DRE voting requires the HPM, a Supervisor iVotronic terminal, Compact Flash cards, and PEBs.

Preparation

At each new election an Election Qualification Trail must be started. The Supervisor terminal displays the Election Qualification Code (EQC) and then permits the qualification of PEBs to be used in the DREs. Each qualified PEB is programmed with the EQC and new system passwords (if changed) are ready to be loaded with the ballot for the current election. A PEB is programmed using a supervisor terminal which is connected through a serial port to a machine running the HPM. One PEB for each precinct is chosen as the master PEB, the others are referred to as supervisor PEBs.

A Compact Flash card (one for each iVotronic used in the election) must be programmed. CF cards are programmed through the HPM using a standard commodity CF card reader/writer.

At some point, after the PEBs and CF cards have been programmed, the blue voter terminals must be cleared and tested using a correctly programmed supervisor PEB. This erases the votes and audit data from the previous election, and loads the EQC and any custom passwords. The Election Test menu options are only available after a terminal has been cleared and tested.

The iVotronic allows the following election tests:

- Logic and Accuracy - Tests whether votes are recorded accurately.
- Automated Multi Vote Test - tests multivote and write-in option.
- Vote Selected Ballot Test - for multiple ballot images
- Manual Vote Test

Election Day

On election day poll workers unpack and set up the blue iVotronic terminals, plug in the RTAL printer and power cables. A properly programmed CF card must already be installed in the terminal, (this is usually done at Election Central and a tamper-evident security seal usually placed over the CF slot), before powering the terminal on. A master PEB is required to open each terminal for voting, and only that same master PEB can be used to close the terminal after the polls have closed. At this time a zero tape showing no votes cast may be printed, but this is optional.

For each voter, a supervisor PEB containing the ballot images must be inserted into the iVotronic. Insertion of the PEB turns on the iVotronic, checks the EQC, and initializes (loads in) the ballot. The poll worker removes the supervisor PEB, the voter votes, the RTAL printer prints the results and the electronic ballot is stored internally in the iVotronic until the terminal is closed.

Vote Tallying

To close a terminal, the master PEB is inserted, which collects and stores the tabulated data, copies of the "images" of the ballots cast and time and date information. At closing the iVotronic firmware automatically uploads Audit Data onto the CF card. At this time a results tape can be printed using a special external printer.

The results tape, zero tapes, Compact Flash cards, and Master PEB are then returned to Election Central. At Election Central the results can be imported into ERM using either the Master PEBs or the Audit Image on the CF cards.

5.3.3 Precinct-Based Optical Scan Voting

The M100 is an optical scan machine used to process individual paper ballots. It can be operated by the voter or by a poll worker. The election process for an M100 machine requires the HPM, a commodity PCMCIA reader/writer and two physical keys, the scanner key and the ballot box key.

Preparation

The M100 reads the proper election definition from a prepared PCMCIA card. This card is programmed at Election Central using the HPM and a locally connected PCMCIA reader/writer. The PCMCIA card can then be inserted into the M100 and the slot secured with a tamper-evident seal.

Election Day

A poll worker unpacks and sets up the M100. Usual procedures include using the ballot box key to open the emergency bin and compartment side doors to verify that the bins are empty. The compartments are then re-locked and the key secured. The M100 requires a key (scanner key) to power on. A poll worker inserts this key into the power on switch, and turns the switch to the Open/Close Polls position. The scanner loads the election definition from the PCMCIA card into its operating system. The scanner will display "S-MODE" in the upper left corner of the LCD screen and the message "ELECTION CARD INSERTED OPEN POLLS NOW?"

The poll worker then turns the scanner key to the VOTE position. After initializing, the scanner automatically prints an Initial State Report plus any other reports it was programmed to print. This may include a report showing no votes on the scanner for each of the races and/or questions as well as a certification message. The poll worker then secures the scanner key.

After a voter has marked their ballot, the ballot is scanned by inserting it into the ballot entry slot in any direction. The ballot count on the display increases whenever the scanner successfully scans a ballot. If there are no issues with the ballot such as over or under voting the operator can press accept to have the ballot deposited into the ballot box. If there are concerns with the ballot (such as over or under voting) there is an option to press reject and retrieve the ballot. The ballot will then be spoiled and the voter is usually issued a new ballot and directed to a voting booth.

Vote Tallying

The M100 keeps a running tally of votes internally and on the PCMCIA card.

To close the polls the scanner key is inserted and turned to the OPEN/CLOSE POLL position. The scanner will automatically print reports that may include a Status report, Poll or Precinct report, Certification report, and/or an Audit Log report. The POLLS CLOSED menu will appear after printing and results is complete.

The PCMCIA card is removed, the ballot boxes removed from the cabinet and the boxes, cards and the printed reports are returned to Election Central.

5.3.4 Centrally Counted Optical Scan Voting

The M650 is an optical scan machine for processing batches of ballots. It is usually kept at Election Central. The election process for an M650 requires the HPM and a commodity Zip Disk reader/writer.

Preparation

The M650 reads the proper election definition and firmware from a Zip disk. The Zip disk with the election definition is placed in the Zip drive in the M650 and the scanner is powered on. This message will appear once the scanner has initialized: “Press Stop to Keep XXXXX Press Start to Load YYYYY (Zip)”, where XXXXX is the name of the election currently in the machine, and YYYYY is the name of the election on the Zip disk.

If start is pressed, the M650 displays a menu option for zeroing out previous election totals (this is optional) and prints out a readiness report. The machine is now ready for ballot tabulation.

Election Day Vote Tallying

Ballots are transported to Election Central from the various precincts in secured transfer cases. On arrival the cases are opened and an election worker processes the ballots into the M650. To tally ballots an election worker places them in batches into the M650 input hopper and presses start. Counted ballots are stored in an output hopper and must be removed by hand before the next batch of ballots is tabulated. The result totals are stored internally and the election worker needs to periodically press the Save button on the M650 to store the results to the Zip disk for transfer to the Unity ERM. The manual recommends saving and transferring the results on the Zip disk to the machine running the ERM every time the hopper is emptied. It is possible to network the M650 and to have the results transferred by network instead of by Zip disk.

5.4 Software Versions

The source code analysis and red teams were provided with the following versions of the Unity environment and source code by the vendor:

Component	Application Version
Unity	3.0.1.1
Audit Manager	7.3.0.0
Election Definition Manager	7.4.4.0
Election Reporting Manager	7.1.2.1
Hardware Program Manager	5.2.4.0
Data Acquisition Manager	6.0.0.0
ESS Image Manager	7.4.2.0
iVotronic Image Manager	2.0.1.0
iVotronic Firmware	9.1.6.2 9.1.6.4
M100 Firmware	5.2.1.0
M650 Firmware	2.1.0.0
RMCOBOL RT	7.5.01
COBOL WOW RT	3.12

Source Component	Version		Source Component	Version		Source Component	Version
4 Key Sound Pic	1.0.0.0		Events	9.1.1.0		REGUTIL	1.0.0.0
AuditManager	7.3.0.0		EXITWIN	1.0.0.0		SCNDAT30	3.0.1.0
BXP_Gen	1.0.0.0		GENPARMS	1.3.0.0		Serve650	1.0.0.0
CB_650	1.0.0.0		GetAuditData	9.1.0.0		SHELL	1.0.0.0
CB_EMP	1.0.1.0		Images	9.1.2.0		SHELLSETUP	1.0.0.0
CB_M100	1.2.0.0		Init650	2.1.0.0		TGEN30	3.0.1.0
CB_Duplicator	9.1.0.0		iVOTIM	2.0.1.0		Undrvote	9.1.2.0
CF_Utility	9.1.0.0		iVotronic	9.1.6.2 9.1.6.4		Viodialog	9.1.2.0
CRCDLL	1.4.0.0		Loader ILD	9.0.0.0		VIOWIN	9.0.0.0
EDM	7.4.4.0		M100	5.2.1.0		Volume Control	1.0.0.0
ERM	7.1.2.1		M650 Display	2.1.0.0		WDAM	6.0.0.0
ERSGEN30	4.1.0.0		M650 Firmware	2.1.0.0		WHPM	5.2.4.0
ESSCRYPT1	1.0.0.0		M650 Support Scripts	2.1.0.0			
ESSCRYPT	1.8.0.0		MAKEIBIN	9.1.0.0			
ESSDECPT	1.8.0.0		MPRBOOT	2.6.0.0			
ESSIM	7.4.2.0		MYDLL	1.0.0.0			
ESSM100	1.3.1.0		PCCARD30	3.0.0.0			
ESSPCMIO	1.0.0.0		PEB PIC	1.7c			
ESSUTIL	1.0.0.0		PGENAZ80	1.2.0.0			

5.5 Unity Acronyms and File Extensions

`.ais`	Ballot layout file extension for ESSIM
BDF	Ballot Data File: EDM database file
`.bdf`	BDF file extension
BIM	Ballot Image Manager: see ESSIM
`.bin`	Unencrypted iVotronic Ballot file extension; created by HPM helper application; used by HPM PEB writer helper application
BSC	Ballot Set Collection: created by EDM; used by ESSIM
`.bsc`	BSC file extension
`.ebn`	Encrypted iVotronic Ballot file extension; created by by HPM helper application; used by HPM PEB writer helper application
EDM	Election Data Manager: Unity elections database program
ERM	Election Reporting Manager: Unity tabulation and results program; alternatively called Election Report Manager and Election Results Manager in source code and documentation
ESSIM	ES&S Image Manager (also known as BIM): Unity optical scan ballot page layout program
`.ifc`	IFF file extension for HPM
IFF	Intermediate Interface File: ballot data file; created by EDM; used by ESSIM and HPM
`.iff`	IFF file extension for ESSIM
iVIM	iVotronic Image Manager: Unity touchscreen DRE ballot layout program
PEB	Personalized Electronic Ballot: Hardware token used to program and vote on iVotronic DRM; alternatively called Personal Electronic Ballot and Portable Electronic Ballot in source code and documentation
`.pxt` and `.px1`	Votronic Election Definition file extension; created by iVIM; used by HPM for PEB
RTAL	Real-Time Audit Log (see VVPAT); Also used to refer to iVotronic with RTAL attached
`.vot`	iVotronic Vote Results file extension; created by ERM PEB reader helper application; used by ERM
VVPAT	Voter Verified Paper Audit Trail: Printed record of votes cast on a DRE
`.xml`	Ballot Foundry Election Definition file extension; created by EDM; used by iVIM

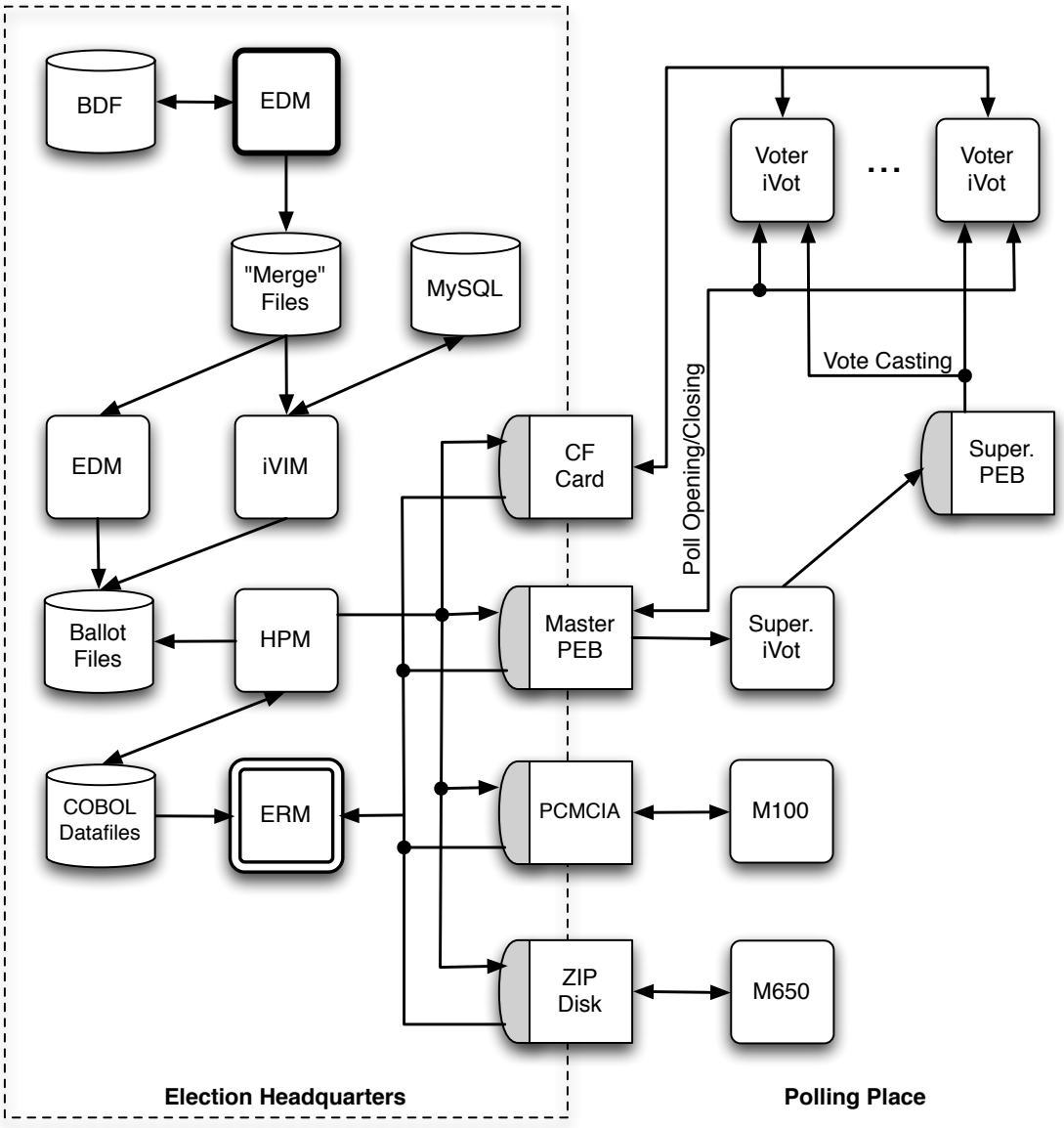

Figure 5.2: The major components and the flow of election data for counties using the ES&S Unity voting system. Arrows indicate direction of data flow. An election begins with configuration using the Election Data Manager (EDM), and ends once results are tallied and reported in the Election Reporting Manager (ERM).

Figure 5.3: A blue Voter iVotronic with a Supervisor PEB inserted

Figure 5.4: A red Supervisor PEB

Figure 5.5: A Compact Flash card used by the Voter iVotronic and AutoMARK

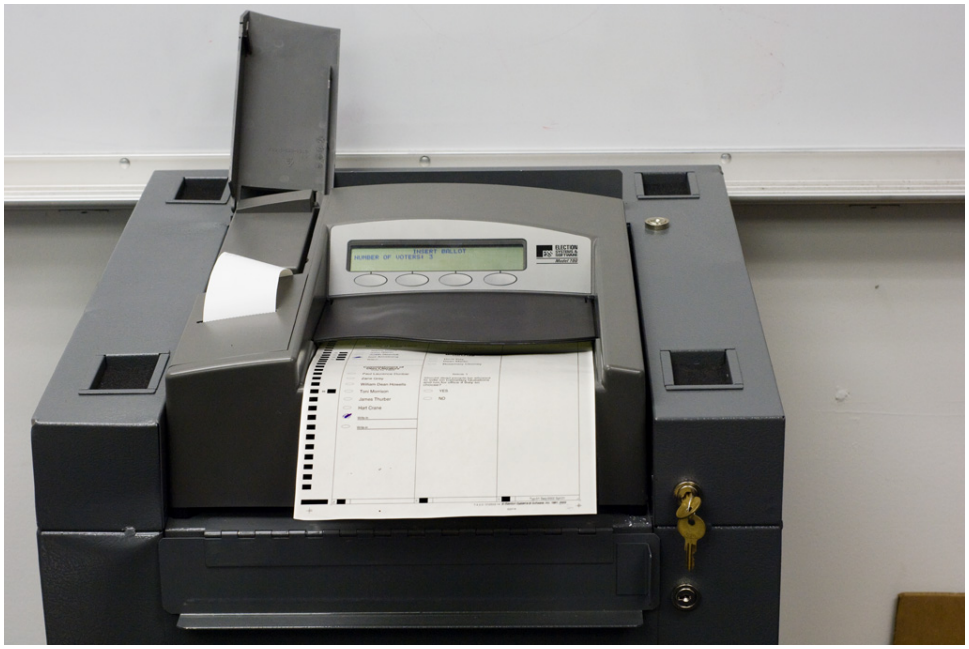

Figure 5.6: The ES&S M100 precinct optical scan ballot counter

Figure 5.7: A PCMCIA SRAM flash card

Figure 5.8: The ES&S M650 centralized optical scan ballot counter

CHAPTER 6

ES&S SYSTEMIC AND ARCHITECTURAL ISSUES

We found fundamental security deficiencies throughout the ES&S Unity EMS, iVotronic DRE and M100 optical scanner software and hardware. Virtually every mechanism for assuring the integrity of precinct results and for protecting the back-end tallying system can be circumvented. Election results can be tampered with in the ES&S system by exploiting any of a number of different vulnerabilities we discovered. The normal access provided to individual precinct poll workers (and in some cases to voters themselves) is sufficient to conduct attacks that alter county-wide election results and that, in some cases, cannot be detected or recovered from through audits or recounts.

Perhaps most seriously, there is a strong potential for practical attacks that propagate "virally" from the field back to the county election management system. That is, a single circumvented piece of precinct hardware (such as a memory card returned from a precinct for vote tallying) can effectively "take over" the county-wide back-end tally system, alter county-wide results reported in the current election, and then corrupt the installed firmware of additional precinct hardware in subsequent elections. The broad scope of such attacks provides great leverage to the adversary and can be extraordinarily difficult to detect, trace, or recover from.

Both the DRE (iVotronic) and the precinct-based optical scan (M100) systems are subject to many exploitable vulnerabilities. The DRE system provides more vectors for attacks that cannot be recovered from through manual recounts, however.

While there are many specific errors and weaknesses in various parts of the ES&S software (and which are detailed in Chapter 7), there are also systemic weaknesses throughout the system's overall design and implementation. These weaknesses render the system as a whole especially difficult to secure in practice. We identify here four fundamental, pervasive deficiencies that give rise to the most serious vulnerabilities we found:

- Ineffective access control
- Critical errors in input processing
- Ineffective protection of firmware and software
- Ineffective cryptography and data authentication

6.1 Ineffective Access Control

The firmware and configuration of the ES&S precinct hardware can be easily tampered with in the field. Virtually every piece of critical data at a precinct – including precinct vote tallies, equipment configuration and

equipment firmware – can be compromised through exposed interfaces, without knowledge of passwords and without the use of any specialized proprietary hardware.

6.1.1 iVotronic passwords and PEB-based access controls

Access to the iVotronic DRE configuration is protected by several hardware and password mechanisms, all of which can be defeated through apparently routine poll worker (and in some cases voter) access.

The primary mechanisms for preparing iVotronic DREs for deployment at precincts and for managing them throughout the election day (e.g., enabling them for each voter) employ the *Personalized Electronic Ballot (PEB)* interface. As discussed in Chapter 5, a PEB is a small module that communicates with the iVotronic via a magnetically switched infrared (IrDA) bidirectional data interface on the face (voter side) of the terminal. PEBs are used for several different kinds of functions. Some of these functions are intended to be performed at the county headquarters (e.g., loading ballot definitions and basic configurations), while others are performed by poll workers (e.g., opening the terminals at the beginning of the day, enabling a voter to use a particular ballot, closing the terminal and collecting vote totals). In the mode used in Ohio, the PEB slot is empty whenever a voter is voting.

PEBs are used as external memory devices that communicate through a simple protocol that allows the iVotronic to read and write memory blocks stored in the PEB. Access to PEB memory is not protected by encryption or passwords, although some of the data stored on them is encrypted (see Section 6.4). PEBs themselves are proprietary devices (and are apparently not commercially available except through ES&S). However, they employ a widely-used infrared communication standard (called *IrDA).*

In spite of the proprietary nature of the "official" PEB, we found it to be relatively simple to emulate a PEB to an iVotronic or to read or alter the contents of a PEB using only inexpensive and commercially available IrDA-based computing devices (such as Palm Pilot PDAs and various mobile telephones).

Most of the administrative and poll worker functions of the iVotronic (e.g., pre-election ballot loading, enabling voting, etc) require the insertion of a properly configured "supervisor" PEB and, in some cases, the entry of a password on the terminal touchscreen. However, we found it to be possible to defeat both of these security mechanisms This makes practical several possible attacks at polling stations.

Unauthorized screen calibration and configuration

One of the simplest, and yet most important, configuration parameters of the iVotronic DRE is the calibration of its touchscreen input sensors. Calibration (which can be performed in the field through the screen itself) affects how voters' tactile input "maps" to different locations on the screen. If the procedure is performed incorrectly (or has been deliberately altered), voter choices might not be correctly recorded.

It is easy to surreptitiously re-calibrate the screen of an iVotronic terminal in a way that allows most input to behave normally but that denies access to specific screen regions (e.g., those corresponding to certain candidate selections).

Access to the screen calibration function of the iVotronic terminal requires the use of a supervisor PEB during power-up (e.g., after voting or at idle times). No password is required. Any supervisor PEB is sufficient for this purpose, even one not specifically configured for the correct precinct or obtained from some other jurisdiction (e.g., through secondary markets such as eBay). A home-made PEB emulator (e.g., a specially programmed Palm Pilot and a small magnet) is also sufficient. The procedure requires about one minute and is, from a distance, largely indistinguishable from normal voter behavior.

A terminal can be maliciously re-calibrated (by a voter or poll worker) to prevent voting for certain candidates or to cause voter input for one candidate to be recorded for another. The terminal will remain in this state until the problem is detected (e.g., through voter complaints) and the terminal correctly re-calibrated by poll workers (which may require consultation with the central county office). Voters may or

may not recognize that their votes are not correctly recorded, depending on voter training and other factors.

While a maliciously calibrated terminal may be noticed by voters and can, in principle, be corrected in the field, the attack is extremely simple for a poll worker (or other person with access to a PEB) to carry out and practical even without a PEB, and so may represent a serious practical threat. We note that iVotronic behavior consistent with such attacks has been reported in various jurisdictions during actual elections.

Undocumented PEB features can be used to bypass password checks

Many of the more sensitive iVotronic administrative functions (closing the polls, clearing the terminal, etc) require the entry of passwords in addition to the insertion of a supervisor PEB. However, there is a special *Quality Assurance (QA)* PEB type recognized by the iVotronic firmware that behaves essentially as a supervisor PEB but that, when used, does not require the entry of any passwords. This PEB type does not appear to have been described or documented in any of the ES&S manuals or training materials provided to our review.

This undocumented PEB feature can be used to neutralize the security of any iVotronic administration features that depend on passwords, no matter how carefully passwords are managed by a county. Anyone with such a PEB – whether it was supplied by ES&S, stolen, or emulated with a palmtop computer – effectively has a "back door" that bypasses this basic security check. As noted above, a simple Palm Pilot-type device can be programmed to emulate a PEB. QA PEBs are no more difficult to emulate than regular supervisor PEBs; they are similar to supervisor PEBs but with a single character changed in the communication protocol.

Note that while the QA PEB bypasses password checks, there is another iVotronic security feature required for access to some (but not all) administrative functions, For these functions, a PEB must be configured with the correct *Election Qualification Code (EQC)* (a 32 bit random number assigned for each election). However, as noted in the next section, precinct poll workers (and others with brief access to the poll worker equipment) can easily extract this code from the precinct's supervisor PEB using a palmtop computer.

Note that even without the EQC, however, an attacker (who needs no more access than that provided to a normal voter) with a QA PEB (or an emulated QA PEB) can do a great deal of harm to an iVotronic terminal. For example, the EQC is not required on QA PEBs used to invoke the "clear" function on an iVotronic terminal, a which erases all stored votes and renders the terminal useless for the rest of the election day.

Unauthorized PEB copying and alteration

Anyone with physical access to polling station PEBs can easily extract or alter their memory. This requires only a small magnet and a conventional IrDA-based palmtop computer (exactly the same kind of readily-available hardware that can be used to emulate a PEB to an iVotronic terminal). Because PEBs themselves enforce no passwords or access control features, physical contact with a PEB (or sufficient proximity to activate its magnetic switch and IR window) is sufficient to allow reading or writing of its memory.

The ease of reading and altering PEB memory facilitates a number of powerful attacks against a precinct's results and even against county-wide results. An attacker who extracts the correct EQC, cryptographic key, and ballot definition can perform any election function on a corresponding iVotronic terminal, including enabling voting, closing the terminal, loading firmware, and so on. An attacker who has access to a precinct's main PEB when the polls are being closed can alter the precinct's reported vote tallies, and, as noted in Section 6.3, can inject code that takes control over the county-wide back-end system (and that thus affects the results reported for all of a county's precincts).

Individual precinct poll workers have many duties that involve handling PEBs throughout the election day (whenever a voter votes, for example), and so are in a natural position to carry out attacks that involve altering or reading PEB memory without engaging in suspicious activity.

6.1.2 Physical security, locks and seals

Many aspects of the ES&S system's security as a whole depend on the integrity of the interfaces and removable media associated with precinct equipment. Some of these interfaces and media are protected by software security (e.g., access passwords, encryption, etc); potential attacks against such mechanisms are discussed in other sections of this report. Many interfaces and media are also protected (partly or entirely) by physical mechanisms: locks, seals, and procedures.

Although this study did not aim to conduct an exhaustive analysis of the physical security of the ES&S equipment, we found many of the basic physical security features that protect precinct hardware to be ineffective or easily defeated.

iVotronic

Several features of the iVotronic's physical security were especially problematic:

- A primary mechanism for logging events (including those potentially associated with an attack) on the iVotronic terminal is the RTAL printer. However, the cable connecting the printer is readily accessible to the voter and can be removed easily and without tools or overtly suspicious activity. It is possible to for an attacker to suppress logging simply by unplugging the cable. It is also easy for an attacker to print arbitrary messages on the printer (including ballot choices) by connecting a small handheld computer to the printer cable.

- The PEB interface on the iVotronic terminal is exposed and readily accessible to the user during voting. As noted above, this facilitates several important attacks.

Locks and seals

The mechanical locks supplied with all of the ES&S precinct equipment sent to the source code review team were uniformly of very low-security designs that can be easily picked or otherwise bypassed.[1] Many locks use keys that are apparently identical in equipment shipped to different customers, and so would provide little security even if the locks were improved. In almost every case, it was not actually necessary to operate the locks, since the equipment cases could generally be opened by removing a few screws and the locks bypassed altogether.

Other physical security mechanisms depend on tamper-evident seals. As noted in Chapter 9, we were not provided with samples of the seals used in every county, and so cannot conclusively comment on the security of the particular seals used in Ohio. However, we note that all but the most sophisticated commercially-available tamper seals are often surprisingly easily to defeat.[2]

Even if effective at revealing tampering, seals are inherently limited in the protection they provide. As noted in Chapter 3 and in other reports,[3] seals do not *prevent* tampering; at best they can *detect* it. But in an election, even reliable detection of tampering may be unsatisfying, since if a seal is found to be broken once

[1] For the first weeks of the project, we did not have the correct keys for much of the equipment; we frequently had to pick the locks in order to conduct our analysis.

[2] Roger G. Johnston, 'Tamper-indicating seals'. American Scientist, 94 November-Decemeber 2006.

[3] Matt Blaze et al., *Source Code Review of the Sequoia Voting System*. University of California, Berkeley under contract to the California Secretary of State, July 20, 2007 ⟨URL: `http://www.sos.ca.gov/elections/voting_systems/ttbr/sequoia-source-public-jul26.pdf`⟩.

polling has started, it is unclear what should be done. If the compromised equipment is used, fraudulent votes may be counted. If it is not used, previously cast legitimate votes may be lost (making breaking a seal a simple way for an attacker to destroy votes).

6.2 Critical Errors in Input Processing

At least two critical components of the ES&S system suffer from exploitable errors in functions that process input over their external interfaces. Both the Unity tallying system and the iVotronic terminal have *buffer overflow* software bugs that allow an attacker who can provide input (e.g., on a PEB or memory card) to effectively take control over the system. A buffer overflow in input processing is common type of programming error, one that has been responsible for many security failures in modern computing. Avoiding buffer overflows in input processing is regarded as one of the most basic defenses a system must have.

We found numerous buffer overflows throughout the ES&S system. Several of these buffer overflows – in the Unity tallying software and in the iVotronic terminal firmware – have extremely serious practical security implications. An attacker who can present input to any these systems (on an iVotronic PEB or on an M100 memory card from a precinct) can exercise complete control over the results reported by the entire county election system.

Most seriously, the nature of these vulnerabilities means that there are few barriers to obtaining the access required to exploit them. In the case of the iVotronic system, voter access to the terminal is sufficient. In the case of the Unity system, brief access to any iVotronic or M100 optical scan results media returned back to the county for processing is sufficient. As discussed in the Section 6.3, it is also possible to carry out the attacks against the Unity system by tampering with the firmware of precinct equipment.

6.2.1 Unity

The Unity election management system processes all precinct results and produces the tally reports that, in most cases, constitute the official tallies in races. After polls are closed, precinct-counted ballot results are received into Unity through several different media, including iVotronic PEBs, iVotronic CF cards, and M100 PCMCIA memory cards.

While Unity appears to correctly process properly-formatted results from such media, buffer overflows in Unity allow a maliciously altered iVotronic or M100 tally from a precinct to execute arbitrary software on the computer on which Unity runs, to replace or alter the Unity software, and to make arbitrary changes to the tally database and other election records. There may be no indication to the operator that this is occurring, and a system thus corrupted may continue to appear to operate normally when it is actually running software controlled by an attacker.

Because these attacks are carried out entirely through media routinely brought in to the county headquarters from precincts on election night, an attacker need not have any physical access to the secure county facility in which Unity is located. It is entirely sufficient for the attacker to have access to media (such as PEBs or M100 memory cards) returned to the county at the end of the election, or to equipment (such as iVotronics and M100s) that write to such media. Poll workers handle such media in the normal course of their duties, and may have unsupervised access at various times of the day. And as noted in the next section, a voter using an iVotronic DRE may be able to circumvent the iVotronic terminal in a way that causes it to automatically produce such media when the polls close at the end of the day.

Note that because these vulnerabilities affect the central counting system, a corrupted media attack conducted from *any single* precinct can corrupt results for the entire county.

We have successfully implemented PEB-based attacks against Unity (at the University of Pennsylvania and at WebWise) and have confirmed that such attacks represent a readily-exploitable threat in both iVotronic

and M100-based systems.

6.2.2 iVotronic

The iVotronic terminal firmware has several exploitable buffer overflow errors in its PEB input processing functions. These buffer overflows allow a PEB containing carefully-structured data (or an emulated PEB based on a palmtop computer) to take control over the terminal. The implications of attacks against iVotronics are discussed in Section 6.3.

We found it to be straightforward to exploit the iVotronic buffer overflows in several different ways (by emulation of a QA or supervisor PEB with a palmtop computer or by writing data to a precinct's supervisor PEB) at various times (while opening polls and during the polling day), and with various degrees of access (as a poll worker or as a voter). The exposed nature of the PEB port and the many different scenarios under which it can be exploited make attacks against the iVotronic very difficult to effectively guard against under operational election conditions.

6.3 Ineffectively Protected Software and Firmware

The integrity of election results depends heavily on the integrity of the software and firmware that runs the central election management system and the precinct hardware. The consequences of any attack that alters, replaces or otherwise compromises this software or firmware are sweeping and often impossible to recover from. The security features that protect election software and firmware from unauthorized tampering are therefore among the most critically important safeguards in the system as a whole.

We found exploitable vulnerabilities that allow an attacker to replace or alter the firmware and software of virtually every component of the ES&S system, either by circumventing access controls or by triggering software errors.

6.3.1 iVotronic firmware

The iVotronic terminal is based on an Intel 80386 embedded computer processor controlled by firmware stored on an internal flash memory chip. The firmware is designed to be field-updated through an administrative menu function, with new firmware loaded though the terminal's CF card interface. Four security mechanisms are intended to protect against unauthorized firmware loading:

- Access to the firmware update menu function requires a supervisor (or QA) PEB.
- A 6-8 character password is required to enable firmware update.
- The firmware is loaded through the CF card interface, which can be protected by a sealed sliding cover.
- The firmware update function is disabled while the polls are open.

Unfortunately, these mechanisms are ineffective. We found several practical ways for an attacker to bypass each of these security mechanisms and successfully replace or alter the iVotronic firmware, without knowledge of any passwords or secret election parameters, possession of a PEB, or breaking any seals. We found ways to carry out these attacks even when the polls are open. It is possible, for example, for a voter (with no inside assistance) to load new firmware into an iVotronic after he or she is finished voting.

We found at least three different vectors that an attacker could exploit to load unauthorized iVotronic firmware under various circumstances.

Via direct replacement of the internal flash chip: The iVotronic terminal housing can be disassembled easily without breaking the seal that protects the CF slot. Disassembly requires only the use of a readily available Torx security screwdriver. Once the housing has been removed, the internal flash chip can be removed from its socket, reprogrammed with a standard flash writer, and replaced. Note that while surreptitious terminal disassembly is unlikely to be possible in an active polling place, it may be an attractive option for an attacker who enjoys unsupervised access to stored terminals (e.g., the night before an election).

Via the firmware update menu: This is the most direct attack against firmware. As discussed above, a palmtop computer and a magnet can be used to emulate a QA PEB and bypass the password check. If the polls are open, they can be closed by using a an emulated QA PEB to clear the terminal first. Note that with this approach, the firmware must be loaded though the external CF card interface, which might be protected with a tamper-evident seal (although that seal can be bypassed by removing the housing).

Via the PEB interface, during the polling day: This is perhaps the most serious practical threat to the iVotronic firmware. As discussed in Section 6.2, errors in the iVotronic's PEB input processing code allow anyone with access to the PEB slot on the face of the terminal (including a voter) to load malicious software that takes complete control over the iVotronic's processor. Once loaded, this software can alter the terminal firmware, change recorded votes, mis-record future votes, and so on throughout the election day and in future elections.

Any attack that compromises iVotronic firmware is extremely serious; it can be very difficult to detect whether such firmware has been used in a live election or meaningfully recover once it has. The firmware controls every aspect of the ballot presented to voters, the recorded votes, and the interface to the tally system. Because the RTAL printer is under the control of the firmware, compromised firmware can easily print misleading choices that evade the notice of voters or that cancels the printed ballot (replacing it with other choices) after the voter has left. The discovery of compromised firmware at a terminal casts doubt upon every vote cast at that machine (and, because of additional bugs in the Unity back-end, on the integrity of the results reported countywide as well).

Compounding the problem is the fact that there are apparently no tools available to counties in the ES&S system that reliably extract or audit the actual firmware present in any given terminal. The version number is displayed at boot time, but that is not a reliable indication of whether the firmware has been compromised, since the message is part of the firmware itself. Compromised firmware can display any version number that it wishes to impersonate.

The iVotronic firmware code includes a number of internal consistency checks intended to detect corrupted firmware. While these checks may be able to detect accidental memory errors, they are ineffective against maliciously installed firmware, which can simply bypass or omit the integrity check functions.

6.3.2 Unity software

No single component of the ES&S system is more important to the integrity of election results than the central Unity election management system. Unity is a complex software suite, consisting of many components that share a common database. Securing a county's Unity system therefore depends on each of its components and on the computing platforms on which it runs.

Because Unity (at least as used in Ohio) apparently runs only in a single, secure location in each county, with presumably only trusted staff permitted access to the computers, attack vectors involving unauthorized direct physical access by poll workers, voters or others are a less significant threat here than in precinct equipment sent to the field. However, because the Unity system processes electronic data received from

precincts, it is subject to a number of indirect – yet devastating – attacks that can originate with poll workers or voters, even if they cannot themselves physically touch the Unity computers.

Attacks via input from precincts

As discussed in Section 6.2, malicious input carried on iVotronic and M100 results media can take over the Unity system when it is loaded for counting. This enables many of the most serious and comprehensive attacks we discovered.

Windows environment vulnerabilities

The Unity software runs on an off-the-shelf version of the Microsoft Windows operating system. This platform is very heavily dependent on many aspects of the local computing environment for its security. When used in a networked environment, even behind a network firewall, there are many potential vulnerabilities that can be mitigated only through careful, expert system management.

Unfortunately, the precise requirements for using Unity in a networked Windows environment are not specified by ES&S, and appear to be left to individual counties to manage without specific guidance. An analysis of these issues is beyond the scope of this report, but we caution that there are many potential problems that could arise from the various ways in which the Unity system's Windows platforms might be managed, some of them quite subtle.

6.3.3 M100 firmware

Firmware can be loaded into the M100 optical scan ballot counter by placing a specially structured file on its PCMCIA card (which is also used during polling for ballot definitions and other precinct parameters). If new firmware is present on the card when the M100 is turned on, there is a brief screen prompt and the new firmware is loaded. No password is required.

Any poll worker (or other person) with access to the PCMCIA card slot can thus easily load new firmware. This slot may be protected by a tamper seal in some jurisdictions, but the seal may be able to be bypassed because of the design of the cover mechanism.

Because the firmware is loaded from the same PCMCIA cards used to load ballot definitions, corrupt firmware can also be loaded into the M100 by a corrupted Unity system when an election is provisioned.

M100 firmware controls how ballot definitions are interpreted, the counting and recording of votes, the format of data returned to Unity, and the acceptance and rejection of ballots. The consequences of corrupt M100 firmware are serious, especially given the vulnerabilities in Unity results processing. However, since the paper ballots remain available, they can be recounted if an attack might have occurred.

Unfortunately, as with the iVotronic, there is no mechanism for reliably determining or auditing the actual firmware installed in an M100, so attacks on these devices may be difficult to detect or confirm.

6.3.4 Viral propagation

The software or firmware of almost every major component of the ES&S system can be altered or replaced by input from the other components with which it communicates. In particular, note that, by design or software flaw:

- The Unity system software can be modified by election results media originating from iVotronics and M100s (due to Unity buffer overflows)
- The iVotronic firmware can be modified by configuration media originating from the Unity system (due to iVotronic buffer overflows).

- The M100 firmware can be modified by configuration media originating from the Unity system (due to the design of the M100 firmware management functions).

This confluence of vulnerabilities creates a "closed loop" for viral propagation into every part of the ES&S system through the compromise of a single system component.

For example, a voter can compromise an iVotronic terminal though its PEB slot. The iVotronic, then, may be programmed to create results media (at the end of the election day) that, in turn, corrupts the software of the central Unity system. The compromised Unity system, in turn, may be programmed to load corrupted firmware into all M100s and iVotronics in the county when provisioning a subsequent election. At this point, every major component of the system is running compromised code, which originated with a single attacker with only voter access in a single precinct. Needless to say, such an attack represents a grave threat to the integrity of the elections of any jurisdiction to which this happens.

6.4 Ineffective Cryptography and Data Authentication

Much of the critical election data in the ES&S system – ballot definitions, precinct vote tallies, and so on – are communicated between the central county headquarters and precincts through small removable storage media. In iVotronic DRE-based systems, the primary media are PEBs and, in some cases, CF memory cards. In M100-based precinct counted optical scan systems, the primary media are PCMCIA memory cards.

These media share two important characteristics that make them attractive targets for attack: they have no intrinsic security properties of their own and they may pass through many hands on the way to polling places, during the polling day, and back from polling places. That is, it is simple to read or alter data on these media, and many people may have the opportunity to do so during an election. For example, iVotronic PEBs are handled by poll workers all through an election day, with memory that can be read or written with a standard palmtop computer and a small magnet. PCMCIA and CF cards, similarly, can be readily read or altered with standard laptop computers.

The usual approach (indeed, generally the only practical approach) for securing data stored on such media is the use of cryptographic techniques that prevent meaningful access to data without knowledge of the correct key.

Unfortunately, the ES&S system does not employ cryptography at all in the M100-based optical scan system. The iVotronic DRE system does use cryptography, but errors in its implementation render the protection completely ineffective.

The lack of effective cryptographic protection enables a large fraction of the exploitable vulnerabilities discussed in this report.

6.4.1 Unauthenticated M100 data

M100 PCMCIA cards are used to load ballot definitions and firmware into the M100 and to report tallies back to the Unity system. The data for each of these functions are not cryptographically protected; an attacker with access to an M100 PCMCIA card can easily forge or modify these data. A linear cyclic redundancy code (CRC) is included with the PCMCIA data, but an attacker can easily calculate this; CRC codes are not keyed and are not designed to provide security against deliberate data modification.

6.4.2 Ineffective iVotronic cryptography

The iVotronic DRE uses cryptography to protect data stored on the PEB and in the CF card. The *Blowfish* cipher is used.[4] Unfortunately, the manner in which the encrypted data is stored on the PEBs ef-

[4] Blowfish is a popular public-domain cipher algorithm from the 1990's.

fectively neutralizes the cryptographic protection. The PEB contains an EQC, encoded using an unkeyed (non-cryptographic) algorithm. The EQC is used to encrypt the Blowfish key, which is used to the encrypt the rest of the data on the PEB. That is, although much of data on the PEB is encrypted, there is unencrypted information stored along with it that allows an attacker to easily discover the key.

6.4.3 Poor data validation of precinct results in Unity

The obvious attacks enabled by the lack of cryptographic protection of precinct media include alteration or forgery of data, unauthorized loading of firmware, as discussed in the rest of this report.

Additional vulnerabilities are introduced by Unity's poor validation of various reported precinct data. In particular, the precinct results reported on an incoming M100 PCMCIA card are not checked against the precincts for which the card was originally provisioned. This allows anyone with access to a card to add tally results for extra precincts, which will be added to (or supplant, depending on the mode the Unity operator is using) the true precinct results when read into the database.

6.5 Procedural Mitigations

We believe the issues reported in this study represent practical threats to ES&S-based elections as they are conducted in Ohio. It may in some cases be possible to construct procedural safeguards that partially mitigate some of the individual vulnerabilities. However, taken as a whole, the security failures in the ES&S system are of a magnitude and depth that, absent a substantial re-engineering of the software itself, renders procedural changes alone unlikely to meaningfully improve security.

Nevertheless, we attempted to identify practical procedural safeguards that might substantially increase the security of the ES&S system in practice. We regret that we ultimately failed to find any such procedures that we could recommend with any degree of confidence.

A particular challenge in securing systems that use the iVotronic DRE terminal is the large number of precinct-based attack vectors whose exploitation must be prevented. Effective procedures that accomplish this, even if they existed, would be arduous indeed, and would likely substantially hamper poll workers in their duties, reduce the ability to serve voters efficiently, and greatly increase the logistical challenges of running an election.

It may be possible to deploy a reduced subset of the ES&S hardware and software that excludes components that present the greatest risks. For example, a system that uses only centrally-counted optical scan hardware eliminates many of the threats of precinct-based attacks. We defer to the expertise of the Ohio election officials to determine whether it is possible to use a version of the ES&S system with reduced functionality in a way that presents an acceptable level of risk to the integrity of their elections.

CHAPTER 7

ES&S SPECIFIC WEAKNESSES AND THEIR IMPLICATIONS

In this chapter, we describe specific weaknesses found in the reviewed ES&S systems. For each discovered vulnerability, we detail the prerequisites necessary to compromise security and/or reliability as well as the impact of such attacks. Since we made no attempt to subject the various ES&S software and hardware systems to equal scrutiny, the number of findings among the perspective ES&S components should not be used to infer any conclusions regarding their relative security. We did, however, concentrate our efforts on areas we deemed most apropos to security and reliability.

It is worth emphasizing that we do not assert that the vulnerabilities described in this chapter constitute a complete and exhaustive listing of the security weaknesses exhibited by the reviewed ES&S systems. The reviewed components consist of nearly 670,000 lines of code, encompassing twelve programming languages and five hardware platforms. Given the extraordinary scope and complexity of the ES&S voting system, it is infeasible that any study could comprehensively analyze all hardware components and software modules in a reasonable amount of time. Consequently, it would be improper to conclude that a particular system can be "fixed" merely by repairing the vulnerabilities listed in this chapter, particularly given the overarching engineering issues identified in Chapter 8.

Throughout this chapter, vulnerability details containing proprietary information are provided in a separate confidential annex. References to such proprietary details are denoted using footnotes.

7.1 Unity

7.1.1 Unity ERM buffer overflow when reading a Master PEB

Description: The Election Reporting Manager (ERM) component of Unity is used to compile results from the precincts. The most common method of delivering the results is on a Master PEB. There is a stack-based buffer overflow in the ERM which can be exploited when election, pre-election, or testing results are processed. To exploit this vulnerability, an attacker can create a specially-crafted PEB that will allow arbitrary code execution. As a result, an attacker has the ability to gain full control of the machine running the Unity server.

The EQC on the results PEBs is not checked, so an attacker does not need to know a valid EQC to create a malicious PEB.

Prerequisites: This attack requires access to a PEB and the facilities to write data to the PEB, such as our pebserial tool. The PEB must be read using the PEB reader.

Impact: Since the attack enables arbitrary code execution on the Unity server, an attacker has the ability to gain full control of the machine running the server. Therefore, the attacker could perform activities such as installing a virus or trojan that could then be used to change the election results and spread the virus to other ES&S components (e.g., the iVotronic or M100 machines). Scenario unity.2 in Section 9.3.2 demonstrates an attack using this vulnerability.

7.1.2 Data from M100 can cause a buffer overflow in Unity

Description: Data from the M100 optical scanner can cause a buffer overflow in Unity. The data, which includes election results and audit data, are stored and transferred on a PCMCIA card.

The ERM module of Unity expects a fixed maximum number of precincts to be reported on a given PCMCIA card. This number is hardcoded in the source code of the program. However, the actual number of precincts is read from the PCMCIA card, and the attacker can thus set it arbitrarily high (within limits of the integer datatype used).[1] To exploit this vulnerability, an attacker can create a specially prepared PCMCIA card that will allow the execution of arbitrary code. The attacker can gain full control of the Unity server in this manner.

Prerequisites: If an attacker is to exploit this vulnerability, he or she must have either a) compromised the M100 optical scanner or b) have access to the PCMCIA card that is used to transfer the data to the Unity back-end system.

If the attacker wants to exploit this vulnerability using an exploit of an M100 optical scanner, he or she must gain enough control over the M100 to be able to make it output arbitrary data. This can be accomplished by loading modified firmware to the M100. For possibilities of loading new firmware to the M100, see Section 7.3.1.

If the attacker has unmonitored access to the PCMCIA card when it has election results, he or she can modify the data on this card, since the integrity of the data that M100 saves on the PCMCIA card is not sufficiently protected (see Section 7.1.4).

A poll worker can carry out this attack in either of the two ways described above.

We have not fully implemented the attack, thus we have not experimentally confirmed that the vulnerability is exploitable. We have performed extensive code analysis, as well as initial experiments towards performing the attack, and everything we found indicates that this vulnerability is exploitable.

Impact: This attack allows the attacker to run arbitrary code on the Unity server. Therefore an attacker has the ability to gain full control of the machine running the server and could alter election results or compromise the Unity software. He or she can then use the compromised software to spread the attack virally to other election equipment (iVotronic, M100, M650).

7.1.3 Unity accepts memory media with unauthorized precinct results

Description: Unity (via the Hardware Programming Manager module) passes election definition and other data to an M100 optical scanner via a PCMCIA card. This card contains information identifying those precincts for which the M100 will accept ballots. The election results are stored on the PCMCIA card and are transferred back to Unity on the same card for processing.

[1] See Section 22.7.1.2 in the confidential Annex.

However, Unity does not check whether only the results for which the card, and thus the M100 scanner, were configured are actually returned. A poll worker can thus take the card with election results and insert results for a precinct for which the card was not configured.

A malicious poll worker can therefore not only modify the results for his or her own precinct, he or she can influence the results for other precincts as well. We have verified experimentally that this vulnerability is exploitable.

When a PCMCIA card with results is entered into Unity, Unity indicates how many precincts have been reported on the card. Thus an attentive operator using Unity can catch this attack, in case he or she has a list of cards (their serial numbers) and the number of precincts that should be reported for each card.

This attack raises a question whether a similar one can be carried via PEBs that contain results from iVotronic. This appears to be possible, because, firstly, it is possible to report results for multiple precincts on a PEB and secondly, analysis of the source code of Unity indicates that Unity does not check whether the returned PEB contains only results for precincts for which it was configured. The attack via a PEB (as opposed to one via a PCMCIA card) was not experimentally verified.

Prerequisites: To exploit this vulnerability, an attacker must have either compromised the M100 optical scanner or have access to the PCMCIA card that is used to transfer the data to the Unity back-end system. For more details on prerequisites, see Section 7.1.2. The attacker can also return an independently prepared PCMCIA card for results processing.

Impact: An attacker (such as a malicious poll worker) can influence the results of an election by reporting fictitious results for different precincts. The attacker will not only have complete control over the results being reported from the M100 from his or her precinct, but can also report extra results for other precincts.

7.1.4 Integrity of data on a PCMCIA card is not protected

Description: A PCMCIA card is used to transfer data from the M100 optical scanner to an iVotronic. The integrity of the data on the card is protected only by CRC checks. The protection is not cryptographically strong, and CRC checks were not designed to protect against deliberate data modification. This protection thus does not prevent an attacker to change the data on the card in such a way that the data pass the CRC checks. An attacker can therefore easily modify, delete or add data to the correct results that are stored on the card. This greatly facilitates the attacks described in Sections 7.1.2 and 7.1.3.

Prerequisites: An attacker needs to have access to the card when the data (election results and audit data) are being transferred to Unity.

Impact: If the attacker has access to the card when it is transferred to Unity, he or she can arbitrarily modify the data on the card. Thus the attacker will have complete control over the results coming from the precinct(s) on the card. He or she can also attempt to overtake the Unity back-end system (see Section 7.1.2) or change the vote totals for other precincts (see Section 7.1.3).

7.1.5 Processing audit data can cause a buffer overflow of a global variable

Description: Audit data uploaded to Unity can cause a buffer overflow of a global variable. The ERM module uses a buffer with a fixed amount of allocated memory to input a string with a variable length.[2] The

[2] See Section 22.7.1.5 in the confidential Annex.

attacker can prepare the input to be longer than the pre-allocated buffer. The audit data enter the system via a PEB or a CF card prepared by the iVotronic.

The buffer in question is a global variable (thus is not allocated on the stack) and the number of bytes by which the attacker can overflow the buffer is limited. Therefore, the attacker might not be able to use this vulnerability to execute arbitrary code on the server running Unity.

Prerequisites: If an attacker is to exploit this vulnerability, he or she must have either compromised the device that put the data to the PEB or the CF card (the iVotronic) or have access to the PEB or the CF card that is used to transfer the data to the Unity back-end system.

Impact: The attacker can corrupt the memory of the ERM module, i.e. change the value of some data or influence the control flow. In the event that the memory layout is favorable to the attacker, he or she might be able to realize more severe attacks. As explained above, this attack involves a global variable, therefore the attacker is not guaranteed to be able to overtake the machine running Unity.

7.1.6 Unity decrypts a PEB using the EQC from the PEB

Description: Unity uses the Election Qualification Code stored on the PEB to decrypt the encryption key stored on the PEB.

If Unity used the original EQC that was used for qualifying the other election equipment and compared it with the EQC on the incoming PEB, it would ensure that the data is coming from a properly qualified iVotronic. However, this is not the case.

Unity reads the EQC from a PEB, then verifies the EQC and key data with a CRC, and finally uses the EQC value to decrypt the encryption key, which is subsequently used to decrypt the data on the PEB. The only protection is thus the CRC check, and the attacker can choose data to be such that the CRC check passes.[3]

Prerequisites: The attacker must be able to obtain a PEB - possibly a new one or one used in previous elections and must be able to return it with the PEBs that are to be processed by Unity.

Impact: An attacker can take advantage of this vulnerability to perform the buffer overflow attack on Unity (see Section 7.1.1) or to return incorrect results for the precinct(s) for which the PEB was configured. Note that the attacker does not need access to a PEB used in an election that is in progress and does not need to know the correct EQC.

7.1.7 Unity contains large pieces of duplicated code

Description: There are three copies of a large segment of code in C++ performing the same task, namely processing audit data.[4] The three copies might have been obtained by cut-and-paste, and slightly modified in subsequent versions. This is an unsafe programming practice. The result is that the three copies have slightly different security properties. For instance, there are security issues in one of the three copies - see Sections 7.1.8 and 7.1.5.

This does not create a vulnerability by itself, therefore we do not list prerequisites and impact here. The purpose of this section is to point out an instance of an unsafe programming style that may lead to security problems.

[3] See Section 22.7.1.6 in the confidential Annex.

[4] See Section 22.7.1.7 in the confidential Annex.

7.1.8 Unity contains many small buffer overflows

Description: There are numerous occurrences of unsafe memory accesses throughout the Unity code. This is indicative of fragile programming style, which might lead to errors that lead to unpredictable behavior of the program and/or open vectors of attack.

In parts of code that process incoming election results and audit data, there are numerous instances of small buffer overflows (1 or 2 bytes). There are also 'hardcoded' buffer overflows, i.e. memory write accesses that are determined by a constant fixed in the code, and that are such that the access is guaranteed to be beyond allocated memory.[5]

While these problems will not result in the attacker running arbitrary code, they cause memory corruption. In case of 'hardcoded' buffer overflows, the corruption of memory will occur regardless of activities of an attacker.

This does not create an exploitable vulnerability by itself, therefore we do not list prerequisites and impact here. The purpose of this section is to point out an instance of an unsafe programming style that may lead to security problems.

7.1.9 SQL injection to bypass authentication in EDM, ESSIM, and Audit Manager

Description: The authentication process for the Election Data Manager (EDM), Ballot Image Manager (ESSIM), and Audit Manager components of the Unity election management system can be bypassed with a simple SQL attack.[6]

The authentication code used by both EDM and ESSIM is considered to be a part of the Audit Manager's functionality. The user manual for EDM explicitly states that the Audit Manager provides security for election and should always be used.

Scenario unity.1 in Section 9.3 demonstrates an attack using this vulnerability.

Prerequisites: The attacker needs access to the Unity election management system. The attacker also needs to know the name of the table that contains users' information. This can either be guessed or found in the Unity documentation; for example, it is in the functional specification for EDM.

Impact: This injection works by bypassing authentication for EDM, ESSIM, and Audit Manager. As a result, if user accounts are not enforced at the OS level, anybody could login to any of the three applications. Once the attacker is logged on, he/she could view in clear text or change the iVotronic passwords (which do not seem be available in clear text outside of EDM), or he/she could change logs for EDM and ESSIM (using the Audit Manager).

The same SQL injection does not work for other Unity modules, since they use different logging and authentication mechanisms, or they do not use any authentication mechanism at all.

7.1.10 An M100 PCMCIA card can be read multiple times without warning

Description: In the ERM module of Unity, if the operator chooses the "add-to-mode" settings when pro-

[5] See Section 22.7.1.8 in the confidential Annex.
[6] See Section 22.7.1.9 in the confidential Annex.

cessing a M100 card, the same card can be read in multiple times without warning the operator that such a thing has occurred. There is graphical notification that could alert the operator about which results have been read in and from what precinct they came, but there are no warnings that there were multiple reads of the same card. It would also not be obvious that this has occurred.

However, there are multiple modes for processing M100 cards, "add-to-mode" and "replace mode." The difference between the two modes is that in "replace mode" if a precinct has already been read into the ERM, then a warning is presented asking the operator if he or she really wants to replace the results for that precinct. In "add-to-mode", this is not the case, and all results are added together.

In large scale elections, with multiple polling devices per precinct, it is not clear whether the "replace mode" can be used, because it would imply that there is only one data device for returning results per precinct. With multiple polling devices per precinct this is not the case.

According to the documentation,[7] there appears to be no way to remove duplicated results once read into the ERM. The only way to correct the mistake would be to zero the results and start from the beginning.

Impact: Reading in results multiple times from a PCMCIA card would skew vote tallies and change the outcome of an election.

7.1.11 Assumptions on the environment for Unity are unspecified

Description: The assumptions for the environment in which Unity is running are left unspecified. This includes the configuration of the operating system (Microsoft Windows), networking environment, services running on the computer, etc. Thus it is not clear what are the security guarantees that Unity expects from the host system.

It is possible to assume that Unity will be run in a relatively isolated environment in county election headquarters and is used only by trusted staff. However, even in this case, many issues need to be carefully considered, including network setup, possible interference with other processes running on the machine, updates and patches to the operating system and removable media that are read by the machine. A concrete example is that "autorun" should be disabled for all removable media drives. If it is not the case, the program run by "autorun" may compromise the host machine.

The assumptions on the security of the host system are not specified by the vendor. Instructions for the operators for setting up and managing the server on which Unity is running are also not provided.

Impact: Many potential platform security issues may be present in any county Unity installation. Enumerating all issues caused by the host system and resulting attacks is beyond the scope of this study.

7.1.12 iVotronic Image Manager bundled with vulnerable Java Runtime Environment

Description: The Windows installer for the iVotronic Image Manager includes an install of Sun Java Runtime Environment 1.4.2_8. This version of Java has a known, exploitable vulnerability in its GIF image display code.[8] iVotronic Image Manager is written in Java and supports importing GIF graphics for political parties. These graphics are displayed by the iVotronic Image Manager when previewing ballots. Anyone

[7] *The Unity Election Reporting Manager Users Guide*

[8] US-CERT, *Vulnerability Note VU 388289: Sun Microsystems Java GIF image processing buffer overflow*. `http://www.kb.cert.org/vuls/id/388289`, January 2007.

who provides a GIF image to be imported into iVIM (or any GIF images downloaded from, e.g., Google) could potentially exercise this exploit.

We have not experimentally confirmed that this vulnerability is exploitable in iVotronic Image Manager.

Impact: A malicious GIF file could lead to arbitrary code executing as the user running iVotronic Image Manager, and writing to any files allowed for that user.

7.2 iVotronic

7.2.1 Election encryption/decryption key is recoverable from a PEB

Description: The per-election encryption key used to protect vote and audit data is trivially recoverable from a qualified PEB. The election key is itself encrypted using the election qualification code (EQC) and is stored as ciphertext on the PEB. However, the EQC is written to the PEB in unencrypted form. Hence, access to the PEB reveals the EQC which can then be used to decrypt the encrypted election key. This is the electronic equivalent to storing a secret in an impenetrable safe and then painting the safe's combination on its door.

Prerequisites: Recovering the encryption key requires physical access to a qualified PEB and the ability to communicate to it via an infrared (IR) link.

Impact: The encryption key is used by Unity and iVotronics to protect election definitions, vote information, and audit logs. Knowledge of the encryption key and physical access to a PEB permits the following actions:

- Arbitrary modification of data on the PEB, including election results (if the PEB has been used to collect votes from an iVotronic) and the election definition.
- Decryption of audit logs stored on the PEB or a Compact Flash card.
- The ability to emulate PEBs (e.g., using a Palm Pilot with an IR port) that are accepted by iVotronics (see Section 7.2.2).

7.2.2 PEBs may be emulated using infrared-capable devices

Description: The iVotronic communicates with PEBs using a proprietary IR protocol. Any device with IR capability can be used to emulate a PEB. Since PEBs and the iVotronic contain magnets and use magnetic switches to detect each other's presence (see Section 5.2.4), communicating with the iVotronic requires close proximity to the iVotronic's voter-facing PEB interface.

As is illustrated in Figure 7.1, we were able to emulate PEBs using a commodity Palm Pilot and a small magnet (the construction of smaller PEB emulators is certainly possible). Due to the PEB slot's location on the front of the iVotronic, it is difficult for poll workers to monitor for potential PEB emulators without sacrificing voter privacy.

Prerequisites: PEB emulators must be capable of communicating using ES&S' proprietary IR protocol. This protocol is simple (consisting primarily of *send-block* and *receive-block* commands) and can be easily understood by probing a legitimate PEB.

Figure 7.1: A PEB emulator running on a Palm Pilot simulates an initialization PEB during an open election, resetting all terminal passwords to "EVEREST"

Impact: The ability to emulate a PEB enables a wide range of serious poll worker and voter attacks. Attacks involving PEB emulation are described in Sections 7.2.4, 7.2.5, 7.2.8, 7.2.9, 7.2.10, 7.2.11, 7.2.12, and 7.2.13.

7.2.3 PEB Emulators can read and/or write arbitrary data on a PEB

Description: A PEB emulator can read and/or write arbitrary data on a PEB placed in close proximity.

Prerequisites: Since PEBs are activated by detecting the presence of a magnet, the PEB emulator must be placed in close proximity when reading or writing to the PEB. Malicious voters may have sufficient access to the PEB when signing in if the poll worker leaves the PEB on the voter sign-in table.

Impact: As described in Section 7.2.1, the EQC, election encryption key, election definitions, and any stored election results are easily discernible by reading the PEB. Additionally, by writing specially crafted blocks to the PEB, an attacker can surreptitiously load exploit code that can compromise the backend Unity system when results are read from the altered PEB (see Section 7.1.1).

7.2.4 Emulated PEBs permit multiple votes

Description: A voter who can emulate a PEB (by combining the attacks described in Sections 7.2.1 and

7.2.2) can use the PEB emulator (e.g., a Palm Pilot) to authorize multiple voter sessions.

Prerequisites: To authorize a voter session, the voter needs the correct election definition, EQC, and election key. All three may be obtained by querying a PEB (see Section 7.2.3).

Impact: Once a PEB has been read, emulated PEBs can be used to authorize voting sessions in any iVotronic terminal that shares the same EQC, election definition, and election key as the cloned PEB. Hence, a malicious voter can repeatedly authorize himself and cast multiple votes. Since the emulated PEB is accepted by all iVotronics with the same configuration, a distributed version of this attack is possible. Here, pertinent data from a single cloned PEB is distributed to multiple attackers, all of whom may cast multiple votes.

7.2.5 Buffer overflow in poll opening process is exploitable

Description: The iVotronic does not verify that allocated memory buffers are sufficiently large to store variable length strings read from the (possibly emulated) PEB. During the poll opening process, the iVotronic copies variable length strings from the PEB into memory allocated on the machine's stack. If the PEB contains specially crafted large strings, an exploitable buffer overflow occurs.[9] During our testing, we confirmed that a specially crafted PEB or PEB emulator can exploit this buffer overflow to take complete control of the iVotronic.

Prerequisites: The PEB or PEB emulator must contain a suitable malformed ballot definition. Additionally, the (emulated) PEB must have an EQC that matches that stored in the iVotronic.

Impact: The buffer overflow allows an attacker to compromise an iVotronic terminal and take complete control over it. Once compromised, the following actions are available to the attacker:

- upload unauthorized and malicious firmware via either the PEB slot or the unprotected serial port (see Section 7.2.14);
- download current election results and/or audit data via either the PEB slot or the serial port;
- modify or erase current election results and/or audit data;
- print arbitrary text to the VVPAT (including forged VVPAT ballots);
- modify or erase the contents of an inserted PEB or Compact Flash card;
- write random data to the firmware, causing the iVotronic to become unusable until it can be disassembled and reprogrammed

7.2.6 Buffer overflow in "Hotspot" image processing is exploitable

Description: The logic that reads image files from the Compact Flash card has an exploitable stack-based buffer overflow.[10] These *hotspot* image files define areas on the screen that function as touchscreen buttons. To support a variable number of hotspots, the headers for such image files are variable length. If the image file specifies a larger-than-expected number of hotspots, memory copied from the Compact Flash card overflows the allocated buffer.

[9] See Section 22.7.2.5 in the confidential Annex.
[10] See Section 22.7.2.6 in the confidential Annex.

The iVotronic contains two independent code patterns for accessing hotspot images. The first pattern, which is used only in one location, reads a fixed number of bytes from the image file into a local buffer of inadequate size. The second pattern, which occurs in ten different locations, copies an unbounded number of bytes from the image file into a memory location on the stack.

Due to the manner in which variables are arranged on the stack, the first code pattern allows an attacker to overwrite the called function's return address, but not with data under his/her control. Therefore, it cannot be used to execute malicious code. However, it can be used to crash the iVotronic, causing denial-of-service. In contrast, the second pattern allows the attacker to execute arbitrary code.

Prerequisites: To carry out the attack, the attacker must write an appropriately crafted ballot header to the Compact Flash card. There are a number of different methods an attacker might employ to introduce a malicious Compact Flash card into the voting process:

- An election insider could prepare a malicious Compact Flash card that is then sent to precincts;
- A virus that previously compromised the Unity server (see Section 7.1) could modify the Hardware Programmer Manager (HPM) to create malicious Compact Flash cards;
- A poll worker could introduce a malicious Compact Flash card either before opening the polls or during a sleepover;
- A voter could extract the Compact Flash card from the iVotronic, modify it to contain the exploit, and reinsert the (now malicious) card into the iVotronic. This approach requires the voter to break and reattach seals. Additionally, the iVotronic will stop responding when the Compact Flash card is removed. However, a voter could claim that a crash occurred during the normal voting operation and convince the poll worker to reactivate the machine. If the machine is reactivated, the exploit code will then be executed and the iVotronic will be compromised.

Impact: The buffer overflow exploit allows the attacker to execute arbitrary code on the iVotronic, permitting any of the actions described in Section 7.2.5.

7.2.7 Buffer overflow in Supervisor iVotronic initialization process is exploitable

Description: The iVotronic supervisor terminal election initialization process contains a stack-based buffer overflow.[11] A device that is connected to the supervisor terminal during initialization (usually the Unity backend system) can exploit this overflow and execute arbitrary code on the supervisor terminal.

Prerequisites: To exploit the buffer overflow, an attacker needs to either compromise the Unity backend system or connect the supervisor terminal to a device (via its serial port) that mimics Unity.

Impact: Since the supervisor terminal is responsible for configuring PEBs for an election, a compromised iVotronic supervisor terminal can write malicious data to the PEBs in order to exploit vulnerabilities in iVotronic voter terminals (e.g., by exploiting the vulnerability described in Section 7.2.5).

[11] See Section 22.7.2.7 in the confidential Annex.

7.2.8 Service Menu Mode in iVotronic does not require properly configured PEB

Description: By pressing the **VOTE** button and inserting a PEB, the iVotronic enters a service menu mode which allows various configuration and election settings to be viewed and/or modified. In our tests, we found that few checks are carried out to determine whether the inserted PEB is properly qualified. Consequently, any PEB (even one not configured for the current election) can be used to enter the service menu mode.

Prerequisites: Most service menu items require the user to enter a password. The iVotronic defaults to specific passwords unless they have been explicitly changed. Note that password checks are bypassed when a (possibly emulated) Factory QA test PEB is inserted (see Section 7.2.10).

Impact: Anyone with an unqualified PEB (or a device that acts like one) and knowledge of the appropriate passwords can carry out the following actions:

- lock the terminal;
- close the terminal early;
- adjust the date and time (which, in effect, may also close the terminal);
- disable or enable audio ballots for ADA versions of the iVotronic;
- (mis)calibrate the touchscreen (see Section 7.2.13)

7.2.9 (Emulated) Initialization PEB can close polls and reset passwords

Description: An initialization PEB may be used to set terminal menu passwords to arbitrary values and close the polls early. The initialization PEB can be inserted at any time, including when polls are open. We have confirmed in our testing that it is straightforward to emulate an initialization PEB using a handheld device with an infrared port (see Figure 7.1). The initialization process takes approximately one minute to complete and hence can easily be accomplished by a malicious voter.

Prerequisites: A specially formatted PEB (or a PEB emulator) can carry out this attack. No election secrets (e.g., the EQC or the election encryption key) are required.

Impact: By closing the polls before the close of the election, this attack causes denial-of-service against a particular iVotronic terminal. Once closed, an iVotronic terminal cannot be reopened unless a new ballot definition is loaded onto the terminal.

The initialization PEB also causes all iVotronic passwords to be set to values configured on the initialization PEB. By setting these passwords to non-default values or values that cannot be entered using the on-screen keyboard, the attack makes it difficult to restore the terminal to a usable state until another initialization PEB is used to reset the passwords.

7.2.10 (Emulated) Factory QA PEBs bypass all password checks

Description: The iVotronic system contains a backdoor that bypasses all passwords when a *Factory QA PEB* (also called a *Debug PEB*) is inserted into the iVotronic.[12] A Factory QA PEB is a special PEB that

[12] See Section 22.7.2.10 in the confidential Annex.

follows the same protocols as a standard PEB. When a PEB is inserted into the iVotronic, the iVotronic polls the PEB for its type. If the type corresponds to a Factory QA PEB, the iVotronic enters a state in which all password prompts are automatically bypassed.

Emulating a Debug PEB is as easy as emulating a standard PEB (see Section 7.2.2). All PEBs (supervisor, voter, and debug) have exactly the same hardware and use the same protocol to interact with the iVotronic terminal. Debug PEBs differ only in the PEB type identifier they report to the iVotronic. Consequently, a PEB emulator can easily identify itself as a debug PEB.

Prerequisites: The debug PEB can be emulated using any device with an IR port (e.g., Palm Pilot). The emulator must be capable of communicating using ES&S' proprietary IR protocol. This protocol is simple (consisting primarily of *send-block* and *receive-block* commands) and can be easily understood by probing a legitimate PEB.

Impact: Factory QA PEBs are presumably used for testing iVotronics as part of internal quality assurance testing. They function exactly like any other PEB except that they bypass all password checks in all iVotronic menus. Debug PEBs allow any voter or poll worker to access important service menu items (see Section 7.2.8) on the iVotronic without the knowledge of any passwords or other election-specific secrets. Further details of various attacks using emulated Factory QA PEBs are described in Sections 7.2.11, 7.2.12, 7.2.13, and 7.2.14.

7.2.11 (Emulated) Factory QA PEB can clear a terminal, erasing all vote and audit data

Description: A (possibly emulated) Factory QA PEB can be used to initiate the iVotronic's clear-and-test procedure, causing all vote data and audit logs to be deleted from the iVotronic flash memory. Clearing a terminal takes less than a minute and can be done surreptitiously by a malicious voter with a Factory QA PEB or PEB emulator.

Prerequisites: If an election has already been loaded on the iVotronic, an Initialization PEB (or emulator) first needs to be used to perform terminal initialization (see Section 7.2.9). The initialization process closes the open election and enables the "Clear and Test Terminal" option in the iVotronic services menu. No election-specific secrets are required.

Impact: Using a (possibly emulated) Factory QA PEB, a malicious poll worker or stealthy voter can quickly (i.e., within a few minutes) conduct the following attacks:

- Delete all previously recorded vote data;
- Upload new firmware to the iVotronic through the Compact Flash interface. Since the Compact Flash card slot is normally protected by a tamper evident seal (of questionable quality, see Section 6.1.2), uploading firmware in an undetectable manner requires the removal and subsequently replacement (or reattachment) of the security seal;
- Exploit the buffer overflow in the terminal opening process (see Section 7.2.5) to take complete control of the iVotronic. This includes the ability to upload firmware via either the unprotected serial port or the iVotronic's PEB interface.

7.2.12 (Emulated) Factory QA PEB can lock a terminal

Description: A (possibly emulated) Factory QA PEB permits a malicious voter to access the services menu without a password and lock the terminal. The locked terminal cannot be used for voting until it is unlocked by keying in the unlock password.

Prerequisites: A Factory QA PEB or emulator is sufficient to lock the terminal. No election-specific secrets (e.g., EQCs or election encryption keys) are required.

Impact: A locked terminal cannot be used for voting until it is unlocked by keying in the unlock password. Assuming poll workers know the unlock password, this is a minor denial-of-service attack which disrupts the voting process.

7.2.13 iVotronic touchscreens can be miscalibrated to deny certain ballot selections

Description: If the **VOTE** button is pressed while a Supervisor or Factory QA PEB is inserted into an iVotronic, the iVotronic enters the service menu. By pressing the **VOTE** button again, the iVotronic enters the calibration screen mode in which the user is required to press crosshairs shown on the screen to calibrate the touchscreen's sensors. By deliberately touching the screen at incorrect places, it is possible to carefully miscalibrate the screen, preventing the voter from making certain selections while marking a ballot.

Prerequisites: A (possibly emulated) Supervisor or Factory QA PEB is required to recalibrate the iVotronic. It is not necessary for the PEB to contain the correct EQC or any other election specific information. No password checks are used for screen calibration, so even an unqualified PEB from a different election can be used.

Impact: By carefully miscalibrating the iVotronic, a malicious voter with access to a PEB or PEB emulator can prohibit future voters from making certain ballot selections.

7.2.14 Connection to RTAL/VVPAT printer is physically unprotected

Description: The RTAL/VVPAT printer is connected via an unprotected serial port on top of the iVotronic (see Figure 7.2). The VVPAT printer cable is not protected by any seals or fastened by any screws and can be easily unattached by a voter.

Prerequisites: The attacks described below require only a hardware device with a serial interface.

Impact: Access to the iVotronic's serial interface enables the following attacks:

- A voter who can access the election services menu (e.g., by emulating a Factory QA PEB, see Section 7.2.10) can download encrypted audit log information to a device with a serial port. Here, the attacker disconnects the RTAL printer and connects the iVotronic's serial port to a handheld device. If the attacker has knowledge of the election encryption key (e.g., by probing a PEB, see Section 7.2.1), s/he can decrypt the audit log to obtain voting records.
- The iVotronic communicates with the printer using a simple protocol via the serial port. By connecting a device to the exposed printer connector, it is possible to print counterfeit audit information as well as additional VVPAT ballots. This is particularly troublesome given that Ohio law specifies that "*[for] any recount of an election in which ballots are cast using a direct recording electronic voting machine*

Figure 7.2: The connection to the RTAL/VVPAT printer port is physically unprotected

with a voter verified paper audit trail, the voter verified paper audit trail shall serve as the official ballot to be recounted."[13]

- The RTAL/VVPAT printer has the ability to rewind the printer tape by approximately one-half inch. (This scrollback limitation is due to hardware, so a malicious voter cannot sufficiently rewind the printer to read printouts from previous voter sessions.) An attacker can utilize this functionality to repeatedly print over the same line of text, creating the appearance of a paper jam. This attack can diminish voter confidence in the election, particularly since VVPAT records are considered the official ballots of record.

- A malicious voter can implant a small serial device between the printer cable and the iVotronic's serial interface. If not removed, such a device could intercept all communication between the terminal and the printer, including voter selections. The recorded votes may be recovered either by adding a wireless transmitter to the serial device or by retrieving the device at the close of an election (perhaps by another voter acting as an accomplice).

- A small device may be attached to the iVotronic's serial interface that mimics the VVPAT/RTAL printer. The iVotronic will incorrectly detect an attached printer, although no future audit information or VVPAT ballots will be printed to the disconnected VVPAT/RTAL printer.

[13] *Ohio Revised Code (O.R.C.) § 3506.18: Electronic voting machine - verified paper audit trail as official ballot in recount.* ⟨URL: `http://codes.ohio.gov/orc/3506`⟩.

7.2.15 Audit data is insufficiently randomized

Description: The iVotronic does not properly randomize voter selections in its audit logs. To maximize voter privacy, voter selections should be randomized according to a uniform probability distribution (that is, each audit record in the audit log should have equal probability of corresponding to a particular voter record).

The iVotronic uses a weak randomization procedure that results in a partial ordering of voter selections.[14] When a voter casts a ballot, the voter's selections are uniformly at random assigned to one of approximately thirty sectors in the iVotronic's internal flash memory. The voter selection is appended to the first available location in the chosen sector (if the sector is full, a new sector is random selected). Consequently, each sector contains a sequential ordering of voter selection images.

Prerequisites: Access to audit information is required.

Impact: An election official with access to audit information may ascertain a partial ordering of voter selections. Such information may reveal voter trends throughout the election.

7.2.16 VVPAT barcodes contain timestamps

Description: After a voter casts a ballot, a barcode containing the current time is printed to the VVPAT printer.[15] These "timestamps" can be used to reconstruct the ordering of votes, even if VVPAT records are detached from the roll to mask vote order (and protect voter privacy) after the close of the election.

Prerequisites: Access to VVPAT records is required.

Impact: An attacker can reconstruct the order of VVPAT printouts using the timestamp encoded in the barcodes, revealing voter trends throughout the election. Additionally, if the order of voters is also known, the attacker can determine how a particular voter voted.

7.2.17 VVPAT barcodes contain vote information

Description: After a voter casts a ballot, a barcode encoding the voter's ballot selections is appended to the VVPAT record.[16] An example of such a barcode is depicted in Figure 7.3. Under the assumption that most voters cannot decipher barcodes, encoding vote information on a VVPAT that cannot easily be *voter verified* constitutes a dangerous and unnecessary practice.

The purpose of such barcodes is unclear. As noted in Section 7.2.14, Ohio law stipulates that "*[for] any recount of an election in which ballots are cast using a direct recording electronic voting machine with a voter verified paper audit trail, the voter verified paper audit trail shall serve as the official ballot to be recounted.*"[17] The tabulation of election results based on barcodes rather than on human comprehendible text eliminates the safeguards promised by voter verified paper audit trails. Interestingly, the iVotronic operator's manual[18] recommends the use of a 2D barcode reader for tabulating votes in the event of a recount, although

[14] See Section 22.7.2.15 in the confidential Annex.

[15] See Section 22.7.2.16 in the confidential Annex.

[16] See Section 22.7.2.17 in the confidential Annex.

[17] 'Ohio Revised Code (O.R.C.) § 3506.18: Electronic voting machine - verified paper audit trail as official ballot in recount.' (as in n. 13).

[18] Election Systems and Software, Inc., *The iVotronic Voting System Operator's Manual version 9.1*. January 2006.

Figure 7.3: A VVPAT barcode encoding voter selections

the document also warns that barcodes should not be used if prohibited by law.[19]

Furthermore, the counterargument that barcodes significantly improve tabulation efficiency and do not undermine security since they can be verified by the accompanying text is fundamentally flawed. Validating all barcodes requires processing all text portions of the ballot, eliminating the efficiency advantage provided by the barcode. Hence any speedup due to the barcode comes at the expense of failing to validate the voter-verified portions of the audit trail.

If barcodes are not used, they should not be included on VVPAT records. The presence of barcodes may induce an election worker (who is likely under substantial pressure to quickly tabulate election results) to take the quick-and-easy path and utilize the barcodes.

Prerequisites: Reading barcodes from a VVPAT printout requires a commodity 2D barcode reader.

Impact: Election officials may tabulate votes using the barcodes printed on VVPAT records, eliminating the safeguards offered by voter verified paper audit trails.

[19]Ohio's recount procedures are vague on the question of permitted uses of barcodes during recounts. An Ohio Secretary of State Directive stipulates that hand counting must be done by "physical examination of the VVPAT roll" and permits "electronic tabulation" if the hand count shows no discrepancy. It is unclear if "electronic tabulation" includes counting VVPAT ballots with a barcode reader. *See:* Ohio Secretary of State, *Directive 2007-30: Recount Procedures*, retrieved November 25, 2007 from `http://www.sos.state.oh.us/sos/ElectionsVoter/directives/2007/Dir2007-30.pdf`

7.2.18 Accuracy testing mode detectable

Description: The logic and accuracy testing (LAT) of the iVotronic can be performed in two different ways, either automatically or manually. Automatic testing seems to be what is most common. The automatic testing is performed by selecting a special logic and accuracy command in the administrator menu of the iVotronic. The automatic test will then cast votes in a specific pattern without any user intervention. In the manual mode the user manually casts votes, and the result is tallied as a special logic and accuracy result.

The logic and accuracy test is effective at catching problems with the ballot programming, but has very limited security benefits. Any malicious firmware running on the iVotronic can detect that the logic and accuracy test is running, and function properly during this test. During an automatic test, no GUI functionality is ever tested. All our example exploits hook into the GUI functions to perform the malicious activity, so no action at all needs to be taken by the exploit writer to pass the automatic test mode. The manual test mode does utilize the GUI, but a special variable in the iVotronic is set to indicate that test mode is running. The exploit code can check this variable, and whenever it is set the code can behave properly.

Prerequisites: The malicious firmware needs to know what special variable to check to determine if it is in a logic and accuracy test or not.

Impact: A malicious firmware can test the special variable and detect the current operating mode and perform different functions accordingly. For example, the malicious firmware could perform a correct count during the LAT procedure and then introduce errors during the actual voting procedure.

7.3 M100

7.3.1 The M100 does not password protect the procedure for firmware uploads

Description: The procedure for uploading firmware to the M100 does not require a password. If the PCMCIA card inserted into the M100 contains new firmware, the M100 displays the install new software menu, permitting anyone to install unverified firmware.

Prerequisites: To install new firmware an attacker requires a properly formatted PCMCIA card and physical access to the M100 PCMCIA slot.

Impact: The firmware on the M100 controls every procedure and operation of the M100. Therefore, malicious firmware would allow an attacker complete control of the machine.

Possible attacks include:

- altering vote totals
- falsifying results tapes
- altering menu displays
- denying service
- removing the ability to install additional firmware (this subsequently makes the M100 unrecoverable unless by the use of additional hardware)

7.3.2 The M100 does not perform cryptographic integrity checks on firmware uploads

Description: There are no cryptographic integrity checks on new firmware before it is uploaded onto the M100. Any correctly formatted M100 firmware (including malicious code created by an attacker) can be loaded onto a PCMCIA card and the M100 will accept it as valid.

Prerequisites: An attacker needs to have knowledge of the hardware components of the M100 in order to create malicious firmware and would need physical access to the M100 in order to install it. Knowledge of the hardware components of the M100 is not particularly hard to obtain and is within the scope of a sufficiently motivated attacker.

Impact: As stated above in Section 7.3.1 the installation of malicious firmware on the M100 gives the attacker complete control of the machine.

7.3.3 The M100 uses the same facility for configuring an election as it does for firmware uploads

Description: The procedure for installing new firmware and the procedure for loading a new election definition on the M100 are both performed by inserting a PCMCIA card into the PCMCIA card slot on the M100 and turning the machine on. If a PCMCIA card containing malicious firmware was placed in an M100, due to issues described in Sections 7.3.1 and 7.3.2, it is possible that a distracted poll worker could unknowingly and unintentionally install malicious firmware.

Prerequisites: A deliberately altered PCMCIA card

Impact: A poll worker, given an infected PCMCIA card, going through typical election day procedures could unintentionally upload malicious firmware to the M100. Additionally, if Unity were infected, the malicious firmware could be distributed and created directly from Unity and installed by a poll worker on election day.

7.3.4 The M100 locks are easily manipulated

Description: The M100 cabinet, Scanner, Ballot Box, PCMCIA slot and power on/off switch are locked with simple cam locks which can be easily picked in a short time using improvised tools.

Prerequisites: An attacker would require privacy and a paper clip.

Impact: Access to the PCMCIA slot would permit an attacker to perform any of the attacks mentioned here which make use of a PCMCIA card. Additionally, since these locks do not provide any evidence of tampering, the fact that they are so easily bypassed makes discovery of this attack difficult.

7.3.5 The keys for the M100 locks are the same across all M100 machines

Description: There are two keys for the M100. The ballot box key locks the M100 scanner in place and protects the PCMCIA slot, ballot box, and the cabinet doors. The scanner key, turns the power on and off to the M100. These two keys are differ from each other, but are the same for all M100 machines. These keys are available online from ES&S for $5.00 each.[20]

[20] Scanner Key Part Number #75506, Ballot Box Key Part Number #75505.

Prerequisites: Access to either a scanner or ballot box key or both.

Impact: A compromise of the ballot box key would grant the attacker access to the PCMCIA card slot, and complete access to the ballots in the ballot box where an attacker could remove or add ballots. A compromise of scanner key would give an attacker the ability to turn on and off the M100.

A compromise of both keys would create a perfect scenario for an attacker to install malicious firmware. The attacker would first need to compromise the ballot box key, after which he can insert a malicious firmware PCMCIA card. The attacker would then compromise the scanner key, turning the machine off then on in order to reach the software installation menu. Once in the software installation menu, the attacker can install malicious firmware and finish the installation by again turning the machine off then on. This could occur without detection.

Additionally, the key blanks for a scanner and ballot box key are easily duplicated, so a compromise of either key could affect machines nation-wide.

7.3.6 CRC routines used on the M100 are easily derivable

Description: CRC routines are used in many places by the M100 to verify the integrity of the data on the PCMCIA card. An attacker can easily discover the constants used to generate the CRC, alter the data on the PCMCIA card, and regenerate the CRC.

Prerequisites: Access to a valid M100 PCMCIA card and a commodity PCMCIA card reader/writer.

Impact: As described in Section 6.4, CRC routines do not provide cryptographic integrity protection. An attacker can write anything to the PCMCIA card, regenerate the CRC and it will be accepted as valid by the M100.

7.3.7 The M100 PCMCIA cards do not contain cryptographic integrity checks

Description: There is no cryptographic integrity protection used on the M100 PCMCIA card. This means that there is no way to verify that the data on the card was created from an authorized source.

Prerequisites: Knowledge of the CRCs as in Section 7.3.6.

Impact: An attacker can arbitrarily change data on the card. The altered data would then be accepted as valid by either Unity or the M100.

7.3.8 M100 uses data offsets defined on the PCMCIA card to determine pointers used by the firmware

Description: During the boot process, the M100 loads a number of structures from the ballot PCMCIA. Some of these structures contain a reference to other structures in the form of an offset value.

After the structures have been loaded, a function is invoked to translate each offset to a pointer. This is accomplished by replacing each offset with the value of the base pointer increased by the offset. Since the attacker can control the offset values and since the function does not check that the resulting pointers are correct, it is possible to make these pointers point to any location in memory. Furthermore, because the code then writes to some of the pointed structures, it is possible to override any sequence of 4 bytes in memory.

Note: Our effort with this vulnerability was hampered by the fact that we did not have all of the source code for the M100 firmware. We requested the missing firmware source numerous times, but it was never delivered. As a result, we spent a lot of time and energy reverse-engineering the firmware.

This is of particularly concern, because it indicates that the M100 contains firmware that is not contained in the firmware provided to the Independent Test Authorities (ITA). The extra firmware is usually not updated, and seems to be the kernel of the operating system running on the scanner (QNX).

Although the kernel is developed by a third party and could be considered a COTS component, it has to be configured with the appropriate hardware drivers, etc. in order to be used with a particular kind of hardware. Since this configuration would be specific to the M100, it should not be considered COTS.

Prerequisites: The attacker needs a way to craft a special PCMCIA card and needs access to the M100. Sections 7.3.6 through 7.3.7 discuss the details of the prerequisites for this vulnerability.

Impact: A number of different attacks can result from this vulnerability. Perhaps the most prevalent is the ability to perform a denial-of-service that could disrupt the ballot counting process. Addition denial-of-service attacks can be performed on the audit data region of the card or on the card header section, the region that defines the layout for the entire card.

It may even be possible to change vote data, that is if the write that follows the altered pointer is done during poll closing. The M100 could unintentionally write into the counter block region changed votes for particular races and subsequently stuff the ballot box.

7.3.9 M100 accepts counterfeit ballots

Description: Portions of the paper ballots read by the M100 are printed using special inks whose reflection properties cause them to be ignored by the optical scanner. To insure against counterfeit ballots, specially designated areas of the ballot are printed using this special ink. To detect counterfeit ballots, the M100 checks for the presence of marks in these specially designated portions of the ballot. Because photocopied ballots are printed using a single (and detectable) ink, a normally copied ballot reproduces these marks. The optical scanner detects no markings in these areas when legitimate ballots are used and perceives the marks if the ballot is a simple photocopy.

Visual inspection is sufficient to distinguish the two inks used to print the paper ballots. By covering areas printed using the reflective ink, it is possible to create duplicate ballots that are accepted by the M100. Although ballots without the special portions may fail visual inspection if they are ever manually scrutinized (since the portions printed using the reflective ink are absent), better forgeries can be constructed by obtaining IR reflective non-read ink. Such ink is available for general purchase at online vendors, or can be easily made from commodity ink jet printer supplies . The M100 accepts a commodity paper in a variety of weights.

Prerequisites: Access to the M100 ballot box; see 7.3.4 and 7.3.5.

Impact: A motivated forger can produce paper ballots that visually appear legitimate and that are accepted by the M100.

7.4 M650

7.4.1 M650 runs executable on Zip Disk on power up

Description: On boot, the M650 checks for the presence of a *firmware update disk*. Firmware update disks are commodity Zip disks provided by ES&S that contain updated programs (binary executables) and an installer script. The M650 determines whether a Zip Disk is a firmware update disk by checking for the existence of three files.[21] If these files exist and the reported version number does not match the currently installed firmware version, the installer script located on the Zip Disk is executed. Only the contents of the file containing the update version number are examined. That is, the M650 checks only for the existence of the three files and performs no integrity or authenticity checks. Hence, any person with physical access to the machine can load new software onto the M650, either on, before, or after the election. As is also the case with the M100 (see Section 7.3.3), both firmware and ballot definitions are loaded through the same medium (in this case, the Zip Disk drive). By surreptitiously inserting malicious firmware onto a ballot definition disk, a corrupted Unity system could carry out this attack.

Figure 7.4 shows the display on the M650 after a forged firmware update disk was used to load unauthorized (in this case, benign and conspicuous) software onto the machine.

Prerequisites: To successfully load new software onto the M650, the attacker must learn the three filenames that convince the M650 that an inserted Zip Disk is a firmware update disk. These filenames can be easily obtained either by cloning a legitimate firmware update disk or by examining the firmware update procedure, a plaintext shell script stored in the M650 (although the latter technique requires prolonged and unfettered access to the machine). Any person with knowledge of these three filenames and approximately three minutes of unmonitored access to the M650 can replace all software on the machine with malware.

Impact: An attacker who can forge a firmware update disk can take total control over the M650. This includes the ability to learn election results, change results, mimic the behavior of an uncompromised M650 (to avoid detection), thwart future attempts to load new firmware (while reporting success to the operator), and/or cause the M650 to become inoperable until its internal storage can be reimaged using separate and uncompromised hardware.

7.4.2 M650 does not authenticate election definition and election macro language (e-code) files

Description: The M650 accepts any well-constructed election definition file or election macro language (e-code) file located on an inserted Zip Disk. No cryptographic authentication checks are used to ensure that the loaded definitions correspond to the paper ballots. Malicious election definitions and e-code can be trivially loaded by inserting a Zip Disk when the system is booted.

Prerequisites: The attacker requires a Zip Disk containing a seemingly valid election definition (that is, one that conforms to the data structures defined in the M650 Functional Specification document.[22]) The election definition and e-code obtained from a legitimate Zip Disk can be used as a template to construct a malicious election definition.

Impact: Using a malicious election definition Zip Disk, an attacker can trivially swap candidate positions,

[21] See Section 22.7.4.1 in the confidential Annex.

[22] Election Systems and Software, Inc., *Software Specifications: Model 650 Central Count Ballot Tabulator, version 2.0.0.0*. July 2004.

Figure 7.4: M650, with display controlled by forged firmware update disk

ignore arbitrary positions on the ballot, and/or alter the weight (the number of votes) assigned to a particular oval position on the ballot.

7.4.3 M650 fails to validate variable-length strings

Description: The M650 fails to check that constant sized buffers are sufficiently large to hold variable-length strings. A malicious ballot definition can define large strings that exceed the storage capacity of the allocated buffer. In particular, the M650 does not check that the length specified in the ballot definition for the election title does not exceed the number of bytes allocated for its buffer.[23] By specifying a large election title, a malicious ballot definition can overflow the allocated buffer and write arbitrary data to the M650's memory.

Prerequisites: To overflow the stack, the attacker needs a Zip Disk containing a seemingly valid election definition (that is, one that conforms to the data structures defined in the M650 Functional Specification document.[24]) The election definition obtained from a legitimate Zip Disk can be used as a template to construct a malicious election definition.

Impact: Due to its location on the stack, it does not appear that the stack overflow can be exploited to run arbitrary code on the M650. However, it is likely that a malicious ballot definition can sufficiently corrupt

[23] See Section 22.7.4.3 in the confidential annex.

[24] Election Systems and Software, Inc., 'Software Specifications: Model 650 Central Count Ballot Tabulator, version 2.0.0.0' (as in n. 22).

the M650 to cause the machine to halt until it is rebooted. It may also be possible to corrupt election results, although such an attack has not been confirmed.

7.4.4 M650 fails to protect against integer overflows

Description: The M650 dynamically allocates memory on the heap to store indices for the precincts (these indices are later used in the tabulation process). The amount of allocated memory is based on the number of precincts reported in the election definition. For example, if there are 10 precincts and 20 bytes are required to store an index for each precinct, then 200 bytes should be allocated. However, the M650 fails to ensure that the product of the number of precincts and the storage requirement of each precinct does not exceed the maximum value that can be represented as an integer. If the product exceeds this limit, then an integer overflow occurs (in which case the result of the multiplication "rolls over" and appears quite small) and insufficient memory is allocated. An attacker can force this integer overflow by specifying a large number of precincts in the election definition file, allowing him to write arbitrary data onto the heap (the amount of data is limited only by the storage capacity of the Zip Disk).[25]

Prerequisites: To overflow the heap, the attacker needs a Zip Disk containing a seemingly valid election definition (that is, one that conforms to the data structures defined in the M650 Functional Specification document.[26]) The election definition obtained from a legitimate Zip Disk can be used as a template to construct a malicious election definition.

Impact: Under some circumstances, the heap overflow may be exploited to inject code that will take control of the M650.

7.4.5 M650 accepts counterfeit ballots

Description: Portions of the paper ballots read by the M650 are printed using special inks whose reflection properties cause them to be ignored by the optical scanner. To ensure against counterfeit ballots, specially designated areas of the ballot are printed using this special ink. To detect counterfeit ballots, the M650 checks for the presence of marks in these specially designated portions of the ballot. Because photocopied ballots are printed using a single (and detectable) ink, a normally copied ballot reproduces these marks. The optical scanner detects no markings in these areas when legitimate ballots are used and perceives the marks if the ballot is a simple photocopy.

Visual inspection is sufficient to distinguish the two inks used to print the paper ballots. By covering areas printed using the IR reflective non-read ink, it is possible to create duplicate ballots that are accepted by the M650. Although these ballots may fail visual inspection if they are ever questioned (since the portions printed using the reflective ink are absent), better forgeries can be constructed by obtaining IR reflective non-read ink. Such ink is available for general purchase at online vendors, or can be easily made from commodity ink jet printer supplies. The M650 accepts forged ballots made of commodity paper in a variety of weights.

Prerequisites: The M650 is a batch scanner used primarily to scan mail-in (absentee) ballots. To be scanned, forgeries must be mailed to the appropriate office in official election envelopes (which, of course, may also be forged).

[25] See Section 22.7.4.4 in the confidential Annex.

[26] Election Systems and Software, Inc., 'Software Specifications: Model 650 Central Count Ballot Tabulator, version 2.0.0.0' (as in n. 22).

Impact: A motivated forger can produce paper ballots that visually appear legitimate and that are accepted by the M650.

CHAPTER 8

ES&S SOFTWARE ENGINEERING ISSUES

It is widely recognized that large and complex software is more difficult to implement correctly and securely than small and less intricate software. The diffuse control flow, data sources, databases, data structures, and usage patterns inherent in large systems decrease the ability to reason about the behavior of the software. Consequently, complex design and implementation permit the introduction of bugs, some of which may be exploited by malicious adversaries.

Software engineering techniques – "best practice" methodologies and tools used to develop and test complex software – attempt to reduce the number of such bugs. Although no software engineering technique can eliminate all complexity or guarantee secure code, common software engineering practices are effective at improving the correctness and the resiliency of the system. Good software engineering practices are critical to develop secure and reliable software, and are particularly vital in the security-sensitive context of electronic voting. In contrast, bad software engineering practices indicate haphazard design and implementation that is potentially rife with error. Hence, the software engineering practices employed by a system often speak to the overall quality of the code.

In this section, we identify design and coding practices which deviate from our perception of good software engineering techniques.

8.1 Complexity

We were provided with nearly 670,000 lines of code written in 12 programming languages and compiled for five hardware platforms (see Figure 8.1). (Lines of code were counted using SLOCCount[1] and the Linux *wc* utility.) Given the size and breadth of the code base, it is unlikely that any quality assurance (QA) process could conduct a comprehensive analysis of the entire system. As a result, the QA process will likely fail to find unintentional bugs and is even less likely to find malicious code surreptitiously inserted into the source code.

8.1.1 Programming Languages

Approximately 63% of the source code is written using programming languages that are *memory unsafe*. These languages (C, C++, and assembly) allow several classes of vulnerabilities, including stack and heap overflows, integer overflows, and format string attacks. More modern programming languages (for example,

[1] David A. Wheeler, *SLOCCount*. ⟨URL: `http://www.dwheeler.com/sloccount/`⟩.

Component	Platform	Language	Lines of Code
Unity 3.0.1.1	Intel 686 (Pentium)	BAT scripts, C, C++, COBOL, Java, Visual Basic	364,136
AIMS 1.2	Intel 686 (Pentium)	C++, SQL, Visual Basic	59,984
iVotronic firmware 9.1.6.4	Intel 80386	C, x86 assembly	87,157
iVotronic PIC source code	Microchip PIC Microcontroller	C, PIC assembly	1,753
M100 firmware 5.2.1.0	Intel 80286	C, x86 assembly, shell script	29,952
M650 firmware 2.1.0.0 (including "display" utility)	Intel 686 (Pentium)	C, C++ shell script	22,458
VAT (AutoMARK)	Intel IXP425	ARM assembly, C, C#, C++, Visual Basic	104,305
			Total: 669 745

Figure 8.1: Platforms and lines of code associated with each component of the ES&S system.

Java and C#) are *type safe* and are not susceptible to such memory violations, and are therefore sometimes favored when developing security-critical applications. Although techniques exist for annotating C code (comprising roughly 23% of the ES&S source code) to achieve type safety,[2] such approaches were not utilized.

8.1.2 Third-Party Software

ES&S makes use of third-party software, including the Windows and QNX operating systems, CompactFlash device drivers, and PCMCIA flash device drivers. Although such software is useful for rapid software development, it is difficult to analyze the security properties of closed-source third-party systems. Without access to the source code or a time-consuming analysis of the third party software's binary objects, these third-party systems are used as "black boxes" under the (possibly incorrect) assumption that they are secure. Since any vulnerability in a third-party product is automatically inherited by the system using it, such trust is unwarranted for security conscious applications.

8.1.3 Insufficient Documentation

Creating and configuring a ballot definition is an arduous process involving at least the following steps:

- Defining precincts, districts, and polling places

[2] T. Jim et al., 'Cyclone: A safe dialect of C'. In USENIX Annual Technical Conference. June 2002; George C. Necula et al., 'CCured: type-safe retrofitting of legacy software'. ACM Trans. Program. Lang. Syst. 27 2005, Nr. 3, ISSN 0164–0925.

- Assigning precincts and districts to polling places
- Defining candidates, contests, ballot questions, and referendums
- Configuring election options (e.g., candidate positions, votes per contest, etc.)
- Generating ballot style information
- Configuring the layout of paper ballots
- Generating "tabular" files for the paper ballot layouts
- Preparing election data for the M100 and M650
- Uploading election data to Compact-Flash cards and PEBs for the iVotronic

The provided documentation is wholly insufficient for election workers to complete the above requirements. During our ten week analysis of the ES&S voting system, we were unable to configure an election from start to finish (in the process experiencing unexplained and frequent software crashes) despite receiving several hours of vendor training and access to user manuals.

According to the office of the Ohio Secretary of State, a majority of counties in Ohio rely at least partially on ES&S technicians to generate their election definitions and tally election results,[3] practices we presume are due to the complexity of the configuration process and the lack of pertinent documentation. Such outsourcing introduces security risks, as the maintainers of the election delegate critical work to the voting machines' vendor. Subtle miscommunication between election officials and ES&S regarding the wide assortment of configuration options could lead to incorrect configurations and tallying. Since they will not necessarily cause election "Pre-Tests" or logic and accuracy tests to fail, incorrect configurations may be particularly difficult to detect.

Such a scenario appears to have transpired in the November 2007 elections in Ohio. The Mount Vernon news service has recently reported that a contract employee of ES&S double-counted more than 400 ballots by failing to "clear one report before running another report", resulting in incorrect election night results.[4] The mistake was noticed only when the report was regenerated (without repeating the mistake) several days later.

8.2 Improper Message Passing

In several instances, Unity makes use of the Windows filesystem to communicate between components. For example, Unity components pass sensitive unencrypted information to other modules by writing the data to disk. Such message passing techniques pose unnecessary security risks. Unauthorized users with access to the filesystem can gain access to election keys and configurations, bypassing Unity's authentication and authorization mechanisms (many Unity applications require a valid username and password). The ability to access sensitive files is increased when Windows filesharing is activated, a scenario apparently realized in some counties in Ohio.

[3] *Personal correspondence with Ohio Secretary of State Office employee*. November 28, 2007.

[4] *Election board verifies results*. Mount Vernon News, November 29 2007 ⟨URL: `http://www.mountvernonnews.com/local/07/11/29/elc.results.html`⟩.

Additionally, message passing via files creates dangerous *race conditions* in which an attacker who has write access to the filesystem can modify files after they are created by one Unity component but before they are processed by another. An attacker who can overwrite files can cause Unity to process arbitrary and incorrect election results. As before, this attack is worsened when Windows filesharing is employed.

Rather than use files to communicate data, good software engineering practice utilizes operating system features to more securely and reliably share information. Microsoft Windows provides such interprocess communication mechanisms in the form of COM objects, pipes, and sockets.

8.3 Static Code Analysis

It is customary for large and complex software systems (particularly those that operate in security-centric domains) to undergo rigorous internal quality assurance testing. As part of the QA process, static code analysis tools – software that searches the source code for bugs and potential security vulnerabilities – help software manufactures catch potential weaknesses before the product is released and used.

Our analysis of the quality of the source code indicates that an acceptable level of static code analysis was never performed. During our review, we found several notable programming errors, all of which were detected using readily available source code analysis tools and techniques:

- Using the Microsoft Visual Studio 2005 compiler (a more recent version of the compiler than that used by ES&S), we discovered that compilation failed due to several serious programming errors. For example, integers declared in "for loops" were used in out-of-scope contexts. This constitutes a violation of the ANSI C++ standard and results in compile time errors under most C++ compilers with which we are familiar. The compiler used by ES&S (Microsoft Visual Studio 6) optionally allows such unorthodox C++ code (presumably by inferring correct addresses for out-of-scope variables).

  Additionally, we discovered at least one instance when a C++ function call that takes no arguments was invoked without the required parentheses. Under most C++ compilers (including later versions of Visual Studio), such references to function names resolve to the address of the function and when invoked in a standalone fashion result in compile time errors. Oddly, such programming practices are accepted under older versions of Microsoft's compiler.

  Arguably, these programming practices do not constitute errors since they are accepted by the compiler. However, incorrect function invocation and the use of out-of-scope variables is a dangerous practice. In both cases, the C++ compiler must infer the programmer's intent (e.g., the scope of the variable or that the function reference should be treated as a function invocation). An incorrect conclusion could produce undesirable behavior. Additionally, if in the future the code is compiled using a different (and more standards-compliant) compiler (including more recent versions of Microsoft's C++ compiler), the best-case scenario results in compilation errors. At worst, the nonstandard programming practices could result in unstable and unpredictable behavior.

- The ES&S source code makes extensive use of memory-unsafe string operations. In particular, the *strcpy* and *sprintf* functions are frequently used throughout the various ES&S system components. These functions copy data into a buffer under the (unchecked) assumption that the target buffer is sufficiently large. If the size of the copied data exceeds the size of the buffer, the strcpy and sprintf functions write beyond the allocated space, potentially overflowing onto other data or process flow structures. Such a condition is known as a *buffer overflow* and represents a common (if not the most common) cause of software exploits and instability.

To prevent possible buffer overflows, memory-safe versions of *strcpy* and *sprintf* should instead be used. The *strncpy* and *snprintf* functions (note the additional *n*) take as a parameter the maximum number of bytes to write to the target buffer, eliminating the possibility of writing beyond the allocated space (assuming correct buffer sizes are provided).

Static source code analysis tools such as Fortify SCA[5] can be used to locate the use of memory-unsafe string operations in a system's source code. In addition, some compilers, including newer versions of Microsoft's Visual C++ compiler, produce compile-time warnings when the *strcpy* and *sprintf* functions are used.

- Using the Fortify SCA source code security analyzer, we were able to quickly identify several programming errors. Although such security analysis tools are imperfect, they serve as a useful means to quickly locate security vulnerabilities. (More thorough testing and analyses are required to find bugs missed by automated tools and to rule out false positives.) In our testing, Fortify SCA reported hundreds of "hot" (those deemed most likely to be correctly diagnosed and potentially exploitable) buffer overflows in Unity's source code. To a lesser extent, Fortify SCA also noted many other types of security problems, including string format vulnerabilities, integer overflows, and the use of non-null-terminated strings (in C and C++, a special character called the null character is used to denote the end of a string). Some of the reported errors are likely false positives (i.e., they refer to correct code) and many of the diagnosed problems cannot necessarily be exploited by an attacker to gain control of the system. However, the discovery of so many potential vulnerabilities implies that serious security and reliability problems likely do exist (a conclusion we repeatedly confirm; see Chapter 7) and, arguably equally troublesome, indicate that the vendor did not sufficiently validate their code.

[5] *Fortify Source Code Analysis (SCA).* ⟨URL: `http://www.fortifysoftware.com/products/sca/`⟩.

CHAPTER 9

ES&S EXAMPLE ATTACK SCENARIOS

This chapter reports on the "red-teaming" exercises performed to evaluate the ES&S Voting System. Prior to the study reported here, the ES&S system had not received a detailed source code and red-teaming security review. There were two studies, however, that should be mentioned. In December, 2006, a team, led by Florida State University's (FSU) Security and Assurance in Information Technology (SAIT) Laboratory, was commissioned to conduct a static software analysis on the iVotronic's version 8.0.1.2 firmware source code. The intent was to determine and identify flaws, vulnerabilities or anomalies, if any, that may have caused or contributed to the higher than expected under-vote rate in the District 13 Race between candidates Vern Buchanan and Christine Jennings.[1] The second study was carried out by two Ohio privacy activists James Moyer and Jim Cropcho. They demonstrated how the time-stamped paper trails produced by the iVotronics could be combined with a list of voters in the order they voted to determine which voter voted and in which way. Since Ohio law permits anyone to request and obtain these two documents, this is a clear violation of the secrecy of personal ballots.[2] We did not attempt to reproduce the findings of either of these groups.

To carry out the red team experiments, we crafted special tools that are designed to obtain information about an election and to circumvent the operations of an election. We also performed a number of physical attacks that enabled us to use the equipment in ways that allowed us to alter or thwart an election.

The first section of this chapter presents some of the special purpose tools that we developed. Next, the security seals and physical security issues are discussed. Finally, a number of explicit attack scenarios are presented.

9.1 Tools

There are several basic tools that we have built to make the process of developing attacks on the ES&S system more efficient and repeatable. These are a framework for delivering the malicious payload to the iVotronic system, a tool for reading and writing PEBs, a serial debugger for the iVotronic, a tool for reading and writing M100 PCMCIA memory cards, a JTAG hardware debugger, and a tool for extracting QNX files from the M100. These are discussed in the following sections.

[1] Alec Yasinsac et al., *Software Review and Security Analysis of the ES&S iVotronic 8.0.1.2 Voting Machine Firmware*. For the Florida, Department of State, February 23, 2007 ⟨URL: `http://election.dos.state.fl.us/pdf/FinalAudRepSAIT.pdf`⟩.

[2] Declan McCullagh, 'E-voting predicament: Not-so-secret ballots'. CNET News.com, 2007 ⟨URL: `http://www.news.com/2100-1014\_3-6203323.html`⟩.

Copies of these tools have been delivered along with the private part of this report.

9.1.1 A framework for delivering a malicious payload to iVotronic systems

We have developed a collection of routines that infect the iVotronic firmware and take control over key voting machine functionality. We refer to the collection as the "payload," because it is delivered by a Personalized Electronic Ballot (PEB) as part of an exploit. More explicitly, the payload is meant to be used by exploits in order to infect a voting machine when a PEB is inserted in the iVotronic. The payload is made up of four different parts: shellcode, flasher, linker, and hooks. These are discussed in the following sections.

Shellcode. The shellcode is a small (32 bytes) piece of code that is shipped with the exploit. The main purpose of the shellcode is to execute code stored on the PEB. This is achieved by loading one block of the PEB into RAM and then jumping to it.

Flasher. The flasher code, which is part of the malicious code on the PEB, is loaded into RAM by the shellcode. The task of the flasher is to first load the rest of the payload (including the linker) into RAM and then to copy this code to an unused area of the iVotronic's firmware flash memory. The code is copied to the flash memory so that it will be persistent. The final task of the flasher is to execute the linker.

Linker. The linker's task is to modify the original iVotronic firmware so that key function calls are intercepted and diverted to the "hooks." More specifically, the linker overwrites the target addresses of the function calls that are to be intercepted and replaces each with an address that points to the payload's hooks. The linker also updates the firmware's Cyclic Redundancy Check (CRC), so that it will pass the iVotronic validation tests.

Hooks. The hooks are a set of functions that are linked into the original firmware. Most hooks perform malicious activity like stealing votes or infecting PEBs inserted into the iVotronic with exploits.

An Example Malicious Payload Using the Framework. One version of the payload intercepts control just before the vote summary page is displayed. The payload steals votes at this point by changing them to select the attacker's candidate. The voter could notice that the ballot has been modified and try to correct the mistake by recasting his/her vote. If this is the case, the malicious firmware detects that the miscast vote has been discovered by intercepting the function that handles the pressing of the "back" button on this page. That is, if the vote changing is detected, the malicious firmware stops stealing votes for a period of time so that it is less likely that someone will discover that something is going on.

The framework could also be modified to use a Compact Flash card as the delivery mechanism, instead of a PEB.

9.1.2 Pebserial: a tool for reading and writing PEBs

Pebserial is a program we developed to read and write PEBs using the serial interface. Pebserial configures the serial interface in raw mode and implements the simple serial protocol used for communication between PEBs and ES&S voting machines (i.e., iVotronic and Unity). Pebserial has the following functionality:

- it retrieves the PEB's Election Qualification Code (EQC) and general election information, such as the PEB's type (e.g., supervisor, user), serial number, and version;
- it reads data blocks stored on the PEB;

- it writes blocks of data to the PEB.

By leveraging these basic operations, pebserial allows one to dump the contents of a PEB and to create PEBs with arbitrary contents.

Pebserial can correctly handle both unencrypted and encrypted PEBs. For encrypted PEBs (identified by a particular value in one of the fields of the EQC), the data is stored in encrypted format using the Blowfish algorithm. It is trivial to determine the encryption key, because the key is formed from fields of the EQC, which is always stored in the clear in the PEB. Therefore, encryption adds no additional security; anyone with access to a PEB can easily bypass its encryption and access its contents.

9.1.3 The iVotronic Serial Debugger

In order to facilitate the development of the iVotronic exploits, we created a specialized firmware with debugging support. The debugging firmware allows the exploit developer to attach a computer to the iVotronic via a serial cable connected to the iVotronic's serial port. The attached computer runs a debugger program (gdb) over the serial port. The attached debugger offers full debugging support of the iVotronic, including memory inspection and single stepping, and partial support for breakpoints.

The debugging firmware was created by attaching a debugging stub to the original firmware. The stub is provided as part of the gdb debugger. The stub can be concatenated to the original firmware and only a few pointers in the original firmware need to be changed in order to hook the stub into the firmware. No source code is needed in order to perform the modifications. Any attacker with access to a binary firmware image can create a firmware with debugging support.

9.1.4 A tool to read and write M100 PCMCIA memory cards

We developed a tool that contains python scripts to read and write PCMCIA cards and firmware update cards. The tool also sets all of the headers and CRC values appropriately. This enables us to put any malicious software or data that we want on the cards.

9.1.5 The JTAG hardware debugger

The iVotronic DRE is equipped with a JTAG connector on the circuit board. JTAG is a standard for low-level hardware debugging. We utilized this port in order to read and write directly to the flash chips without the help of the firmware. We needed this functionality mainly for two reasons. First, we wanted to validate that the firmware installed on the iVotronic was the firmware that had been placed in escrow and also that no other code was installed on the machine. Checking the version number displayed by the iVotronic would not suffice, since any malicious firmware could provide us with the correct version number. Second, we needed a way to recover from a situation where the firmware was corrupted in such a way that upgrading the firmware the usual way would not work. Since we were going to create a modified firmware for the voting machine, chances were that the machine could end up completely disabled if we made any mistakes when modifying the firmware.

In order to communicate with the onboard flash chips we modified OpenOCD,[3] which is an open source

[3] http://openocd.berlios.de/web/

JTAG tool created for ARM processors. Unfortunately, the processor used by the iVotronic is drastically different from the processors supported by OpenOCD; therefore, we had to develop support for this processor.

The M100 scanner utilizes the same processor as the iVotronic. This processor has JTAG support, but the M100 is not equipped with an easy to access JTAG port. In order to be able to read the firmware of the scanner and update it if necessary, we mounted a JTAG connector on the scanner. After dumping the content of the scanner's firmware we noticed that the firmware held in escrow was only part of the scanner's firmware. The second unknown part of the firmware was located at the top 64KB of the firmware flash. It appears to contain the QNX kernel, but we have no way to validate this.

We asked ES&S for source and binaries of this piece of the firmware, but we never received it. ES&S said that the firmware is the bootloader code that is part of the burned image on the hardware chip that is installed on the motherboard as part of the manufacturing process. They also said that it is write protected and cannot be changed or updated after installation on the motherboard.

We have no way of validating that the code located in the upper 64KB of the firmware flash contains the original QNX kernel. It is correct that the bootblock is write protected, but the protection is not fool proof. More specifically, we accidentally overwrote two bytes of this part of the firmware during our evaluation. The scanner stopped working at this point and we had to reprogram the bootblock using our earlier firmware dump.

9.1.6 A tool for extracting the QNX filesystem from the M100

The main part of the M100 firmware consists of a filesystem that is mounted as the root filesystem by the kernel. This filesystem contains a set of hardware drivers and startup scripts in addition to the main scanner application. The filesystem is a proprietary QNX filesystem specifically made for flash memory. Even though we had access to the latest QNX development environment, we were not able to extract the files contained in this filesystem. The filesystem contained in the firmware image was created with an old version of the QNX development tools, which is no longer supported by QNX.

Since we could not find any tools to access the filesystem or any description of the format of the filesystem, we had to reverse engineer the format. The content of the filesystem is compressed in order to save memory, which further complicated the reverse engineering.

We created a tool that given a firmware image will extract and decompress all the files contained in the filesystem. The tool is also able compress a file using the same compression as the filesystem, but there is no support for generating a filesystem from scratch.

We had access to the uncompressed versions of some of the files contained in the filesystem (all the COTS components and text configuration files). We validated the correctness of our dump utility by comparing the output files to the files we had access to. The file we were most interested in getting off the filesystem was the main scanner application, which we did not have a copy of. We were able to extract this application and disassemble it.

9.2 Physical Security and Security Seals

It is our understanding that in some counties in Ohio the doors on the front of the iVotronic, the Compact Flash slot in the iVotronic, and the PCMCIA card slot in the M100 are guarded by tamper-evident seals. Unfortunately, we were not given any samples of these seals; therefore, we could not determine how effective the seals are at preventing access to these components.

We also noted that none of the hardware devices contain factory installed tamper-evident seals. Neither the M100 nor the iVotronic contains factory installed tamper-evident seals. These devices are, however, shipped with a "warranty void" sticker on them, but this is just a regular sticker that can easily be removed and replaced without a trace. In addition, the sticker is not very solid and often breaks just by handling the equipment. Therefore, a broken sticker is not considered suspicious.

The lack of factory installed seals allows anyone with access to the equipment to open it and tamper with the inside. The counties could install seals on receipt of the equipment, but the equipment we received did not have any traces of county installed seals. We find it unlikely that the county officials would have removed all traces of the seals before shipping the equipment to us. It is more likely that the equipment never was sealed.

In addition to the seals, the M100 optical scanner has a number of locks that are supposed to keep intruders from getting at the scanner itself and at the PCMCIA cards. It was our experience that an inexperienced lock picker in our group was able to successfully pick the M100 lock in a matter of a few seconds, using two paper clips. After the lock was opened, the scanner could be opened by removing a few screws. The PCMCIA cards could then be removed and reinserted by removing the two small nuts and bolts that hold the protective covers over the cards. These covers are supposed to have seals on them, but the cover fixture can be removed and put back on without disturbing the seals.

The PEB communicates with the DRE and the PEB reader using an infrared link. The PEB contains a battery that is normally in the off state. In order to turn the PEB on, a magnet has to be in proximity of the bottom of the device. The PEB slot on the iVotronic contains a magnet and an infrared link for this purpose. When the iVotronic doors are sealed there is not enough space to insert a PEB in the iVotronic slot. We discovered, however, that by placing a strong magnet on the outside of the iVotronic doors when they are closed and by slipping the inside of a PEB, which is quite thin, behind the door and close to the PEB slot (but not in the slot) we were able to activate the PEB and load malicious firmware on the iVotronic. This was the same malicious firmware as is used for Scenario peb.1 in Section 9.3.1. If the doors are only sealed on the top, as we have seen in some literature, we could slip the whole PEB under the bottom of the doors and do the same thing, again using a strong magnet on the outside of the closed iVotronic door.

9.3 Successful Attack Scenarios

We implemented and tested all of the attack scenarios in this section.

9.3.1 Attack Scenario peb.1: Changing an Unattentive Voter's Vote

This scenario assumes an unattentive voter. A malicious PEB was crafted to use the PEB ballot header overflow vulnerability, discussed in Section 7.2.5, and the PEB was introduced into the system using one of the methods presented in that same section. The malicious code on the PEB contained the payload binary

as presented in Section 9.1.1.

Before the election, the PEB is inserted in an iVotronic machine. When the PEB is inserted, the exploit is automatically triggered, and, as a result, the malicious firmware is installed on the iVotronic. The malicious firmware behaves normally during the pre-election, but it starts to actively modify the election results as soon as the actual election starts. The malicious firmware uses the LAT detection vulnerability discussed in Section 7.2.18 to determine whether the iVotronic is in election mode.

The malicious firmware monitors the votes being cast and modifies the ballot to give advantage to a certain candidate. The firmware uses the payload "hooks" and links in just before the vote summary page is displayed. It steals votes at this point by assigning them to the other candidate. The modified vote shows up on both the screen and the Real-Time Audit Log (RTAL). If the voter actually checks the printed output of his/her votes and discovers that an error has been made, the malicious firmware detects that the voter recasts his/her vote, because the firmware is also hooked into the "back" button on this page, which is used to recast a vote. That is, if the voter detects the modified vote, the malicious firmware allows the voter's corrected vote to be recast and stops stealing votes for a period of time. In this way, it is less likely that someone will discover that something suspicious is going on.

It is worth noting that both the summary page and the paper trail report the modified selection, rather than the original one. From an attacker's perspective, it is better to keep the screen and paper consistent, because, if an abnormality is detected, then it is more likely to be attributed to a screen miscalibration rather than to an attack.

If one assumes that many voters do not check that their vote was properly cast, then this attack scenario can modify the results of an election, and it cannot be detected by a manual audit. The assumption that people do not check the paper trail is supported by at least two studies. In Everett's thesis[4] the author reports that over 60% of the voters she tested did not notice their votes had been changed in the review screen. It is reasonable to expect that a similar result would be obtained by changing the vote on the paper record. In another report, Selker and Cohen present a study in which errors were intentionally inserted into the VVPAT.[5] No voters reported the errors during voting, and only 8% agreed that there were errors in the VVPAT when asked.

9.3.2 Attack Scenario peb.2: Changing a Careful Voter's Vote

This scenario assumes that voters check their votes on both the screen and the printed ballot, but that they are not familiar with all of the details of how their votes are recorded on the paper audit tape. In this scenario, the iVotronic machine is compromised in the same way as described for the peb.1 scenario. However, in this case the voting process proceeds normally. That is, the voters actual choices are displayed and printed as cast by the voter.

After the voter has completed all of his/her votes, the voter has to cast and confirm his/her choices. Once the voter touches the "cast ballot" button and confirms the vote by touching the "confirm" button, the malicious firmware takes over. That is, the malicious firmware does not intercept the normal process until after the cast ballot and confirm pages have been presented to the voter.

At this point the malicious firmware changes the voter's electronic ballot, and the RTAL prints "Race S Canceled: Candidate X" and "Race S Selected: Candidate Y", where Y is the attacker's choice for Race S. The RTAL then immediately prints "Accepted," the other normal tabulation information, the bar code, and

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

[5] T. Selker and S. Cohen, *An active approach to voting verification*. May 2005 (28). – Technical report.

the time stamp. This whole printing process plus the paper scrolling up and out of site takes less than 4 seconds. The printer is also scrolling forward and back when printing this information, which makes any attempt to read what is printed even more difficult.

After carefully checking, casting, and confirming their votes, most voters do not pay attention to what is printed on the RTAL.[6] Also, unless the voter is watching closely as the RTAL is printing, he/she will not be able to see what was printed.

This version of the attack is more likely to evade detection, since the stealing happens only after the voter has confirmed his/her selection. By doing this, the modified selection is never shown on the screen and is printed on the audit log only when the paper is scrolling up, immediately before the barcode is printed.

9.3.3 Attack Scenario peb.3: Canceling the Vote of a Fleeing Voter

In this scenario, the iVotronic machine is compromised in the same way as described for the peb.1 scenario. However, this time the malicious firmware takes advantage of "fleeing" voters. These are voters that leave the voting station before having completed their voting, which is not uncommon. In Ohio the votes of fleeing voters are discarded.[7]

When a voter flees, the iVotronic makes a chirping sound after about twenty seconds, which alerts a poll worker. The poll worker has to insert a supervisor PEB in the iVotronic and follow a specific procedure to discard the ballot. For privacy reasons, the poll worker has no access to the content of the ballot.

For this scenario, the malicious firmware intercepts the call to the routine that enables the chirping sound, indicating a fleeing voter. The malicious firmware behaves differently depending on whether or not the fleeing voter cast a vote for the attacker's candidate.

Case 1: Voter voted against the attacker's candidate. If the fleeing voter did not vote for the attacker's candidate, then the malicious firmware does nothing and lets the chirping program perform as it should. In this case, the fleeing voter's ballot will be discarded, and there will be one less vote for the undesired candidate.

Case 2: Voter voted for the attacker's candidate. If the fleeing voter voted for the candidate that the attacker wants to win, then the malicious firmware completes the voting process, by faking the pressing of the "VOTE" button. This results in another vote being cast for the attacker's candidate.

This could result in a lower than average number of fleeing voters. Also, it might be possible to detect that all of the fleeing voters had voted against the attacker's candidate. Since this could possibly arouse suspicion, the firmware only completes the voting process for a certain percentage of the fleeing voter ballots that are for the attacker's candidate.

9.3.4 Attack Scenario peb.4: Canceling a Vote by Faking a Fleeing Voter

In this scenario, the iVotronic machine is compromised in the same way as described for the peb.1 scenario. However, in this case the firmware fakes a fleeing voter. If a voter does not select the candidate that the attacker wants, the malicious firmware intercepts the confirmation page's confirm function and pretends to cast the ballot: the normal "thank you" page is displayed, but nothing is printed on the audit tape. After

[6] Everett (as in n. 4); Selker and Cohen (as in n. 5).

[7] This differs from California, where a fleeing voter's ballot is cast by the poll worker.

waiting a few seconds (during which time the voter likely leaves the booth) the firmware again displays the confirmation page. After some time, the firmware calls the fleeing voter code and the machine will start chirping. A poll worker will think the voter was a fleeing voter, and, in accordance with Ohio's procedures, the ballot will be canceled.

9.3.5 Attack Scenario flash.1: iVotronic Denial-of-Service

This scenario causes an iVotronic to crash when a malicious flash card is inserted and the iVotronic attempts to read an image file from the card. A malicious flash card was crafted to take advantage of the flash card hot spot buffer overflow vulnerability, discussed in Section 7.2.6, and the flash card was introduced into the system using one of the methods presented in that same section.

In Section 7.2.6 we saw that there are two different code patterns that are used in the iVotronic to read image headers. For this attack, code that uses the first pattern for reading the image header is exploited. Recall that executing this code allows an attacker to overwrite the stack return address, but not with data under the attacker's control; therefore, a system crash is likely.

A denial-of-service attack was implemented by replacing one of the system's election image files with a file that overflows the hotspot handling function that uses the first code pattern. Because the attacker cannot control what gets written on the stack, he/she cannot introduce malicious code. However, triggering the image overflow vulnerability can cause the iVotronic to crash.

When the modified image was to be displayed, the iVotronic crashed, resulting in the expected denial-of-service.

9.3.6 Attack Scenario flash.2: Voter Confusion

In this scenario, the iVotronic machine is compromised in the same way as described for the flash.1 scenario. That is, a malicious flash card was crafted to take advantage of the flash card hot spot buffer overflow vulnerability, discussed in Section 7.2.6, and the flash card was introduced into the system using one of the methods presented in that same section. However, for this attack code that uses the second code pattern described in Section 7.2.6 is exploited. Recall that when executing this code an attacker can overwrite the stack return address with an address pointing to code of the attacker's choosing. Therefore, the possibilities for the attacker have no limitations.

To demonstrate how the election process could be subverted, one of the system's election image files was replaced with a file that overflows an image hotspot handling function, exploiting the second code pattern. The exploit crafted on the image file triggered the image overflow vulnerability and introduced code that displays an obviously wrong message on the screen to confuse the voter. Although displaying this message was basically benign, it demonstrates that one could introduce code of their choice and could subvert an election. For instance, exploits peb.1, peb.2, peb.3, and peb.4 could all be realized using this image overflow vulnerability. The attacker need only change the code introduced.

Recall that in Section7.2.6 there were ten different locations in the iVotronic's code that read image headers using the second of the two code patterns. Although the attack presented in this section was for only one of the ten occurrences of the second pattern, the other instances could be used for delivering malicious code in the same way.

9.3.7 Attack Scenario unity.1: Unrestricted Access to Unity Ballot Preparation Software

The Unity election management system is used to prepare ballots for all of the iVotronics and M100s in all of the precincts. Unfortunately, as shown in Section 7.1.9, the authentication process for the Election Data Manager (EDM), Ballot Image Manager (ESSIM), and Audit Manager components of the Unity election management system can be bypassed with a simple SQL injection. This gives the attacker unrestricted access to these components.

This scenario demonstrates an SQL injection exploit that uses a secret constant string[8] as the username parameter, and the password parameter is left blank.

We tried this for EDM, ESSIM, and for the Audit Manager. In all cases the user was logged on with no further authentication checks.

9.3.8 Attack Scenario unity.2: Compromising the Unity Election Reporting Manager

The Unity election management system is used to compile results from all of the precincts. The most common method of delivering the results is on a results PEB. Unfortunately, the Election Reporting Manager (ERM) component of Unity, which is used to compile the results, has a buffer overflow in the code that reads the results PEB.

For this scenario, a malicious PEB is crafted to take advantage of the ERM PEB reader buffer overflow, which was presented in Section 7.1.1. The malicious PEB is introduced into the system using one of the methods presented in Section 7.2.5. The malicious code on the PEB creates an account "attacker" with password "weownyou."

When the PEB is read by the ERM using the PEB reader, the data on the PEB overflows the stack of the ERM application. The return address of the current function is overwritten to point to malicious code, which is then executed. The malicious code adds the "attacker" account and control is returned to the application, which, after a few seconds delay, displays a message indicating that reading the PEB has failed. This is expected, since the malicious code in its current form does not take care to exit cleanly after it is executed.

The success of the attack is verified by logging in to the system as user "attacker" with password "weownyou." The login was successful.

9.3.9 Attack Scenario m100.1: Changing the Firmware on the M100 Scanner

The M100 optical scanner is used to count physical election ballots. The firmware on the M100 is updated by inserting a special firmware PCMCIA card. These are usually provided by the ES&S service representatives.

The development of a malicious firmware was complicated by the lack of tools required to both compile the M100 sources and build the final image. For this scenario the QNX filesystem extraction tool and the PCMCIA read/write tool described in Sections 9.1.4 and 9.1.6, respectively, were used to craft a malicious PCMCIA update card.

The malicious firmware that we developed behaves exactly like the original firmware, except that it gives all the votes to the first candidate. To accomplish this, we had to extract the binaries from the firmware installed on the M100 and patch them to steal votes.

[8] See Section 22.9.3.7 in the Annex of the private report for the actual string.

When the card is inserted in the M100 and it is powered up the card will be accepted and the CRC check will be passed, because the card was crafted with an appropriate CRC value and in the appropriate format. Next, the M100 will display a menu asking the user if he/she wants to replace the firmware. A "yes" response results in the firmware being updated.

9.3.10 Attack Scenario m100.2: M100 Denial-of-Service

The M100 optical scanner, which is used to count physical election ballots, uses ballot PCMCIA cards to get ballot information from Unity.

This scenario uses the PCMCIA read/write tool described in Section 9.1.4 to craft a malicious PCMCIA ballot card, which forces the M100 to write to an invalid address. This results in a segmentation fault. The kernel intercepts the segmentation fault signal, prints an error on the printer, and freezes the M100, asking the user to reboot.

Our effort with this vulnerability was hampered by the fact that we did not have all of the source code for the M100 firmware, even though we requested the missing firmware source numerous times. As a result, we wasted a lot of time and energy reverse-engineering the firmware. The result is that we did not have enough time to find a way to control what is written. Therefore, this scenario results in a system crash.

9.3.11 Attack Scenario virus.1: Compromising Entire Election Process with a Virus

The ES&S voting process forms a loop. That is, the Unity election management system is used to initialize Personalized Electronic Ballots (PEBs), Compact Flash cards, and PCMCIA cards. These are in turn used to initialize the iVotronic DREs and the M100 optical scanners with ballot information. The PEBs are also used to activate the DREs for voting, power the machines on, activate supervisor functions, and to transport the election results from the iVotronics to Unity. Figure 9.1 shows this flow of information. Inserting malicious code at any step in this process could result in a virus spreading to all of the other components, completely compromising the election.

This exploit uses the payload framework presented in Section 9.1.1 to implement the propagation of malicious code from an infected iVotronic DRE to another iVotronic DRE, where "infected" refers to a running malicious firmware. Similarly, the payload also implements spreading from an infected iVotronic DRE to the Unity election management system, such that subsequent PEBs generated from an infected Unity installation will propagate this payload to all iVotronic DREs in its jurisdiction. In this manner, a persistent viral infection of malicious code in an ES&S electronic voting infrastructure has been implemented.

In the case of spreading from iVotronic to iVotronic, a PEB is infected when it is inserted into an infected DRE to activate it for voting. This allows a malicious firmware to infect a master PEB used for pre-election logic and accuracy tests. Then, on election day, the master PEB can spread the infection to all machines in a polling location as they are activated.

Spreading from iVotronic to Unity is accomplished in a similar manner, except that a PEB is infected when inserted to collect votes as the terminal is closed. The PEB used for this function will subsequently be loaded into Unity. Taking advantage of scenario unity.2 in Section 9.3.8 it can be used as a vector for spreading the virus to election central. That is, when the PEB is loaded into Unity, a program will be installed to infect all future PEBs generated from that Unity installation with the malicious code to implement the previous component of this virus.

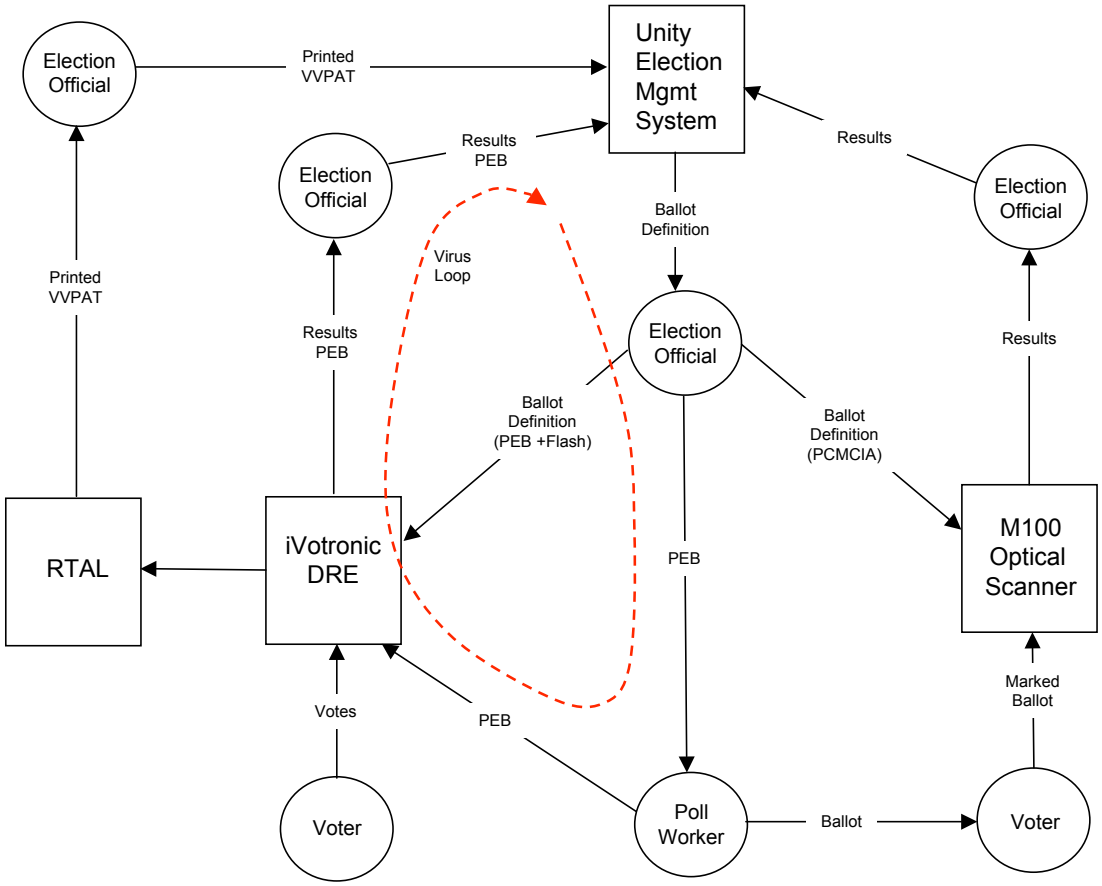

Figure 9.1: ES&S Architecture and Virus Loop

Scenario unity.2 in Section 9.3.8 could also be used during the initial testing to modify the Unity code. Again, the modified code puts a virus on all of the PEBs that are distributed to all of the precincts. These PEBs, in turn, spread the virus to all of the iVotronic DREs in all of the precincts. The result is that every iVotronic now has the malicious code that compromises the election.

9.4 Potential Attack Scenarios

9.4.1 Attack Scenario flash.3: iVotronic Exploits Using a Flash Card as Delivery Mechanism

Given more time, we could modify the payload framework discussed in Section 9.1.1 to work with the Compact Flash code vulnerability discussed in Section 7.2.6 to deliver a payload similar to those discussed in Section 7.2.5, which used a PEB as the delivery mechanism. This would give us an alternate path to introduce malicious code into the system if PEBs or PEB-like clones were not available.