Tampering with Special Purpose Trusted Computing Devices: A Case Study in Optical Scan E-Voting

Aggelos Kiayias Laurent Michel Alexander Russell Narasimha Shashidhar
Andrew See Alexander Shvartsman Seda Davtyan
{aggelos,ldm,acr,karpoor,andysee,aas,seda}@engr.uconn.edu
Voting Technology Research Center
Department of Computer Science
University of Connecticut
Storrs, CT

Abstract

Special purpose trusted computing devices are currently being deployed to offer many services for which the general purpose computing paradigm is unsuitable. The nature of the services offered by many of these devices demand high security and reliability, as well as low cost and low power consumption. Electronic Voting machines *is a canonical example of this phenomenon. With electronic voting machines currently being used in much of the United States and several other countries, there is a strong need for thorough security evaluation of these devices and the procedures in place for their use. In this work, we first put forth a general framework for special purpose trusted computing devices. We then focus on Optical Scan (OS) electronic voting technology as a specific instance of this framework. OS terminals are a popular e-voting technology with the decided advantage of a user-verified paper trail: the ballot sheets themselves. Still election results are based on machine-generated totals as well as machine-generated audit reports to validate the voting process.*

*In this paper we present a security assessment of the Diebold AccuVote Optical Scan voting terminal (*AV-OS*), a popular OS terminal currently in wide deployment anticipating the 2008 Presidential elections. The assessment is developed using exclusively reverse-engineering, without any technical specifications provided by the machine suppliers. We demonstrate a number of security issues that relate to the machine's proprietary language, called* `AccuBasic`*, that is used for reporting election results. While this language is thought to be benign, especially given that it is essentially sandboxed by the firmware to have only read access, we demonstrate that it is powerful enough to* (i) *strengthen known attacks against the* AV-OS *so that they become undetectable prior to elections (and thus significantly increasing their magnitude) or,* (ii) *to conditionally bias the election results to reach a desired outcome. Given the discovered vulnerabilities and attacks we proceed to discuss how random audits can be used to validate with high confidence that a procedure carried out by special purpose devices such as the* AV-OS *has not been manipulated. We end with a set of recommendations for the design and safe-use of OS voting systems.*

1. Introduction

A special purpose computing device is a computing system designed to be reliable for a certain specialized class of applications. This is in stark contrast to the goals of a general-purpose computer, designed to provide a broad spectrum of services without addressing specialized security concerns. The design of specialized devices, on the other hand, should make it possible to offer several services to the end-user in a more secure, reliable fashion, something that may not be as readily feasible when using a general-purpose computer. Notable examples of special purpose trusted computing devices in current use are automatic teller machines (ATMs or Bancomats) in banks, home gaming stations, and electronic voting terminals.

Given that the security concerns for such specialized devices vary from one application to the next, we first present a general architecture of such a typical system. We briefly discuss the different modules, both active and passive, that collectively form the system. We analyze the different parameters of each module from a security standpoint and illustrate the several classes of threats or attacks that can be launched against such a system.

Based on this framework we proceed to analyze a widely-used electronic voting technology, called Optical

Scan (OS) voting. Such voting terminals have been in active use in many elections in the United States. Subsequently, we focus on the proprietary language for writing software in a particular OS voting terminal; we present malware written in this language that affects the intended system operation. We then provide pointers and recommendations for safe use of such systems including random audits. The hope is that readers may garner the lessons learned from the defects of the particular terminal (*that is currently in use in many states and will be employed in the upcoming 2008 presidential elections*) and that the industry standards for the security for such systems will be improved.

Electronic Voting. E-democracy [4], the use of electronic technologies to support the democratic process, is a topic of much debate within the government, industry, and academia. Elections form the foundation of any successful democracy and safeguarding their integrity is naturally an issue of paramount importance to the electorate. Thus, a principal cause of concern is the accuracy, security, and effectiveness of the electoral apparatus used to conduct elections. After the disputed 2000 presidential election, the role of technology in the voting process has attracted an international audience. Electronic Voting Machines (EVMs) have since then been brought to the focus of attention and they were touted by many as the much needed replacement of the previous voting technology using punch cards and lever machines. The Help America Vote Act (HAVA) [5] enabled the upgrade of voting equipment nation-wide with a promised budget of $3.8 billion for election reform. As reported in [6], depending on the state, 30% to 90% of the funds that were eventually allocated would be spent on voting equipment. The effects of the upgrade are already evident, since in 2006 it was estimated that about 130 million voters would be using EVMs to cast their votes [7].

While EVMs appear to offer improved performance in terms of reducing residual vote rates, see, e. g., [8], and provide more flexible human interfaces, they also became the subject of intense scrutiny from a computer security viewpoint. Several studies [3, 9, 10, 11, 12, 1, 13, 14, 15] investigated the competence of some EVMs in use as well as performed evaluation and security assessment and returned alarming results. Evidently, there are significant design challenges to be overcome before EVMs can be considered truly satisfactory election instruments. To gain the trust of the electorate and to maintain the integrity of the electoral process the need for a thorough security evaluation of these devices and the procedures in place for their use cannot be understated.

It should be stressed that not all EVMs are "equal"; excluding minor differences there are two major types of electronic voting equipment: Direct recording electronic (DRE) machines and optical-scan (OS) machines. There is heated debate over which technology is more suitable, and at present the adoption is split between the two types of systems [7], with larger counties favoring DRE machines and smaller counties favoring OS machines. From a security assessment point of view DRE's have attracted most of the criticism [10, 12, 13, 15], while OS technology is typically touted as the safer alternative (though not without its own problems [1, 14]). Indeed, an important benefit of the optical scan technology is that it naturally yields a voter-verified paper audit trail (VVPAT)—the actual "bubble sheet" ballots marked by the voters. This differentiates OS electronic voting from DRE voting terminals (such as the Diebold AccuVote TS [10, 15] and TSx [13] terminals for example) that provide a digital interface for voting during the elections.

Contributions. We present the following contributions.

First, we present a general framework for describing the general architecture of special purpose trusted computing devices that highlights their vulnerable components (cf. Section 2); we show how OS voting terminals in general, and the AV-OS in particular, fit into this general framework.

Second, we focus on the vulnerabilities of one particular component in the architecture that deals with the critical reporting functionality of the device. For the AV-OS, the reporting functionality is based on the `AccuBasic` proprietary interpreted language that we reverse engineered based on a compiler that is publicly available. We stress that we did not have "insider access" to any of the system's components, and we did not have access to any vendor design or communication specifications.

The `AccuBasic` language is thought to be relatively benign, given that it is "sandboxed" by the firmware and has only read-only access to the sensitive memory areas of the AV-OS system. Previous works [1, 2] touched briefly on the role of this language in developing attack vectors against the AV-OS and did not utilize its full potential (from the attacker's viewpoint). The results presented in another report [14] (that was based on insider access of the actual source of the `Accubasic` interpreter), hinted to some potential issues with the language, although no concrete malware was presented.

Here, we demonstrate that by implanting "`AccuBasic` Malware" code into a terminal we can (i) strengthen the previous attacks of [1, 2] by making them *undetectable to pre-election audits*, thus substantially increasing the seriousness of the threat by such attacks, and (ii) *conditionally* bias the election results to reach a desired outcome. These results are reported in Section 3.

Finally, we deal with random audits and how they can be used to validate that a certain procedure carried out using specialized devices is not compromised. Audits are based on executing the machine operation independently on a small random sample of the device population; these results are addressed in Section 4. We finish the paper with a review of the lessons learned from our investigation.

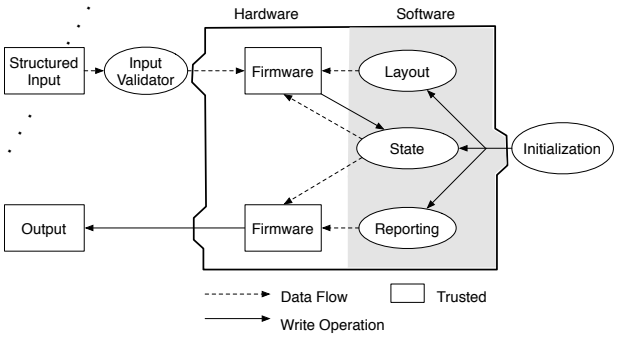

Figure 1. Architecture of a special purpose trusted computing platform.

2. Special Purpose Trusted Computing Devices

In this section we describe the general architecture that characterizes special purpose trusted computing devices. Typically, such a system is composed of both software and hardware components and the interactions between the modules are illustrated in Figure 1. From a security viewpoint, the key features are the components that are accessible or easily replaceable from outside the system, namely, the software components shown in the diagram. Note that the perspective we take in this work is more narrow that the general trusted computing platform paradigm, as e.g., in `www.trustedcomputinggroup.org`. By narrowing down the scope and focusing on simpler machines we are able to better identify the issues with the particular applications that are of concern, e.g., electronic voting.

2.1. Model Description

Structured input: The system is designed to process data in the form of a stream of input. The end-user may need to have certain privileges to place his input data on the stream to be processed and the input should be structured in a certain fashion. The conformity of the input as well as the privilege of placing something into the input stream is taken care of by the input validator:

Input Validator: Authenticating the input can be done either independently of the system, e.g., by physically restricting access or handled by the system itself, e.g., by using a PIN number or a cryptographic authentication mechanism; a combination of these approaches is also possible. In addition to authentication, the input validator must ensure that the input is properly formed. For example, it should prevent buffer-overflows from input of the wrong size. For some devices, the input mechanisms are very limited, making such validation simpler.

Output: The output of the system is controlled by the firmware which uses the reporting software component to form the output based on the internal state.

Initialization: The system needs to be initialized for use before being deployed. This may be done through communication with a central server. Initialization typically includes authentication between the system and the server, checking the integrity of the system, formating any internal storage components, and loading updated software components onto the system. Clearly the initialization stage has the potential for security vulnerabilities as it touches several parts of the system that determine its behavior. Note that initialization may not necessarily update the firmware of the system (only the software components will be changed).

Firmware: The firmware is the program that controls the system. Typically this is stored in read only memory and should be hard to replace. In our formalism, this includes any code that is "off-limits" for an attacker (as an attacker that controls the firmware can literally direct a device to do anything within its capabilities). Note that other, not necessarily trusted, software components may be loaded and executed by the firmware to materialize the final operating environment of the device and thus trusting the firmware does not trivialize the security analysis for the device. Since in our formalism the firmware represents aspects of the device that are trusted, it follows that the larger the scope of the firmware, the easier it would be to ensure reliability in an adversarial setting. It should be stressed that upgrading the firmware is a critical operation that should be strictly safeguarded[1].

State: The state includes any accumulated stored information in the system. The state can normally only be written by the firmware, and can only be read through the firmware.

Layout Description: The layout software component describes how to map the input stream into events that change the state. That is, the firmware uses the layout to interpret the input stream and decide how to update the device state.

Reporting Component: When in the proper state, or after receiving the proper input, the firmware reports the final (or current) state of the system. The reporting component describes how the state is translated into the output.

2.2. Auditing

Since failures or security compromises are likely in a widely deployed system, auditing is essential to identify failures and security flaws. In our formal model, the input stream may be stored independently of the device for future reference and auditing purposes. Given the output of

[1] Some voting terminals, e.g., the AV-TS machine, have backdoors that allow operating system upgrades through a removable memory card, thus enabling devastating attacks against the terminal, cf. [13, 15]. Still, we stress that a proper firmware alone is not enough to guarantee security.

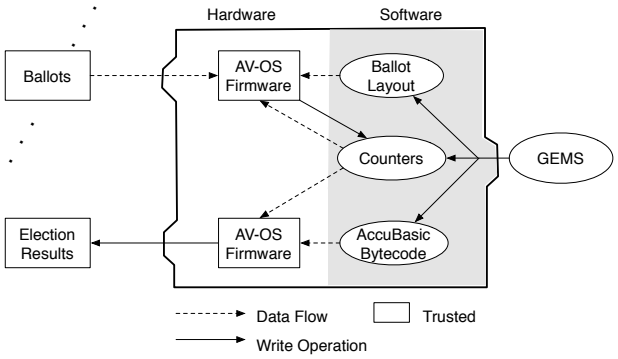

Figure 2. Architecture of the AV-OS voting terminal. Note that "GEMS" is the initialization system.

a device, it would then be possible to determine if the output is correct given the input, and also to determine if the discrepancies are statistically significant for the given application. We note that if it is not possible to store the input stream directly the voting terminal may assist an operator to produce a snapshot of an input; this is what happens for example in "voter verifiable paper audit trail" (VVPAT) voting terminals that first print a little paper receipt of the voter's choices, then require the validation of the voter and finally store the receipt for future auditing purposes.

3. Case Study: Optical Scan Electronic Voting

In this section we turn our attention to Optical Scan (OS) electronic voting, an architecture for electronic voting devices that fits into the model of special purpose trusted computing devices of Figure 1 and we discuss the security vulnerabilities of the Diebold Accu-Vote OS optical scan voting terminal (AV-OS), specifically, those related to the software components. First, Section 3.1 describes the AV-OS machine in relation to the special purpose trusted computing architecture. Section 3.2 then presents attacks against each software component based on previous work and our own findings. Methods of delivering these attacks are illustrated in Section 3.3. Section 3.4 discusses some lessons to be learned from these attacks.

3.1. The AV-OS Optical Scan Voting Terminal

The AV-OS election system consists of two components: the AccuVote Optical Scan voting terminal (the AV-OS terminal) and the ballot design and central tabulation system (GEMS, for Global Election Management System). These components have the following characteristics:

- The GEMS software is installed on a conventional PC that is equipped with a serial port and includes a ballot design system and a tabulation system.
- The specifications of an election are downloaded onto a 40-pin 128KB Epson memory card present in the AV-OS. This specification includes the *layout* of the bubble sheet and candidate names.
- The AV-OS system used in this study contained the firmware version 1.96.6 (in the form of an EPROM chip). It is equipped with an optical scanner, a paper-tape dot-matrix printer, a LCD display, a serial communication port, and telephone jacks leading to a built-in modem. It runs on a V25 CPU (an 8088 compatible processor). For election deployment the system is secured within a ballot box so that no sensitive controls or connectors are exposed to the voter.
- In addition to the firmware, the AV-OS is given a bytecode which provides functions used for *reporting* election results by printing to the audit tape.

The AV-OS terminal fits into the special purpose trusted computing device model we presented. Figure 2 shows the architecture of the AV-OS (cf. the general framework as shown in Figure 1). The firmware can be considered trusted since an attack against it would require replacing the memory chip storing the firmware. This, of course, does not imply any guarantees regarding the correctness of the firmware, but only that an attacker other than an insider is unlikely to be able to tamper with it.

During an election the input stream consists of bubble sheets in which voters have marked their votes. For our purposes, these are assumed to be valid from the design and printing process, and authenticated by the poll workers that distribute the ballot sheets during an election. The machine also has two buttons, YES and NO, which are hidden during an election and are used by poll workers during initialization and when printing the final report or audit logs.

Before an election and prior to delivering the system at a poll site, state officials load two pieces of data using the GEMS software: the ballot layout and the bytecode. Together with the counters, these are the three software components of the system. The ballot layout indicates how bubble sheet locations correspond to the counters located on the memory card. The bytecode consists of functions used, for example, to print the zero total report prior to an election and to print the election totals after an election. The initial state of the counters is zero.

3.2. Attack Vectors

In this section we present "attack vectors" against the AV-OS that tamper with the software components. We first

review existing attacks that were demonstrated against the AV-OS terminal and then we proceed to our new results.

3.2.1 Previous Attack Vectors: Initial State and Ballot Layout

In [1], Hursti demonstrates an attack in which counters are given values k and $-k \bmod 65536$. After the election, k votes will thus be transfered from one candidate to the other, and the total votes reported will remain unchanged. Specifically, an attacker must gain access to the memory card and use a card reader/writer to alter the state, after the machine has already been initialized (note that zeroing the counters is part of the initialization process). The reporting functionality is also altered in [1] to make sure that the counters are reported as zero whereas they are not. We expand on this direction substantially on our own demonstrated attack vectors. A downside of this attack (and upside from the system's viewpoint) is that the counters can be zeroed at the poll site by running a "mock auditing election" (of course it is up to the state officials to incorporate such procedure into the poll preparation procedure).

In the AV-OS terminal, the ballot layout given from the GEMS initialization system is not digitally signed and no attempt was made to authenticate the source of the layout (beyond using a proprietary integrity checking protocol that was non-cryptographic). In [2], an attack was demonstrated in which the ballot data is downloaded and modified, and a computer masquerades as a GEMS server in order to load this altered layout. The result is an attack in which a candidate's votes may be nullified by moving their bubble sheet location to an area with no bubbles, or swapped with another candidate's by swapping their bubble locations.

3.2.2 Our Results : `AccuBasic` *MalWare* for Concealing Tampering and Results Manipulation

The AV-OS bases its reporting functionality on the firmware and the `AccuBasic` bytecode that contains the formating desired for the election (cf. Figure 2). `AccuBasic` is a proprietary language that was developed by Diebold. The `AccuBasic` bytecode programs are compiled from `AccuBasic` code using a compiler that is part of the GEMS initialization system. The bytecode itself is an ASCII file and can be edited with ordinary text editors. An `AccuBasic` compiler is publicly available from [20]; we took advantage of the compiler for reverse engineering the bytecode as `AccuBasic` is not a publicly specified language. For readability, we will use the `AccuBasic` syntax to illustrate the functionality of the bytecode. `AccuBasic` is a procedural language and should be understandable for readers familiar with typical procedural programming languages.

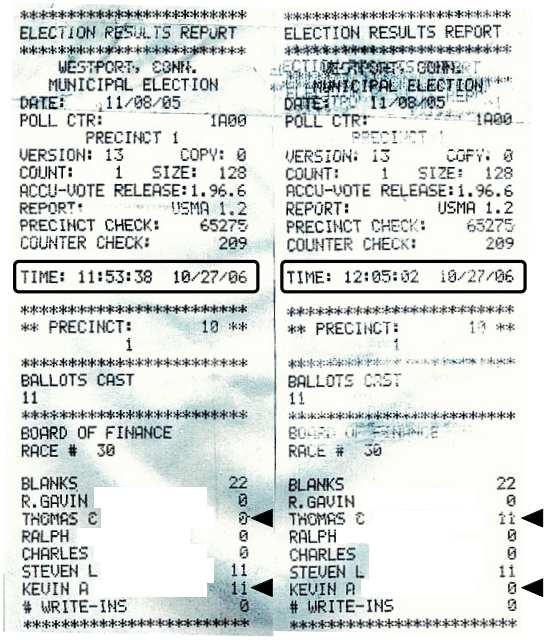

Figure 3. Election results reported by the same AV-OS terminal before (left) and after (right) the time 12:00 election time specified in the bytecode. The time-bomb was activated and the counters were unswapped to reveal the tampering.

Like the ballot layout, the bytecode is not cryptographically authenticated. In [1], special bytecode was used where counters were reported to be zero when in fact they were not, i.e., an attacker could force the machine to print only 0 in the "Zero Total Report," meant to insure that the counters are indeed 0. This could be used to hide an improper initial state as in the case of [1].

During our own experimentation we found that the bytecode language offers a wealth of functions that can be potentially exploited by an attacker. In particular, we will demonstrate a "time bomb" attack in which the bytecode checks the date and time in order to decide whether the election has begun. An attack utilizing such code can retain proper behavior in pre-election testing, in which the machine is verified by comparison with hand counted ballots, while behaving improperly during the actual election. We report on these findings below.

Concealing Tampered Initial States. The advantage of modifying the counters (e.g., by tampering with the layout or altering the initial state) is that the reported results will be compromised whether they are reported via the audit tape or electronically through the GEMS software. This means that the recording of the votes is permanently altered, at least, with respect to the counter area in the memory card. An

AV-OS terminal compromised with an attack such as those described in [1] or [2] records the actual results improperly. Still, the improper recording that is performed can be detected by pre-election testing and thus this may allow the poll-workers to isolate a tampered terminal. Still, as we will demonstrate, it is possible to make a terminal behave properly during pre-election tests.

In this section we demonstrate how a properly modified bytecode can test for the date and time and alter the reported results to conceal the tampered initial state prior to the actual election. This "double deceit" of a compromised machine — behaving improperly in the real election but properly when tested — can be achieved as follows: the report functionality of the terminal is altered so that it corrects misaligned counters or non-zero initialized counters in the event that the ballot count is too low (which would correspond to the case when the poll officials test a small batch of hand-counted votes) or when the date and time is prior to the real election time. In the case of a candidate swapping attack, the votes can be "un-swapped," and in the case of modified initial counters, the pre-loaded values can be subtracted to obtain the true value.

In other words, in standard computer security terminology, the attacker can plant a "time-bomb" in the terminal. Before the election, the program in the terminal's card inverts the swapped counters to conceal the malicious behavior (the swapping of votes). When the time of the election comes, the illicit behavior is triggered automatically. This sensitivity to time will prevent poll-workers that perform the standard test procedures from revealing that a machine is compromised prior to the election.

The rest of this section will briefly describe the bytecode alteration, illustrating the damage that can be done with this seemingly benign language used for reporting. When the AV-OS terminal is asked to print the election results, it executes a routine *z* in the bytecode located on the memory card. In its untampered state, the reporting routine loops over all the candidates and prints out the vote count for that candidate. This can be written, following `AccuBasic` syntax, roughly as follows:

```
1  PROC z
2    %c = 0
3    FOREACH candidate
4      %c = candidate.ctr[0]
5      {PRINT VOTE COUNT AS %c}
6    ENDFOREACH
7  ENDPROC
```

The variable *c* holds the vote count to be used in the code that performs the layout and printing on line 5. This vote count is initialized to zero on line 2 and takes the correct count inside the loop on line 4. The "time-bomb" attack adds a loop at the beginning of this routine to lookup the vote counts recorded for the two candidates that have been swapped. It then checks the date and time and, if the election has not yet begun, it sets the variable *c* in order to swap the candidate votes, undoing the swap that was done by changing the ballot layout. The actual `AccuBasic` code then becomes:

```
1  PROC z
2    %c = 0
3    %i = 0
4    %j = 0
5    FOREACH candidate
6      IF STRCMP(candidate.name, "A") = 0
7        %i = candidate.ctr[0]
8      ELIF STRCMP(candidate.name, "B") = 0
9        %j = candidate.ctr[0]
10     ENDIF
11   ENDFOREACH
12   FOREACH candidate
13     %c = candidate.ctr[0]
14     IF  STRCMP(DATE, "11/07/06")!=0
15       OR STRCMP(TIME, "07:00:00") <= 0
16       IF
17         STRCMP(candidate.name, "A") = 0
18           %c = %j
19         ELIF
20           STRCMP(candidate.name, "B") = 0
21           %c = %i
22         ENDIF
23     ENDIF
24     {PRINT VOTE COUNT AS %c}
25   ENDFOREACH
26 ENDPROC
```

Lines 3–4 declare the variables *i* and *j* which will hold the vote counts for the swapped candidates. Lines 5–11 look up the votes for the candidates with names "A" and "B" and store the counts in variables *i* and *j*. Lines 12–23 are the loop from the original code, but now it swaps the votes for the target candidates on lines 16–20 when either of the conditions on lines 14–15 are met. The first condition tests whether it is the election day while the second condition tests whether it is late enough in the day (i.e., polls are open and any tests must be complete). Additional conditions, such as the total number of ballots received, are also possible. Notice that resetting the counters and going back into pre-election testing mode will not help poll-workers to reveal the attack and will not invalidate the vote swapping attack.

A key insight from this attack is that the limitations imposed by `AccuBasic`, namely the read-only access to the memory card, do not prevent attacks such as these. In particular, local variables and arithmetic expressions can be used to perform this "time-bomb" attack without using any write-access to the memory card.

Altering Results. As evident from the previous sections, the `AccuBasic` election reporting functionality is powerful enough to perform various kinds of biased reporting. In particular, if the AV-OS election reporting printouts are the sole means of reporting the election results (as it is the case in fact in many jurisdictions) then one can write quite complex malicious reporting functionalities that get triggered in specific cases (when e.g., the number of votes of a certain candidate are below a certain percentage) and perform arbitrary vote transfers between the candidates. The election totals report also includes the number of blank votes in each race. A blank indicates that a voter decided not to assign their vote to any candidate. Thus, the total votes for all candidates plus the blank votes should equal the total number of ballots cast. The bytecode has access to the blank count as well, and so can also transfer votes from these blanks to a target candidate in the report, thus preserving total voter counts and possibly avoiding suspicion. The code fragment below shows part of the implementation. This fragment is located inside a loop over all the races that prints the number of blank votes followed by the election results for that race. Line 1 in the fragment tests for the target race number, here race 30. Lines 3–4 count the total number of ballots cast. Lines 6–8 then compute the number of votes that were not cast and are free to assign to our target candidate, here "RALPH". Note again that this attack is possible despite read-only access to the counters. Figures 4 and 5 illustrate the result of attack.

```
1  IF race.race_no = 30
2    %ballots = 0
3    FOREACH card
4      %ballots = %ballots + card.tot[1]
5    ENDFOREACH
6    %can_select = 4
7    %total_votes = race.tot[0]
8    %free_votes = %ballots
9      * %can_select - %total_votes
10   FOREACH candidate
11     IF STRCMP(candidate.name,
12               "RALPH") = 0
13       IF %free_votes <
14        (%ballots - candidate.ctr[0])
15         %add_to_fav = %free_votes
16       ELSE
17         %add_to_fav = %ballots
18           - candidate.ctr[0]
19       ENDIF
20     ENDIF
21   ENDFOREACH
22   PRINT "BLANKS: "(%i - %add_to_fav)
23 ELSE
24   PRINT "BLANKS: " %i
25 ENDIF
```

It is worth mentioning again that the tape is used as the primary or sole source of election results and auditing for at least some areas using this system. This avoids potential security issues associated with alternatives, such as connecting the terminals to a network or transporting the memory cards to a central location for tabulation. However, the ability to tamper with the printer from the loaded software, as described here, shows that the tape alone should not be trusted unless significant measures are taken to validate this code.

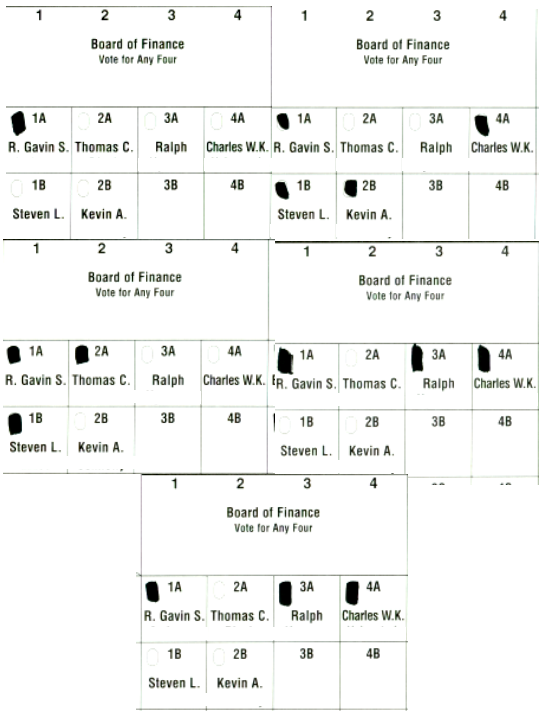

Figure 4. Five ballots cast in the test election to illustrate moving blank votes to a candidate (only the relevant part of each ballot is shown).

3.3. Loading the `AccuBasic` Malware into a terminal

The attacks described above highlight several vulnerabilities in the dynamic software portion of the voting terminal, but leaves out how an attacker may modify these components to achieve their goals. This section describes two methods to deliver the attack that have been demonstrated against the AV-OS machine: the removable memory card [1] and the serial port [2]. It should be noted that although these mechanisms are described using the AV-OS as context, similar vulnerabilities may exist in other voting terminals since removable media and communication ports are common to such devices.

Memory Card. The memory card used by the AV-OS is dis-

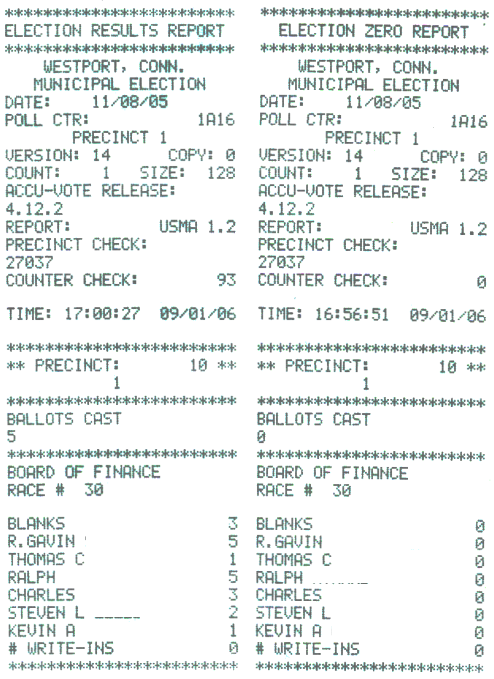

Figure 5. The election zero report and the final total report. Compare the reported tally to the actual ballots cast in Figure 4.

continued in the market, though readers/writers do exist and are obtainable[2]. In [1], attacks against the initial state and the bytecode are demonstrated using such a reader/writer, though clearly attacks against the layout data would be similarly possible. These attacks demonstrated that no check was made by the terminal to ensure that the card was not tampered with or otherwise corrupted while the machine was powered down. This represents a significant concern because of the potential for tampering, but also because of potential failures in the card such as bit errors or a failed battery (the memory card is battery powered).

Serial Port. The AV-OS machine contains a phone line and a serial port which can be used for loading election data from GEMS and for sending results back to GEMS. These ports can also be used to obtain a dump of the memory card, presumably for debugging puposes. The report in [2] demonstrated a reverse engineering of the protocol being used, and a method of forging a communication given a memory card dump. In essence, the AV-OS machine itself was used as a reader/writer. Accessing the terminal in this way gives an attacker control over the layout and reporting components, though not direct access to the counters.

[2]We actually rented on a weekly basis such a card reader/writer. It was not possible to find one for sale.

3.4. Lessons Learned

In this paper we have shown that the AV-OS system has several serious vulnerabilities that can be used to alter election results. Here we outline some particular shortcomings and identify aspects that should be addressed to obtain a robust OS voting terminal. We also note that a number of previous works ([16, 17, 18] to name a few) addressed various aspects of designing improved voting system machines.

Executable Code. Our attacks, and that of [1, 14] show that *any* executables loaded onto a voting terminal must be treated with great care, even with such limited functionality as the bytecode used in the AV-OS terminal. In particular, any executables should be digitally signed to authenticate the origin of the program and the source of out of which such executables originate must be cryptographically authenticated. Note that digital signatures come with their own concerns, e.g., key-management issues that they will have to be carefully addressed.

Authentication. For a relatively simple system such as the AV-OS system that uses only serial line or telephone for communication, a cryptographic authentication mechanism may have seemed superfluous when it was being designed. However, as it was shown, it is possible to deliver `AccuBasic` malware into the AV-OS having limited physical access to the machine, using only standard hardware, and it takes only minutes to do so. Furthermore, this could be carried out at any time between the original initialization and election day. Therefore, there are ample opportunities for an attacker to gain access to the machines. The lesson here is that any voting terminal should authenticate the sender of election data, including both the layout and reporting components.

Removable Hardware. Voting terminals should perform a cryptographic integrity check on the contents of removable devices if such contents are persistent; in the AV-OS such checks are missing. In such a situation, any removable device (e.g., a memory card) must be sealed and any removed device should be considered compromised.

Random Audits. Post-election random audits of the voting machines coupled with manual-counts would help identify faulty or compromised voting machines. In Section 4, we discuss the degree of auditing that needs to be performed in order to achieve an acceptable level of confidence in the election results.

Finally we note that without due electronic security, cryptographic integrity checks, and authentication, the only remaining defense is to impose strict control of physical custody of the voting terminals and associated election management systems.

4. Audits

In this section we discuss how audits can be used to test the integrity of a procedure that is carried out by a special purpose trusted computing device. The fundamental assumption here is that the real input stream is stored and is available for auditing purposes. The principal issue we resolve in this section is the following: suppose that a procedure was carried out by a number of devices that could have been tampered with individually. Given that the input stream is stored for each one, for how many devices the calculation should be independently repeated and compared to the "machine counts" to have a reasonable confidence that no machine tampering occurred?

It should be stressed that a random sampling plays a crucial role for an unbiased audit report. In [19] a simple method for sampling precincts in an observable way is presented. Here we give some tradeoff between the number of machines to be audited and a level of confidence to find at least one compromised machine.

First consider the case in which compromised machines are sampled with replacement. In this case, we can compute the number of machines that must be audited for a given level of confidence that at least one compromised machine is found. Table 1 shows the level number of machines that should be audited for a probability that at least one such machine was found and a given fraction of machines that are compromised. For example, if 95% confidence is desired and 10% of the machines are compromised, then 28 machines should be audited in order to be 95% sure that one such machine was found. This is computed as

$$s = \frac{\log(1-c)}{\log(1-f)}$$

where s is the required sample size, c is the level of confidence and f is the fraction of machines that are compromised. This table is valid for any number of total machines and represents a "safe" bound on the sample size.

Next, consider the case in which the number of machines is known and we wish to compute the probability that we have found at least one compromised machine. Table 2 shows, for the case of 800 machines, the probability that at least one compromised machine is found for a given sample size and compromised ratio. For example, if 10% of the machines (80 in this case) are compromised, and a sample size of 20 is used, then we can be 88% sure that our sample will include a compromised machine. This is computed as

$$c = 1 - \frac{\binom{N(1-f)}{s}}{\binom{N}{s}}$$

where c is the confidence that a compromised machine will be found, N is the total number of voting machines, f is the fraction of machines that are compromised, and s is the sample size.

Table 1. Number of machines that need to be audited to achieve desired level of confidence that at least one compromised machine was found.

Machines compromised	Confidence				
	90.0%	95.0%	97.0%	99.0%	99.5%
5.0%	45	58	68	90	103
7.5%	30	38	45	59	68
10.0%	22	28	33	44	50
12.5%	17	22	26	34	40
15.0%	14	18	22	28	33
17.5%	12	16	18	24	28
20.0%	10	13	16	21	24

Table 2. Probability of finding at least one compromised machine given a known number of machines (800 in this case).

Sample size	Fraction of machines compromised					
	5.0%	7.5%	10.0%	15.0%	17.5%	20.0%
10	0.40	0.54	0.65	0.81	0.86	0.89
15	0.54	0.69	0.80	0.92	0.95	0.97
20	0.67	0.79	0.88	0.96	0.98	0.99
25	0.73	0.86	0.93	0.98	0.99	0.99
30	0.79	0.91	0.96	0.99	0.99	1.00
35	0.84	0.94	0.98	0.99	1.00	1.00
40	0.88	0.96	0.99	0.99	1.00	1.00

These tables show that auditing is feasible, even for a high level of confidence.

5. Conclusion

In this paper we presented an architectural model for special purpose trusted devices that models well electronic voting systems. We then presented a case study of the AVOS system including an analysis of recent findings from the literature and our own experimentation. We analyzed the proprietary `AccuBasic` language that is used by the AVOS and we presented malware written in this language that can be used to either strengthen previously known attacks or make them undetectable by pre-election tests. We also presented `AccuBasic` malware that biases the reporting functionality and misrepresents the counts. Next, we discussed how to perform audits based on random sampling in large scale deployments of trusted devices to ensure that no devices have been tampered. Finally, we discussed the lessons to be learned from these investigations.

It is clear from this work that security issues exist in the

AV-OS system, and any system with a similar architecture will posses similar flaws unless proper security procedures are put in place. Of particular interest is the flexibility of attacks through the bytecode, despite having no ability to write to the internal storage. The ability to modify the layout and reporting functionalities is essential to make such a system flexible enough to be practical. However, in addition to authentication mechanisms, verification that these components are behaving properly should be a mandatory requirement.

Acknowledgement. The authors thank Michael Korman and David Walluck for contributing in the AV-OS analysis at an earlier stage of the present research effort.

References

[1] Harri Hursti, Critical Security Issues with Diebold Optical Scan Design, Black Box Voting Project, July 4, 2005
http://www.blackboxvoting.org/BBVreport.pdf

[2] Aggelos Kiayias, Laurent Michel, Alexander Russell, Narasimha Shashidhar, Andrew See, and Alexander A. Shvartsman An Authentication and Ballot Layout Attack against an Optical Scan Voting Terminal 2007 USENIX/ACCURATE Electronic Voting Technology Workshop (EVT '07)August 6, 2007, Boston, MA

[3] Brennen Center Task Force on Voting System Security. *The machinery of democracy: Protecting elections in an electronic world, 2005.* Lawrence Norden, Chair. Brennen Center for Justice, NYU School of Law. http://www.brennancenter.org

[4] Dimitris Gritzalis, editor. *Secure Electronic Voting.* Springer-Verlag, Berlin, Germany. Jan 1, 2003.

[5] Help America Vote Act (HAVA), http://www.fec.gov/hava/law_ext.txt

[6] National Association of Secretaries of the State, NASS Survey summary and highlights, How States Are Spending Federal Election Reform Dollars, November 24, 2004.

[7] Election Data Services Inc. 2006 Voting Equipment Study, February 6, 2006.
http://www.electiondataservices.com/EDSInc_VEStudy2006.pdf

[8] Caltech Press Release, Caltech-MIT Team Finds 35% Improvement in Florida's Voting Technology, September 19, 2002.
http://pr.caltech.edu/media/Press_Releases/PR12284.html

[9] Jonathan Bannet, David W. Price, Algis Rudys, Justin Singer, Dan S. Wallach: Hack-a-Vote: Security Issues with Electronic Voting Systems. IEEE Security & Privacy 2(1): 32-37 (2004)

[10] Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin and Dan S. Wallach, Analysis of an Electronic Voting System, IEEE Symposium on Security and Privacy 2004, IEEE Computer Society Press, 2004.

[11] Poorvi L. Vora, Ben Adida, Ren Bucholz, David Chaum, David L. Dill, David Jefferson, Douglas W. Jones, William Lattin, Aviel D. Rubin, Michael I. Shamos, Moti Yung: Evaluation of voting systems. Commun. ACM 47(11): 144 (2004)

[12] RABA Innovative Solution Cell. Trusted agent report Diebold AccuVote-TS voting system, January 2004.

[13] Harri Hursti, Diebold TSx Evaluation, Black Box Voting Project, May 11, 2006
http://www.blackboxvoting.org/BBVtsxstudy.pdf

[14] David Wagner, David Jefferson and Matt Bishop, Security Analysis of the Diebold AccuBasic Interpreter, Voting Systems Technology Assessment Advisory Board, University of California, Berkeley, February 14, 2006.

[15] Ariel J. Feldman, J. Alex Halderman, and Edward W. Felten, Security Analysis of the Diebold AccuVote-TS Voting Machine, September 13, 2006
http://itpolicy.princeton.edu/voting

[16] David Chaum, Peter Y. A. Ryan, Steve A. Schneider, A Practical Voter-Verifiable Election Scheme. ESORICS 2005, pp. 118-139.

[17] Rebecca Mercuri, A Better Ballot Box?, IEEE Spectrum, Volume 39, Number 10, October 2002.

[18] D. Molnar, T. Kohno, N. Sastry, and D. Wagner, Tamper-Evident, History-Independent, Subliminal-Free Data Structures on PROM Storage -or- How to Store Ballots on a Voting Machine. Extended abstract, in IEEE Security and Privacy, 2006.

[19] Arel Cordero, David Wagner, and David Dill. The Role of Dice in Election Audits – Extended Abstract, IAVoSS Workshop On Trustworthy Elections (WOTE 2006), June 29, 2006

[20] Black Box Voting. http:/www.blackboxvoting.org.