Software and Security Review for Elections Systems and Software iVotronic Firmware Versions 9.1.8.0 and 9.2.0.0

**Alec Yasinsac, Breno de Medeiros,
David Gainey, John Kerski, Zachary Hartley,
Ryan Walega, and Michael Gerke**

August 14, 2007

**Security & Assurance in Information Technology Laboratory
Florida State University
Tallahassee, Florida 32306-4530**

Table of Contents

Software and Security Re-review for Elections Systems and Software iVotronic Firmware Versions 9.1.8.0 and 9.2.0.0

1. Executive Summary

The growing use of electronic voting systems has led to a corresponding increase in voting system software review frequency. This report details a software review conducted for Florida Department of State (FLDoS) as part of their voting system state certification process. This review is a follow-on to an earlier analysis conducted during an election audit.

1.1. Background

As part of their audit of the 2006 Florida Congressional District 13 (CD13) Election Audit, FLDoS commissioned the Florida State University Security and Assurance in Information Technology (SAIT) Laboratory to review the voting system firmware (iVotronic 8.0.1.2) used in that election. The resulting report [1] (hereafter referred to as "The SAIT Report" without citation) found that the firmware did not cause or contribute to the high undervote in that election, but also detailed a variety of flaws, issues, and concerns in the examined software.

Subsequently, Electronic Systems and Software (ES&S) submitted two new versions of the examined software (iVotronic 9.1.8.0 and 9.2.0.0) for certification. The former is a maintenance upgrade to 8.0.1.2 (the version that was reviewed in the SAIT Report) and is presently submitted to FLDoS for certification. The latter is presently submitted for federal certification with subsequent submission to FLDoS expected. Correspondingly, FLDoS asked SAIT Laboratory to examine these later versions to determine if the flaws identified in the SAIT Report were corrected in them.

1.2. Analysis Scope

This review is conducted to analyze the two given firmware versions to determine if previously identified software flaws are corrected. It does not reflect a comprehensive review, rather considers whether known flaws were removed. Specifically, as specified in the Statement of Work:

> "*...the purpose of this review is exclusively to determine if flaws identified in the SAIT Report of Feb 23, 2007 are repaired in the subsequent software version that is presently submitted to the FLDoS certification process.*"

Thus, we limited our focus to flaws identified in the SAIT Report, if any repairs corrected those flaws, or if repair efforts may have injected regression faults into the code.

1.3. Findings Summary

Many issues identified in the original SAIT Report [1] are fixed or mitigated in the two evaluated iVotronic firmware versions. The three most prominent flaws: (1) Buffer overflow when reading from the Personal Electronic Ballot (PEB), (2) Zero PEB, and (3) Password weaknesses have all been addressed, the latter two largely fixed and with significantly more limited impact of the former. The code base enjoyed several architectural upgrades including implementation of encryption for data on the iVotronic removable media known as the PEB.

While we find that the buffer overflow during PEB read operations is corrected, other buffer overflow flaws still exist. We also determined that although the encryption employed is strong,

the key management approaches, particularly in version 9.1.8.0, are fundamentally flawed. When combined with the legacy buffer overflow flaws, this regression fault leaves both versions vulnerable to a malware injection attack that can only be mitigated by rigorously employed media protection procedures. We detail our findings on the key management approaches in Section 4.2 below .

As the version that is submitted to Florida Department of State for certification and as the maintenance upgrade product to version 8.0.1.2, version 9.1.8.0 is imminently important to Florida. We find that while this version corrects most major reported flaws, flaws remain that dictate that elections officials employing these terminals must exercise rigorous media and terminal protection procedures that go well beyond those in common practice.

2. Project Background and Investigation Scope

The growing use of electronic voting systems has led to a corresponding increase in voting system software review frequency. This report details a software review conducted for Florida Department of State (FLDoS) as part of their voting system state certification process. This review is a follow-on to an earlier analysis conducted during an election audit [1].

2.1. Background

As part of their audit of the 2006 Florida Congressional District 13 Election Audit [2], FLDoS commissioned the Florida State University Security and Assurance in Information Technology (SAIT) Laboratory to review the voting system firmware (iVotronic 8.0.1.2) used in that election. The resulting report [1] found that the firmware did not cause or contribute to the high undervote in that election, but also detailed a variety of software flaws, issues, and concerns in the examined software.

Subsequently, Electronic Systems and Software (ES&S) submitted two new versions of the examined software (iVotronic 9.1.8.0 and 9.2.0.0) for certification. The former is a maintenance upgrade to 8.0.1.2 (the version that was reviewed in the SAIT Report) and is submitted to FLDoS for certification. The latter is presently submitted for federal certification with subsequent submission to FLDoS expected. Correspondingly, FLDoS asked SAIT Laboratory to examine these later versions to determine if the flaws identified in the earlier version were corrected in the presently submitted versions.

2.2. Analysis Scope

This review was conducted to analyze iVotronic firmware versions 9.1.8.0 and 9.2.0.0 to determine if identified flaws in version 8.0.1.2 are corrected, so we limited our focus to flaws identified in the SAIT Report. We acknowledge using a liberal interpretation of the term “flaw” and we investigated items as though they were flaws even if they were only mentioned in passing, thus we investigated items some people may not consider flaws at all.

Using this approach, as we examined each reported flaw, we generally encountered one of the following distinct situations.

1. The code in the subsequent version was identical, or nearly identical, to the earlier code. We examined architectural changes to determine if they may have impacted these flaws.
2. The newer version reflected changes that directly addressed the reported flaws while retaining code functionality. In this case we attempted to determine if the flaw was still

present in any form or if the detected code changes injected other [regression] faults into the system.

3. The functionality related to the reported flaw was fundamentally changed in the subsequent version. Here, we also attempted to determine if the flaw was still present in any form or if the corresponding code changes may have injected regression faults into the system.
4. The reported flaw reflected an abstraction that is not isolated to a data item or code segment. For example, the SAIT Report identified a concern that the code contained widespread memory overrun flaws. We investigated the specific memory overrun flaws identified in the SAIT Report and also considered the broad question of whether any new code may have injected new memory overrun flaws.
5. The functionality related to the reported flaw no longer exists. We report these flaws as "fixed".
6. We judged that the investigated item was not a flaw or that it was not relevant to this analysis.

We also note that we include our analysis of flaws detailed in the SAIT Report private appendices in the public portion of this report. It is clear that flaws that were non-pertinent in the original study are pertinent in this re-review; that is, the purpose of this review is to determine if flaws detected in the original report still exist. Thus, we report on *all* SAIT Report flaws in the public part of this report. We continue to protect proprietary information and refrain from including unnecessary details in the public report. We confirmed this approach with the vendor and FLDoS during our joint conference call on July 5th, 2007.

2.3. Out of Scope Activities

We limited our activity to the task of determining if known flaws have been removed from the two given software versions. For clarity, in this section we distinguish our work from other efforts and techniques. We caution readers that they should consider our results only in this limited context. This section may help the reader understand the nature of the limitations.

We first note that this analysis is a re-review, not an extension, of the earlier analysis that resulted in the SAIT Report. Thus, essentially all disclaimers and qualifications given in the SAIT Report also apply to this work.

2.3.1. Comprehensive Code Review

Many have proposed that voting systems should be subject to comprehensive code review. California recently conducted a one-time project to conduct such a review[1]. California also previously commissioned a comprehensive review for one limited-functionality electronic voting software component [3].

In this project, we did *not* conduct a comprehensive review. Our results only reflect our limited investigation into previously identified software flaws.

Moreover, the limited set of flaws that we investigate were *not* identified during a comprehensive review. Thus, even if all of the flaws that we investigate were removed, it would

[1] http://www.sos.ca.gov/elections/elections_vsr.htm

not be considered a comprehensive security process. Our analysis only considers if flaws identified during a limited investigation were removed from the two analyzed firmware versions.

2.3.2. Third Party Software

Any non-trivial commercial software employs third party components to address general purpose functionality or specialized recurring functions. Flaws that exist in employed third party software become part of the employing system and any comprehensive review must evaluate all third party components. We did *not* evaluate any third party components of the target software for flaws, as this was beyond our scope.

2.3.3. Red Teaming

Another popular computer system vulnerability analysis approach is Red Teaming, also known as "Penetration Testing", where experts that are given access to the complete system seek to find and exploit technical or human vulnerability. Our team conducted static code review and did not conduct Red Team analysis or Penetration Testing.

2.3.4. Dynamic Testing

Dynamic testing involves interactive code analysis, input/output testing, and stepwise system execution and analysis. This technique requires that testers extensively exercise the tested code in a complete execution environment. During this project, we did not build executable modules from the source code and we did not conduct dynamic testing.

2.3.5. Verified Build

Depending on the analysis goal, it may be important in a software analysis to ensure that the source code matches a corresponding executable module. In our case, we were not asked to determine if the source code matched any executable module, nor did we verifiably produce an executable module for matching. In fact, we did not produce any executables, as this was beyond the project scope. Our exclusive goal was static software review to determine if a set of known flaws are present in the two analyzed firmware versions.

We further note that the source code that we evaluated was delivered to us directly from the vendor and we did not confirm that the source matched either the version submitted to FlDoS or for federal certification.

3. Investigative Process

The investigators conducted the software analysis in SAIT Laboratory on the Florida State University campus. We began by reviewing the SAIT Report [1] to identify applicable flaws, and subsequently received a change summary from the vendor that addressed many of these flaws. The team partitioned the identified flaws and collaboratively investigated the code based on flaw assignments. Team members referred extensively to the SAIT Report and vendor comments during the code analysis.

Several investigators utilized automated tools to facilitate the code review. These tools identified questionable or suspicious code properties and the investigators followed these leads with hands on code review to determine if the identified items constituted flaws.

3.1. The Senior Investigators

We point out that while SAIT Laboratory conducted the original review that resulted in the SAIT Report and SAIT Laboratory is also conducting this re-review, many of the original investigators were unable to participate in this portion of the project. We added five investigators to the team to work with two returning senior investigators to conduct the bulk of the analysis. Two original SAIT Report team members reviewed and commented on a draft of this report, as is noted in the acknowledgements section below. The senior investigators are:

Breno de Medeiros is a Florida State University Assistant Professor of Computer Science. He is an information security expert with extensive software experience.

Alec Yasinsac is a Florida State University Associate Professor of Computer Science, a co-founder and co-Director of SAIT Laboratory, and is the lead Principal Investigator on this project.

3.2. The Investigators

David Gainey is a computer science graduate student and a member of SAIT Laboratory at Florida State University and he also works for the Florida State University Office of Technology Integration as a computer specialist.

Michael Gerke is a computer science graduate student and a member of SAIT Laboratory at Florida State University. He is presently employed by the Florida Department of State, Division of Elections.

Zachary Hartley graduated from FSU with a Master's in Computer Science degree in April 2007. During the bulk of this study, he was employed by the Florida Department of State, Division of Elections.

John Kerski is a computer science graduate student and a member of SAIT Laboratory at Florida State University.

Ryan Walega is a computer science graduate student and a member of SAIT Laboratory at Florida State University.

3.3. External Communications

The SAIT Team was pleased to maintain cordial and productive communications with the vendor, receiving technical documents, change summaries, and conducting conference calls with their engineers on several occasions. These communications facilitated the review.

3.4. Terminology

3.4.1. SAIT Team.

This phrase refers to the investigators organized for this project by SAIT Laboratory. The team members are identified in Sections 3.1 and 3.2 above.

3.4.2. Flaw/Fault/Vulnerability/Threat.

In this report's context, these four terms are closely related, distinguished only by subtleties and nuance. A software *flaw* is a defect that may have a negative impact on normal program operation. We use the terms *flaw* and *fault* interchangeably. Software vulnerability is a code state or status that may lead to improper program operation. Thus, a flaw may introduce vulnerability.

The term *threat* refers to a potential occurrence that exploits a vulnerability and is sometimes loosely used to refer to the threat manifestation.

3.4.3. Adversary/Attacker/Intruder.

We use these terms interchangeably to refer to any malicious party that may try to manipulate a voting system. Our analysis focuses on two attacker categories, based on their role in the voting process. The first prospective attacker class is *Voter*. Three important voter properties are: (1) there are many voters; (2) they are untrusted; and (3) they have very limited access to the system.

The second major prospective attacker category is *poll worker*. There are fewer poll workers than voters, but they have greater system access in terms of time and capability.

Elections officials and voting system vendors represent two other recognized potential attacker categories. However, we consider attacks by these parties to be outside the scope of this review.

3.4.4. Adversarial Skill.

In this report we may refer to resources required by an adversary, one of which might be technical sophistication. When we make such statements regarding requisite *attacker skill*, we generally consider the first adversary to develop the attack, not necessarily the individuals carrying out the attack. In some cases, the individuals that carry out the attacks would not need to have the same knowledge and skill level as the individuals that develop the attacks.

3.4.5. Fixed.

A flaw is *fixed* if the corresponding vulnerability is no longer present in the source code. We emphasize that it is impossible to identify all flaws, so any statement in this report that an item is fixed simply reflects our best professional opinions.

3.4.6. Benign.

A flaw is *benign* if its technical effect does not interfere with the intended system operation. Note that this is our technical assessment based on the extent of this study and the assumptions stated herein. Readers should consider all reported and potential flaws with a broader eye for system impact, in particular taking into account the specific policies and procedures under which the systems will be used.

3.4.7. Regression Fault.

Broadly speaking, a *regression fault* or flaw is a flaw that is injected into code in the course of software repair. While the term may also be used to refer to the narrow set of previously corrected flaws that may reappear during software maintenance, we accept and apply the more broad definition and seek to identify any error that may be injected during intended repair efforts.

3.4.8. Malware.

Short for "malicious software," we use this term to refer to any code designed to corrupt normal system operation. In order to corrupt a system, malware must be injected into the computer's execution sequence, for example through a buffer overflow exploit, a virus attack, or other malicious activity. Malware injection is a common computer attacker sub-goal.

3.4.9. Buffer Overflow/Memory Overrun.

These two terms are technically equivalent, with both referring to a program operation where data written into one defined memory location extends beyond its expected range and overwrites

another defined data area. For clarity, we use the term *buffer overflow* to refer to maliciously engineered memory overruns that allow the attacker to arbitrarily transfer program control. Conversely, we use the term *memory overrun* when referring to the large class of overrun flaws.

3.4.10. Sleepover.

Occasionally elections officials allow equipment such as terminals and other elections equipment to be issued to poll workers, or stored in areas where poll workers have access, the evening before an election. This process is sometimes necessary to allow remote polling places to open on time, but it also may allow poll workers to have a significant period of unsupervised access to the issued equipment.

3.4.11. Defense in Depth.

With foundations in military terminology, this security principle dictates implementing redundant security mechanisms that provide layered protection. Systems designed with *defense in depth* reflect positive robustness properties because compromise of any single mechanism does not cause system compromise or failure. Particularly critical or exposed systems or components may employ multiple security layers.

4. Findings

It is important to note that many of the flaws identified in the iVotronic 8.0.1.2 firmware related to text ballot management and that the two firmware versions under evaluation (9.1.8.0 and 9.2.0.0) no longer utilize text ballots, rather transitioned to bit-mapped ballot representation. Thus, many flaws that were present in the 8.0.1.2 version are naturally obviated by this transition. Where appropriate, we consider whether the corresponding bit-mapped management functions display the same or similar flaw characteristics as those identified in the text ballot software.

We detail our findings regarding remaining flaws in the subsections below. Again, we do not claim that we detected all flaws that remain, only that we inspected the two code versions provided and make our best professional analysis regarding the status of reported flaws.

Also note that we are analyzing two code versions against the given flaws. Our comments below apply to both versions, except where we indicate otherwise.

4.1. Memory Overrun and Buffer Overflow Vulnerability Remains (#26)

The SAIT Report detailed buffer overflow threats that could lead to a virus attack in the iVotronic 8.0.1.2 firmware version. In versions 9.1.8.0 and 9.2.0.0, we detect two code improvements that could prevent the reported buffer overflow flaws from being exploited. First, access to the vulnerable variable length string identified as the buffer overflow threat is now controlled to prevent the previous vulnerability (see Section 5.6). Second, in a change that addresses the SAIT Report's noted architectural issue of misplaced trust, in these firmware versions data is encrypted on the PEB. Addressing both the code flaw itself and the architectural issue reflects defense in depth and offers the potential to increase confidence in PEB data integrity. Nonetheless, significant weaknesses remain.

4.1.1. Architectural Weaknesses and Pervasive Flaws (#27, 28, 29, 32, 33, 34, 35, 36)

The SAIT Report noted many routine coding flaws within the 8.0.1.2 code, most based on misplaced trust in PEB data. Additionally, the SAIT Report encouraged defensive coding and

rigorous code analysis to provide defense in depth by eliminating as many flaws as possible. Our results in this section reinforce the need to follow the defense in depth principle.

The vendor clearly attempted to improve architectural trust issue by encrypting the PEB. We analyzed the encryption process that they employed, the offending Variable Length Record (VLR) changes, and other potential code faults to determine the level of protection achieved.

Our analysis revealed that while the encryption mechanism itself is sound, the key management strategy exposes the data to straightforward compromise as we describe in Section 4.2 below. Thus, any remaining buffer overflow flaws on the PEB may be exploitable, and we identified several such flaws.

Moreover, we identified many other routine coding flaws that are common in software of this vintage. Many of these minor issues were also reported in the SAIT Report.

4.1.2. Unsafe Operations (#39, 40, 41, 42, 43, 44)

As noted in the SAIT Report, the buffer overflow vulnerability could only be exploited when executing unsafe operations, that is, operations that referenced (read or accessed) the potentially corrupted PEB data. While many 8.0.1.2 operations were safe and their execution would not trigger malware injection even if used with a corrupted PEB, many operations were also unsafe.

We analyzed the 9.1.8.0 and 9.2.0.0 operations and found that many operations are unsafe, but in a slightly different way. That is, if an adversary can acquire a PEB, defeat the PEB encryption, copy malicious data onto the PEB, deliver the malicious PEB data into the terminal memory, and subsequently trigger an unsafe operation, the operation can inject malware into the terminal. The technical exploits in these versions are only slightly more difficult than that described in the SAIT Report even in the face of the PEB encryption protections, though the attack paths are more limited.

A difference in the remaining flaws, and an overall improvement in these versions is that it appears that virus propagation is more limited than that noted in the SAIT Report. That is, we do not find any buffer overflow or other flaws associated with reading the PEB, because the length of the record read from the PEB is checked before it is stored in the terminal's memory (see Section 5.6). Thus, if the proper firmware and properly constructed ballot definition files are loaded when the polls open, we do not find any unsafe operations regardless of the data on the activating PEB. This eliminates the class of attacks noted in the SAIT Report where a malicious poll worker could inject malware through a rogue PEB after an election is open.

Thus, election preparation and poll opening are now the only known points of attack for PEB-related malware insertion.

Conversely, the length of read actions from fields within the record copied into terminal memory from the PEB are not universally checked. It is these actions that may trigger a buffer overflow. The buffer overflow potential exists only when accessing fields that were previously loaded into the terminal's internal memory at or before poll opening.

This means that in order to exploit these buffer overflow flaws, the attacker must either inject malicious data onto the Master PEB (i.e. the PEB used at each precinct to open the polls) or into a voting terminal memory before the polls open. Possible attack avenues include attempting to:

(1) Compromise the ballot creation process to infect Master PEBs when they are created

(2) Modify Master PEBs after they are created but before they leave the Supervisor of Elections (SOE) office

(3) Modify Master PEB(s) after they leave the SOE's office, but before poll opening.

(4) Inject malicious software into a terminal during sleepover through a rogue PEB.

We consider the first two of these to be insider attacks; that is, they must be carried out by someone that has explicit trust within an SOE office, where rigorous security and audit procedures can be applied.

Conversely, depending on pre-election procedures, the third option may provide less-trusted poll workers access that could allow malicious PEB manipulation. If Master PEBs are distributed to poll workers for sleepover without strong tamper protection, the poll worker(s) could write data onto the PEB that would inject malicious software at poll opening.

In the fourth option, a malicious poll worker could prepare a rogue PEB that they would use to open the polls and inject malware into memory on their sleepover terminal. They could then execute an unsafe operation to inject the inserted malware into the execution sequence. The malware must then reset the terminal mode so the terminal responds as expected at poll opening on election day, which is technically straightforward.

In spite of the encryption and other improvements in these two code versions, creating the described malware and installing it on a rogue PEB for use during terminal sleepover requires only code-base understanding and strong computer skills, but is otherwise uncomplicated. Of course it also requires the malicious poll worker to acquire a working PEB.

We see the fourth option as the most dangerous because a corrupted sleepover terminal may infect the Master PEB at poll opening. The infected Master PEB could correspondingly infect any subsequently opened terminal(s). Thus, a poll worker with a sleepover terminal, but without direct access to any official PEB, could inject malware that could infect other, or even all, terminals in that polling place.

If terminals are not uniformly assigned to the same precinct for each election, it is possible that an infected terminal that is assigned to a different precinct in subsequent elections could spread the malware throughout its new precinct, possibly propagating throughout the county several election cycles after its first infection.

4.2. Regression Faults in Cryptography Implementation (#27)

In the iVotronic 8.0.1.2 firmware, PEB data was neither encrypted nor authenticated. In versions 9.1.8.0 and 9.2.0.0 that we analyzed, the ballot definition file on the PEB is encrypted using the well-known, strong encryption algorithm known as Blowfish[2]. If properly implemented, this could mitigate many problems identified in the SAIT Report.

All encryption use requires a careful key management strategy. We specifically note that for symmetric key mechanisms, initially distributing the key (sometimes termed "key boot-strapping") is often challenging. When the encrypted data resides on non-authenticated, removable media, the challenge is greater still.

[2] http://www.schneier.com/blowfish.html

Unfortunately in this case, the adopted key management approaches in these two code versions negate the security guarantees of the encryption primitive.

4.2.1. Version 9.1.8.0 Key Management

According to the vendor, PEB encryption is a general improvement implemented in all of their 9000 firmware series products. *Encryption* is a term that is sometimes used to refer to a broad variety of operations including encipher, hash, digitally sign, etc. In our case, the more common connotation is intended, as the PEB is only enciphered and we use this connotation hereafter. While it is generally not recommended that encryption be used as a sole integrity protection mechanism, encryption can establish some integrity properties, though the precise nature of these properties is dependent upon the protocol that the encryption implements. Conversely, key management flaws can compromise any or all of encryption's positive security properties.

In version 9.1.8.0, the PEB symmetric encryption-decryption key is passed to the terminal on the same PEB that it protects. The key itself is encrypted under a different key that is determined by the Election Qualification Code (EQC) data structure. Both the encrypted key and the key-encrypting value are stored in the PEB, alongside the other data blocks that are encrypted under the key. To decrypt the key, the attacker would need only know the EQC-defined value. This value is shared by all PEBs within the same county for the same election. It is stored in the clear and not cryptographically protected, thus can be acquired by anyone with an access to any PEB from the same county that is prepared for the same election. Therefore, an attacker that understands the EQC structure and that has access to a valid PEB could craft ballot definition files and encrypt them under the easily revealed key.

We highlight that the knowledge and skills necessary to exploit the encryption vulnerability are similar to that identified in the SAIT Report to exploit the PEB virus. While encryption appears to be an improvement out of hand, anyone that could attain a PEB and that has the skill necessary to successfully alter PEB files to create a virus, is highly likely to have sufficient skill to defeat this encryption process as well. Thus, the 9.1.8.0 software version is still vulnerable to a PEB malicious software injection attack, but as noted in section 4.1.2 above, only when triggered through operations accomplished before the terminal is open for voting.

One positive effect of this version change that implements PEB encryption is that it restarts the clock on the number of election cycles necessary for an injected PEB virus to widely propagate. That is, if a virus was injected onto a PEB or into a terminal under a prior firmware version in a prior election, even if that virus could survive the software update, it could not propagate to another uninfected, upgraded terminal. As noted in the SAIT Report, it is highly unlikely that a PEB virus could propagate beyond a single polling place during a single election cycle and wide propagation would require several election cycles.

4.2.2. Version 9.2.0.0 Key Management (#27).

In the 9.2.0.0 version, the vendor seems to have recognized the key management problem noted above, and they offer a separate key bootstrapping approach. In this version, they define two encryption modes: "ORIGINAL" and "KEYPEB". The "ORIGINAL" encryption mode in 9.2.0.0 appears to be identical to the one in 9.1.8.0 in which the PEB key is passed on the same PEB with the encrypted data. We note that ORIGINAL mode is not accessible since, even though the code for the original encryption approach is included in version 9.2.0.0, the operational

encryption style is statically set to "KEYPEB" and we found no mechanism or hook in the code to change this mode, so version 9.2.0.0 uses only KEYPEB mode.

The "KEYPEB" mode of encryption uses a key that must be loaded from a special type of PEB appropriately termed a “KEYPEB”. The KEYPEB’s exclusive purpose is to bootstrap the iVotronic terminal with the symmetric key that is used to decrypt the encrypted data on “ACTIVATOR” PEBs.

When the terminal detects that a PEB is inserted, it checks the PEBTYPE. A KEYPEB is distinguished by a special (guessable) string in the PEB. If the proper string is present and the machine is in BLANK state, the machine updates the key from the inserted PEB, overwrites any existing key with the key value stored in the KEYPEB, and does so without prompting the user for a password or other confirmation. The machine then displays a message indicating that the key update was successful and triggers a shutdown. Once the key is established, ACTIVATOR PEBs (i.e., any PEB not marked as a KEYPEB) must be encrypted under the installed key for successful use by the terminal.

If the machine is in state CLOSED or in state OPEN and no votes are registered when a KEYPEB is inserted, it must go through clear and test procedure before accepting a new key. Thus, the clear and test password must be supplied in order to change the key in these cases. For a terminal in any other state, if a KEYPEB is inserted, the terminal displays a warning message and shuts down.

Thus, if an attacker acquired a PEB and desired to change a terminal key, they could only do so before the terminal is open or if it is open, with no votes registered, and they know the clear and test password. These conditions are most likely to be met during a terminal “sleepover” or early in the morning at the polling place before the terminals are opened for voting or after they are opened but before any votes are cast on the target terminal(s).

4.2.3. Cryptography Implementation

While key management is the most prominent and dangerous flaw in the implemented encryption processes, there are other issues that should be addressed. First, it is not clear why Blowfish is the chosen encryption algorithm and AES should be considered as an alternative choice. However, Blowfish is a commonly used cipher that has resisted cryptanalysis for a number of years and should be considered safe even from a conservative viewpoint.

Of more concern is that the cryptography protocol is unsound. In the implementation, Blowfish is combined with a Cyclical Redundancy Check (CRC) to ensure ballot configuration data integrity and authenticity. The use of encryption as an authentication/integrity strategy is not supported by theoretical considerations and should be deferred to other cryptographic mechanisms such as keyed hashes, digital signatures, etc.

Still, while the use of encryption instead of other integrity primitives is a weak cryptographic design in general, it does not necessarily indicate avenues of attack. However, when the cipher is used in Electronic Code Book (ECB) mode, as is the case here, integrity protection is fundamentally diminished, or lost. The 9.1.8.0 and 9.2.0.0 encryption-decryption software leverages ECB mode’s block independence to allow targeted decryption, so a request to access a single PEB field does not necessitate decrypting the entire PEB.

This minor computational efficiency creates widely recognized integrity weaknesses. For example, using ECB mode may allow an attacker to "cut and paste" along block boundaries to

undetectably alter the ballot definitions or to potentially re-use ballot definition blocks from machine test cycles run earlier during the present election preparation process. This weakness can be offset by utilizing a different encryption mode that diffuses data changes many encrypted blocks.

Finally, the integrity provision that is incorporated into this encryption process is based on a Cyclic Redundancy Check (CRC) that is incorporated into the plaintext prior to encryption. This mitigates some of the risk of using ECB mode, but again, CRC is not a cryptographically strong process. In light of the level of sophistication required of an attacker (even against the unencrypted scheme), the implemented CRC does not provide significant additional security, though it may have some reliability value.

We restate for emphasis that these weaknesses, while fundamental, are dominated by the key management deficiency magnitude. Because the key management deficiencies are so easily exploited, it is unlikely that an intruder would need to attack the ECB/CRC weaknesses in the present configuration. Nonetheless, these mechanisms should be reengineered to provide strong security properties, thus supporting the defense in depth principle.

4.3. Code Structure

4.3.1. Non-Standardized Control Flow (#2, 3, 4).

The code structure for both analyzed versions remains consistent with that encountered in the original review, though there have been improvements. For example the control structure now handles process termination more gracefully via a single exit point where all functions return control before powering down the device. This "power off" function triggers an exit condition that is only satisfied in the main function that handles all necessary shut down procedures. We note that this is only one aspect of the challenging control flow and that the code base remains largely unstructured, non-standardized, and difficult to follow.

4.3.2. Code Not Highly Readable (6, 21)

Per the original report, part of the problem with unstructured code is that coding errors are repeated and extra code exists. This remains an issue in both submitted iVotronic versions. One example of how this still exists is the overuse of the function called to determine if a PEB is inserted. The PEB identifier (ID) is retained a global variable that is set when the PEB is inserted into the machine. However, the function to determine if the PEB is inserted is called in many locations to obtain the PEB ID. This pattern identified in the SAIT Report still exists in its entirety.

Additionally, Paragraph 6.3 (3) on page 34 of the SAIT Report details the iVotronic interrupt handler procedures and notes the complexity and readability issues associated with global, read-only state variables. There are still an excessive number of read-only state variables.

We note that while the code base contains a significant new code volume for processing bit mapped ballots, the newer software coding style is consistent with the legacy code, thus retains its architectural readability and reliability issues. These practices are exemplified by employing unstructured habits such as heavy reliance on global variables, conducting pointer arithmetic to determine string lengths, compounding multiple operations in a single line or statement, etc.

4.3.3. Potential for Timing-dependent Errors Associated With Updating GLOBAL Variables (#7)

This item addresses a general issue related to software architecture that implements interrupt processing using global variables. This architectural approach is still prevalent in the firmware.

4.3.4. Counter Issues (#19, 20)

The SAIT Report notes that counters are used for a wide variety of purposes. It further notes that their use appears correct, but suggests that counters should be consolidated and their use standardized. There is no improvement in this coding issue.

4.3.5. No Type Specified for Array Elements (#79)

This poor coding practice remains in the code as specified in the SAIT Report.

4.4. Code Defect When Trying to Read PEB Voltage (#13, 14)

The offending call to function to attain the PEB ID pointed out in the SAIT Report, has been removed. However, as the report pointed out, the code is aging and dangerous coding practices are present throughout. In this case, one function retrieves the PEB ID from the PEB. If there is no PEB inserted, the values for the PEB ID are set to 0. If this function is called after the PEB has been removed, it will cause the current vote to be canceled. Therefore, this is a dangerous function and its access must be carefully controlled.

Unfortunately, there are many opportunities for this function to be called. The code should be re-written in such a way as to ensure this function is only called when the PEB is inserted, and the PEB ID is cleared when the vote is complete. Although the specific instance mentioned in the report does not exist in this version, the possibility for this flaw to recur remains due to the wide-spread use of this function. It is recommended that this be resolved by updating the code to reflect these various possibilities.

4.5. Incorrect Invalid Vote PEB Log Entry (#16)

The improvement here was to modify event recording to log the events triggered by quick PEB insertion and record this event only to the auxiliary log. This presents the possibility of valid errors only being recorded in the auxiliary log. This seems like a quick-fix to something that should be examined more closely. Events need to be accurately identified in the logs, rather than minimizing log entries. In our opinion, instead of moving the audit events from the log, there should an event that records that a PEB was inserted and removed before normal PEB insertion processing could complete.

4.6. Automated Source Code Analysis (#29)

An important advance in software development is the emergence and widespread adoption of automated static source code analysis tools. SAIT Report identified many of the detected flaws using such automated tools against the 8.0.1.2 version and recommended that the vendor investigate and adopt applicable tools in their development environment. To date, it appears that this recommendation has not been heeded.

The SAIT Team exercised an automated source code analysis of the 9.1.8.0 and 9.2.0.0 software and in each case the automated tools identified a wide variety of potential flaws in each version's code base. Our hands-on investigation of these potential flaws led directly to many of the findings in this section.

Correspondingly, we reinforce and emphasize the SAIT Report recommendation that the vendor investigate and adopt an automated source code scanning approach for source code flaw identification, confirmation, and removal.

4.7. Off by One Errors (#65)

The SAIT Report identified two instances of loop iterations that, under rare circumstances could cause a read to uninitialized memory. One of these flaws is corrected (See Section 5.16), the other is not corrected.

4.8. Memory Comparison Not Comprehensive (#67)

In order to ensure flash currency, the iVotronic firmware compares the PEB data to flash memory contents at the beginning of the voting session by computing and comparing the CRC's of each. CRC computation is an efficient mechanism, but it does not detect all inconsistencies. In this case, since the compared data volume is small, the SAIT Report recommended comparing the entire one hundred and twenty eight bytes.

The version 9.1.8.0 code for this computation is unchanged from 8.0.1.2, though the repair is in place in version 9.2.0.0 (see Section 5.18).

4.9. Comment-Code Inconsistency (#68)

As stated in the SAIT Report, "The comment says 'Fast CRC verify' but the code actually implements the full check (not the fast CRC verification)". The impact of this flaw is operationally negligible, but it may indicate a confusion whose effects elsewhere.

4.10. Logic Errors in Quick CRC Code (#69)

When a quick verification is selected, only part of the ballot header is checked. There is no change in the 9.1.8.0 code, though the repair is in place in version 9.2.0.0 (see Section 5.19).

4.11. Logic Error in Ballot Definition Update Check (#70)

The process to update the ballot definition compares the new ballot definition header to the old ballot definition header and it is only updated from the PEB if the header changes. However, changes to the ballot definition can be made without anything changing in the ballot definition header. As a result, an intended ballot definition change can be ignored.

4.12. Possible Premature PEB Vote Cast Status (#71)

This issue is not applicable to Florida voters, but nonetheless, it remains in the code. The flaw is that in voter-activated mode, the PEB is marked as voted before the voter casts their ballot. Thus, in the case of a machine malfunction, the PEB may erroneously reflect that the voter had voted.

4.13. Non-Random Electronic Ballot Image Sequence (#89)

This problem recorded in the SAIT Report concerns the approach taken to randomly store ballots to prevent correlating a voter with their ballot. As noted in the SAIT Report, when the voter triggers a ballot write operation by pressing the VOTE button, the algorithm randomly selects one of twenty six flash memory sectors and appends the ballot to the randomly selected sector. The weakness here is that ballots are stored sequentially within the sectors. This approach has several negative, though low-impact implications.

The flaw remains in the examined code versions as described in the SAIT Report.

5. Improvements and Repairs

The purpose of this report is to determine whether flaws in the 8.0.1.2 iVotronic version are corrected in the currently examined versions, and in this section we address flaws that seem to be fixed. The nature of software security ensures that negative results (flaws that are NOT fixed) are naturally easier to identify and confirm than positive results (fixed). This is a direct result of Dykstra's principle that reflects the power of testing approaches to confirm a flaw's presence, alongside its theoretical limitations in showing the absence of flaws. The essence of this notion is captured in the following quote from Dykstra document archives at the University of Texas:

> *...program testing can be used very convincingly to show the presence of bugs, but never to demonstrate their absence, because the number of cases one can actually try is absolutely negligible compared with the possible number of cases.*[3]

Thus, while we discuss our opinions regarding fixed flaws below, we offer the strong disclaimer that we do not, and cannot *guarantee* that the noted flaws are fixed. More broadly, and maybe more importantly, we also cannot guarantee that the changes or repairs that removed, mitigated, or reduced these flaws did not cause regression faults. Thus, we do not offer guarantees, but only our best professional and academic opinions.

5.1. Weak Undervote Notification (#1)

One issue identified during the SAIT Report analysis was that the 8.0.1.2 ballot review process did not highlight or otherwise amplify undervotes. While the assessment of this user interface issue is beyond the scope of code review, we note, without evaluating their effectiveness, that versions 9.1.8.0 and 9.2.0.0 incorporate additional undervote emphasis mechanisms.

5.2. Non-votable Candidates (#9, 10)

In some election conditions, candidates may need to be listed on the ballot, but not be available for a voter to cast a vote for them. Version 8.0.1.2 employed a non-intuitive mechanism of inserting an ampersand ("@") as the first character in a non-votable candidate's name in text ballots. Since text ballots are no longer used, this vulnerability is not present in series 9000 iVotronic firmware versions.

While the control flow does not allow calling text ballot functions in 9.1.8.0, the code that exercises this option remains. All text ballot code is removed from version 9.2.0.0.

Non-votable candidates are identified in bitmapped ballots by a field in the ballot definition that now contains a properly named attribute that establishes this semantic property (i.e. non-votable candidate). This seems to meet the objective of the SAIT Report recommendation.

5.3. PEB Zero Audit Message (#12)

The code now checks for three different types of PEBs: SUPERVISOR, VOTER, and all other. If other than a legitimate PEB type is presented, the firmware now logs an appropriate event.

5.4. Get PEB Battery Voltage (#13, 14, 15)

The original flaw resulted from an attempt to check PEB voltage when no PEB was in place. The code had an ELSE statement that triggered the flawed call, but that statement no longer exists.

---

[3] http://www.cs.utexas.edu/users/EWD/transcriptions/EWD03xx/EWD361.html

5.5. Expand Audit Log to Record User Interaction (#23)

The SAIT Report recommended that the audit log be expanded to include a record of every user action. While this suggestion is not implemented, there is now an additional audit log and a Real Time Audit Log (RTAL) printer is now supported. This printer has the ability to produce a voter verifiable audit trail as votes are cast.

5.6. Memory Overrun in Variable Length Record (VLR) on the PEB (#24, 31)

The PEB virus identified in the SAIT Report directly resulted from a buffer overflow flaw in the primary function for getting VLR string data from the PEB. In the inspected versions, the length of the input string in this function is checked and the target string is always null terminated, thus repairing the previous overflow.

Correcting this flaw was an important improvement to the iVotronic firmware.

5.7. Buffer Overflow When Seeking the Election Identifier (#25)

According to the SAIT Report, this flaw is based on a string comparison of data from the compact flash card. The code now reads from the memory card and stores the maximum allowed number of characters into a temporary array. It checks the maximum length and pre-places a null terminator to end the string. The code then calls the function to access the ballot definition on the PEB, which, according to our analysis is now safe, and returns a string of the selected length or less. So, while the function call (strcmp) that the SAIT Report references still exists, the noted buffer overflow is no longer possible.

5.8. Passwords

5.8.1. General Password Handling (#45, 46)

This set of flaws has been significantly improved in these two versions. Passwords are no longer hard-coded or embedded in the source code, rather they are now user configurable. In addition, the password length is expanded from three to at least six characters.

This represents a dramatic improvement. Nonetheless, one further improvement is suggested. While passwords are user configurable, there is no mechanism that checks password strength. This leaves users free to select weak, easy to remember passwords, which violates security best practice. Such a mechanism to enforce strong passwords should be added.

5.8.2. Factory Test PEB (#47)

The SAIT Report identified a critical flaw in the 8.0.1.2 code in the form of a dangerous backdoor, where there is a provision to override all passwords by inserting a Factory Test PEB that can be easily created.

The identified code and all reference to the Factory Test PEB are removed from both 9.1.8.0 and 9.2.0.0. We do not find any other backdoor provisions in either version.

5.9. Flash Memory Consistency Handling (#48, 49, 50, 51)

According to the SAIT Report, if there is a mismatch in the flash, the ballot data is copied from the PEB. This solution focuses on recovery, potentially trading off accuracy and forensics capability. That is, if the flash memory copies differ, it may be important to know why that inconsistency occurred. The former approach of immediate recovery and continued processing

does not repair the inconsistency's cause and it is likely to prevent, or at least limit, later diagnosis capability.

In the analyzed versions, if the flash does not match, the system is shut down and displays an error message that describes the integrity event in the flash storage.

5.10. LogEvent Ignores Some Errors (#52)

The offending function no longer exists in the code base and the new function that writes to the audit log checks and appears to properly handle error conditions.

5.11. Error Handling Omitted (#54)

This item identified two functions (fseek and fread) that do not consistently handle error codes properly in the 8.0.1.2 firmware. In the subsequent versions, the noted errors are captured and are either recorded or displayed as determined by the user defined configuration.

5.12. Error Handling for Data Read From Compact Flash (#53)

This item is an extension of the previous flaw (Section 5.11 above). The function that accesses the compact flash does not check for returned error codes in the 8.0.1.2 firmware. In the subsequent versions, the noted errors are captured and are either recorded or displayed as determined by the user defined configuration.

5.13. Error Handling for Files Written to Compact Flash (#55, 56, 57, 58)

This functionality moved from the text ballot code to bit mapped ballot code in 9.1.8.0 and 9.2.0.0. Previously, the code did not check the return value to make sure the data was written successfully. In this version the calling function processes the return value. This is an improvement, though the code does not distinguish between the varying types of write errors.

5.14. Improper Error Handling on Emergency Close (#18, 59, 60, 61)

The SAIT Report noted fifteen instances where the Emergency Close function has potential flaws. Five instances related to improper string handling. The 8.0.1.2 code used a function with a character string parameter for emergency close. This function stored the character string into a fixed sized buffer in flash memory. The flaw was that the function copied the supplied string without proper bounds checking, which meant that the user could supply a long string which could end up being not null terminated in flash memory. Their error reporting strategy now uses integer status code values to pick from pre-defined error strings rather than user supplied data.

In nine other instances, it was possible that the terminal may return to user control rather than power down after the emergency close function was triggered. In tracing the code, this process did not return *normal* execution to the user, rather set the terminal into a mode that allowed graceful shutdown that provided user feedback and captured audit data as the terminal shuts down. Our analysis indicates that this is an appropriate emergency close strategy.

The fifteenth instance identifies a potential, though unlikely, recursion in the emergency close process. If the event log is full, the code makes a call to trigger an emergency close. The call to trigger an emergency close would have caused a recursive attempt to write a record to the already full event log.

The offending code is no longer executed in versions 9.1.8.0 or 9.2.0.0, so we analyzed the corresponding functionality in these versions and the recursion potential seems to be handled appropriately.

5.14.1. Error Handling Code Does Not Check for Errors When Writing to Flash Memory (#62, 63)

Previously, status codes produced by specific functions were ignored. This was fixed so that the status code is checked and the code exists gracefully if it fails.

5.15. File Index Search Algorithm Error (#64)

This flaw is a likely typo error in a search algorithm that is corrected in the analyzed versions.

5.16. Off by One Errors (#65)

The SAIT Report identified two instances of loop iterations that, under rare circumstances could cause a read from uninitialized memory. One of these flaws is corrected, the other is not corrected (See section 4.7).

5.17. Global Variable Wrap can Cause Premature Timeout (#66)

This logic error as described was possible during the modem process. The error involved timer logic that did not handle the timer's wrap-around behavior.

The code now expects a special start value to indicate to the function that it should reset this global counter variable. This prevents the wraparound from occurring based upon the assumption that the wraparound period is larger than any timeout value that would be requested.

5.18. Comparison Not Comprehensive (#67)

In order to ensure flash currency, the 8.0.1.2 code compares the PEB data to flash memory contents at the beginning of the voting session by computing CRCs of each and comparing the CRCs. CRC computation is an efficiency optimization that does not detect all inconsistencies. In this case, since the compared data volume is small, the SAIT Report recommended comparing the entire one hundred and twenty eight bytes.

The version 9.2.0.0 code compares all data. This issue is not addressed in version 9.1.8.0 (see Section 4.8).

5.19. Logic Errors in Quick CRC Code (#69)

In the 8.0.1.2 code, for a *quick verification*, only part of the ballot header is checked. The 9.2.0.0 version appears to correct this flaw, though it is not fixed in 9.1.8.0 (see Section 4.10).

5.20. Logic Errors Leading to Dead Code (#73, 74, 75 76, and 77)

The SAIT Report lists four instances where code segments are unreachable. Two of these instances are corrected in both analyzed versions. Two others are corrected in 9.2.0.0, but are not fixed in 9.1.8.0.

5.21. Poor Coding Practice (#80, 81)

These specific, operationally harmless issues were corrected in the analyzed versions.

5.22. Boolean Value Misuse (#83)

There is a flaw in 8.0.1.2 code that checks that the official election name in the election definition structure in terminal flash matches the official election name on the compact flash. It contained a logic error when checking the compare function's return value. This error is corrected in the analyzed versions.

5.23. Anomalous Vote PEB Event Messages (#88)

See Sections 4.4, 5.3 and 5.4, as they are related to this issue (this flaw and #13 were previously caused by the same error). This flaw has been fixed. It was only an issue with text ballots.

In addition, there is more thorough event logging and checking against the PEB. Previously, the code only checked whether the PEB was a supervisor PEB and if not assumed it to be a voter PEB. Now it checks for valid PEB types and fails gracefully if the PEB is not of an expected type.

5.24. Incorrect Code Parameterization for "NOCANDIDATES" Option (#85)

The SAIT Report identified a logical operation precedence error in a module that is triggered during Logic and Accuracy testing. The error appears to be fixed in the analyzed versions.

5.25. Insufficient Code Parameterization (#86)

The SAIT Report identified a logic error that can occur under rare circumstances. The proposed fix is incorporated into version 9.1.8.0. The function underwent major revision in 9.2.0.0 and appears to provide the desired functionality.

5.26. Analysis of Anomalous Audit Log Messages Regarding Voter PEBs (#88)

This error involved erroneous audit messages that indicated that ballots were cast with Voter PEBs, which are not used in Sarasota. The code now more precisely checks and reports PEB type.

6. Conclusion

Based on our analysis of the 9.1.8.0 and 9.2.0.0 iVotronic firmware versions, it is clear that the vendor took the findings and recommendations of the SAIT Report seriously and invested significant effort in removing flaws. The reported PEB Virus is no longer possible through the channel identified in the SAIT report, the weak password system is dramatically improved, and the audit reporting flaws are also addressed.

In spite of the significant improvements, the system remains at risk, with the most serious risk involving terminal sleepover. We provide summarizing notions below, beginning with Table 1 that summarizes our flaw status analysis.

Version	Not fixed	Improved	Fixed	NA
9.1.8.0	2,3,4,6,7,16,19,20, 21,26,27,28,29,32, 33,34,35,67,68,69, 70,75,77,79,90	1,11,13,14,15, 23,38,39,40,41, 42,43,44,46,55, 56, 57,58,65	9,10,12,18,24,25,31,45,47,48 ,49,50,51,52,53,54,59,60, 61,62,63,64,66,71,74,76, 80,81,83,85,86,88	5,8,11,17, 22,30,37, 72,73,78, 82,84,87,89

9.2.0.0	2,3,4,6,7,16,19,20, 21,25,26,27,28,29, 32,33,34,35,68,70, 79,90	1,13,14,15,23, 38,39,40,41,42, 43, 44,46,55,56, 57, 58,65	9,10,12,18,24,31,45,47, 48,49,50,51,52,53,54,59,60, 61,62,63,64,66,67,69,71,74, 75,76,77,80,81,83,85,86,88	5,8,11,17, 22,30,37, 72,73,78, 82,84,87,89
Table 1: Flaw Status Summary				

6.1. Elections Officials Must Rigorously Protect Removable Media Year-Round

All removable media associated with electronic voting systems is highly sensitive and must be protected year-round with the same status and priority as voted ballots. Removable media can transfer viruses and other malware in voting systems just as it can on personal computers and the impact can be much greater. Until and unless secure hardware enables strong removable media mutual authentication and integrity protection, elections officials must elevate the security level that they employ for every item of removable media associated with their voting systems. For iVotronic users, we strongly endorse the media handling recommendations from Section 7.9 of the SAIT Report, repeated here for emphasis, and we further recommend that all electronic voting system users employ corresponding strong media handling procedures.

1. Each terminal and each PEB should be assigned to a single precinct. This assignment should never be changed or rotated among precincts and should remain fixed for the lifetime of the equipment.
2. Master [and KEY] PEBs should be strictly controlled using procedures similar to those applied to paper ballots. They should be constantly under lock and key during the voting day, with sign-out and sign-in procedures to maintain the chain of custody at all times.
3. Polling place procedures should minimize PEB cross-pollination: i.e., minimize the number of terminals that any particular PEB is ever inserted into and minimize the number of PEBs that are ever inserted into any given terminal. For instance, officials might set an upper bound of 5, specifying that no PEB be used with more than 5 terminals and no terminal be used with more than 5 PEBs. Optimally, a poll-worker with a PEB would be assigned a set of terminals, no other PEBs would be used on those terminals, and that PEB would never be inserted in any other terminal.
4. Supervisor terminals should be rigorously controlled. No unsafe operation should ever be performed on any supervisor terminal, if it possibly can be avoided. (See Appendix C [of the original SAIT Report] for a list of safe and unsafe operations.)
5. Numbered tamper-evident seals should be used to deter tampering with the CF card slot. Logs should be kept of all seals applied and/or removed, and two-person controls should be applied when election workers handle CF cards.

6.2. Comprehensive Voting System Source Code Security Review is Essential

Electronic voting is here to stay. Several recent software reviews, while conducted by different scientists and under varying circumstances, they share one consistent theme: Dangerous software

errors exist in virtually all electronic voting systems that have been rigorously examined to date. This clear message is again reinforced in this analysis. In order to assess the ability to mitigate software flaws and to develop and implement such mechanisms when they exist, the flaws must be identified before electronic voting systems are allowed to be used in elections.

We restate that this project is limited in its scope and does not reflect the depth of comprehensive review that we describe in this paragraph.

6.3. Voting Systems Must Employ Software Defense in Depth

While encrypting the PEB could reduce buffer overflow vulnerability, all memory overrun instances should be detected and removed. This principle is perfectly illustrated in this report, where a flaw in an encryption protocol allows attackers access to a buffer overflow vulnerability.

Voting systems are high assurance applications where defense in depth is essential. The developer's goal must be to identify and remove as much vulnerability as possible and to layer defenses wherever possible.

6.4. Key Management is a Difficult [Voting System] Problem

It is a well-known that key management is one of the most challenging issues in using cryptography to protect information systems. Whether a system embeds public keys in the source code or attempts to bootstrap a private key separate from application initialization, the challenges of managing keys that enable even mutual authentication are great.

6.5. Voting Systems Must be Engineered for High Assurance

Representative government officials can only be duly elected if voting systems are secure and reliable. Sophisticated software systems are naturally error prone, but voting systems cannot endure the high flaw rates that some shrink wrapped applications absorb. Moreover, there is no way to thoroughly test voting systems for security properties in an operational setting. Because they are critical, testing-limited systems, voting system vendors must employ methodologies that facilitate simplicity, maintainability, and reliability. They must study security principles and be willing to find and apply sophisticated security skills to ensure that voting systems reflect necessary security properties.

7. Acknowledgements

The SAIT Team utilized the static analysis tool Coverity Prevent SQS to facilitate our review during this project and note that Coverity is a SAIT Laboratory partner.

We thank Matt Bishop, David Jefferson, and David Wagner for their helpful comments on an early version of this report and to Ted Baker for his assistance and comments in the Lab.

8. References

1 A. Yasinsac, D. Wagner, M. Bishop, T. Baker, B. de Medeiros, G. Tyson, M. Shamos, and M. Burmester, "Software Review and Security Analysis of the ES&S iVotronic 8.0.1.2 Voting Machine Firmware, Final Report", Security and Assurance in Information Technology Laboratory, Florida State University, February 23, 2007, http://election.dos.state.fl.us/pdf/FinalAudRepSAIT.pdf.

2 Florida Department of State Audit Report of The Elections Systems and Software, Inc.'s iVotronic Voting System in the 2006 General Election for Sarasota County, Florida, February 23, 2007, http://election.dos.state.fl.us/pdf/auditReportSarasota.pdf

3 David Wagner, David Jefferson, and Matt Bishop, "Security analysis of the Diebold AccuBasic Interpreter", Voting Systems Technology Assessment Advisory Board (VSTAAB), University of California, Berkeley, February 14, 2006.

Appendix A List of Flaws From the SAIT Report

1	Pg 9 -2.5.4	Ballot design issue- text ballot.
2	Pg 16 - 3.4.3	Relevant Code Properties- Aging code base lacking standardized control flow.
3		Team suspected corresponding reliability.
4		Too many globals – lack of high level design.
5		Did not examine San Disk code
6	pg 16- 3.4.3	Code not highly readable
7	pg 17- 3.4.3	Potential for timing- dependent errors associated with updating GLOBAL variables.
8	pg 18- 4.1.4	Re-vote Pages- Revote contest very different from original presentation- may confuse voters.
9	pg 22- 6.1.1	Overview- The "@" used for (non-votable) No Candidate filed- may cause a no vote to candidate if it appears as part of candidate name.
10	pg 22 6.1.1	Potential Remedy- non-votable candidates should be controlled by well-defined, clearly identified mechanism. (in place of @ symbol)
11	pg 22 6.2.1.1	Overview- Slow touch response
12	pg 23- 6.2.1.2	Event 18 "Invalid Vote PEB" If PEB not inserted – default is "VOTER PEB"- may need another EVENT To avoid confusion
13	pg 24- 6.2.1.2	Technical Analysis- Code defect when Spanish ballot not enabled- trying to read PEB voltage
14	pg 24- 6.2.1.2	Technical Analysis- Normal ballot cast reads PEB type w/no validity check (PEB queried when not present) Part of ADA & Spanish issue
15	pg 24- 6.2.1.2	Technical Analysis- PEB type 0 caused from language selection on ADA when no Spanish.
16	pg 25- 6.2.1.2	Paragraph 4- "Invalid Vote PEB" - PEB dropped in to verify the PC is ok, but removed faster than time to read PEB- see pg 23- 6.2.1.2
17	pg 30- 6.2.1.6	Mitigating factors- Would like to see calibration each election -could we show previouis calibration location on screen overlayed w/ ideal points?
18	pg 32- 6.2.3.2	Technical Analysis- Emergency close (event memory full) can cause infinite recursion because "EC" also calls log event
19	pg 34- 6.3	Counters- Code could be simplified by consolidating some of the timers
20	pg 34- 6.3	Counters- code could be simplified by adopting a uniform policy of count down or count-up paradigm instead of current arbitrary alternation.
21	pg 34- 6.3	Read-only variables- too many read-only state variables in code- confusing
22	pg 35- 6.4.1	Suggestions to improve audits:

23	pg 35-36, 6.4.1	Voter Action Log- Expand audit log to record user interactions- may even allow manual tally by reading voter activity.
24	pg 37, 7.2-7.3	Vunerability Verification- Buffer overflow that may allow hacker attacks (VLR Bounds Check only)
25	pg 38, 7.4.1	Compact Flash (CF) Cards- CF vulnerability: check string lengths in order to prevent buffer overflow (other places)
26	pg 37-41	General- Fear of spread of viruses using PEB, CF, Terminals
27	pg 44- 7.8	Fixing the Virus Vulnerabilities- Introduce input validation and defensive programming throughout the code.
28	pg 44- 7.9	Procedural Defenses to Remediate these Vulnerabilities- Procedural changes- ES&S side- none- customer level suggestions only
29	pg 53- #10	Use source code analysis tools
30	pg 57, Appendix B	Technical Analysis of the PEB Virus Threat - Numerous instances of possible overflow:
31		▪ buffer over-run
32		▪ array out-of bounds errors
33		▪ integer overflow vulnerabilities
34		▪ other security holes?
35		▪ missing "input validation" in iVo
36		▪ find buffer over-run w/o needing source code
37		▪ Virus Barriers?
38	page 62-65, Appendix C	Safe and Unsafe Operations - Unsafe to perform w/untrusted PEB:
39		▪ opening Poll/ Terminal
40		▪ Late Zero Tape
41		▪ Printing/ modeming
42		▪ L&A - all test
43		▪ Clear Supervisor PEB votes
44		▪ Upload PEB votes
45	Page 66-67, Appendix D	Passwords - Fixed passwords are hard-coded in version 8012.fmw
46		▪ firmware upload very vulnerable
47		Factory test PEB
48	Page 69, #2 Appendix E	Flash mismatch, no events logged- should show error- and close term. ▪ Refresh from PEB if flash errors- called from many places. ▪ ET1- ▪ ET2- No action taken if flash mismatch
49		
50		
51		

52		▪ ET3-Error handling- check 3 Audit loop ignores errors- should do EC
53	pg 69, #3 Appendix E	▪ EC1-no CF file checks- when using fseek, fread, errors ignored
54		▪ EC2- fread()- fails to notify of certain status error conditions. ▪ EC3- fseek()- need to check both return value & err_flag.
55 56 57 58		▪ EC4- no check of return value ▪ EC5- ballot image limit check needed. ▪ EC6- assumes \ PEB directory. ▪ EC6 (cont'd) already exists
59	pg 70, #4 Appendix E	▪ EE1-EE5 - does not handle Emergency Close (EC) events safely
60		▪ EE6- EE14- Functions call EC but fail to die
61		▪ E15- Emergency Close Event - If event log full, infinite loop occurs- due to Emergency Close causing another Emergency close.
62 63	Page 71, #5 Appendix E	▪ EM1- ▪ EM2- no error checking when writing to flash or erasing flash
64 65	pg 71, #6 Appendix E	▪ L1- error in binary search algorithm - ADA files- may cause incorrect bitmap display with bitmap ballot. May play wrong audio file for text ballot ▪ L3- may return wrong file from CF- unlikely ▪ L4- Loop termination error
66		▪ L2- Timer wrap may cause premature timeout. May cause modem transmission failure.
67		▪ L5- More robust just to compare PEB w/ RAM skip CRC checking
68 69		▪ L6- Comment in code incorrect about "Fast CRC verify" ▪ L7- Several issues in quick verify
70	pg 73, #6 Appendix E	▪ L8- Voter PEB (Voter Activated_ checks first 256 bytes Ballot Header if match- assumes same as already stored- unsafe assumption
71		▪ L9- Voter Activated- Marks (and time stamps) Voter PEB before voter actually casts ballot could cause disenfranchised voter if terminal crashes prior to vote cast.
72	pg 73, #7 Appendix E	C Language Issues
73		7.1 Dead Code:
74		▪ CD1-
75		▪ CD2-
76		▪ CD3-
77		▪ CD4-
78		7.2 Missing or Inaccurate types

79		▪ CT1-
80		▪ CT2-
81		▪ CT3-
82		7.3 Erroneous use of Booleans
83		▪ CB1-
84		7.4 Syntactic Issues-
85		▪ CS1-
86		▪ CS2-
87	pg 78, Appendix F	Analysis of Anomalous Audit Log Messages Regarding Voter PEBs - Numerous voter shown cast by voter PEBs. Only Supervisor PEBs used in election. The votes cast by "Voter PEB: bug was found in 8.0.1.2 text FMW. It had to do with ADA set and Spanish not enabled (text ballot).
88		
89	pg 82, Appendix G	Anonymization of cast vote Records- Vote images not random within sectors.
90		

Appendix B Buffer Overflow Discussion

Buffer Overflow and Root Kits

Buffer overflows are a common software vulnerability that are well understood and a good overview is given in the SAIT Report, so we omit tutorial information here. However, we note that the malware we discuss in this report could include functionality commonly known as "root kits." Once injected into a system (often through a buffer overflow flaw) root kits take control of the system. One of the critical capabilities that root kits enjoy is that they can mask their presence and may be able to delete all evidence that they were ever present.

We point out the buffer overflow vulnerabilities in this code, like that identified in the SAIT Report, could inject malware with root kit functionality into the terminal. Once this software was in control, these programs can be pervasive even against sophisticated removal procedures.

Low Hanging Fruit

Virtually all of the buffer overflow vulnerability that we detected could be found by searching the code base for "strcpy", "strcat", and "sprintf". There are many lists of vulnerable functions available in the literature and on the Internet and several open source and proprietary software tools automatically detect these flaws.

It is also possible to easily repair the most commonly occurring flaws by replacing the vulnerable functions with safe functions, such as strncpy, as we see used in new code in version 9.2.0.0, though many vulnerable operations remain in legacy portions of that code.

Unsafe Operations

In this table, operations are UNSAFE if their execution can result in a buffer overflow. Note this is slightly different from the SAIT Report definition where UNSAFE operations pulled malicious data from the PEB. The operations below access malicious data that may have been installed into terminal flash at poll opening.

Operations listed as UNKNOWN either were not evaluated, or we investigated them but did not find a specific buffer overflow. We did not comprehensively review these functions so we do not label any operations as SAFE.

Operation	9.1.8.0	9.2.0.0
Starting the terminal	unsafe - if ServiceFlag set	unsafe - if ServiceFlag set
Opening Polls	unsafe	unknown
Voting	unsafe	unsafe
Closing Polls	unsafe (reports vulnerable)	unknown
Printing Reports	unsafe (reports vulnerable)	unknown
Modeming Reports	unknown	unknown
Prepare Voter Peb	unknown	unknown
Opening Terminal (BLANK/LOADED)	unsafe	unsafe
Closing Terminal (OPEN)	unknown	unknown
Late Zero Tape	unknown	unknown

Unlocking Terminal	unknown	unknown
Clear and Test	testing votes unsafe	canceling ballot unsafe
Set Time and Date	unknown	unknown
Qualify Pebs	unknown	unknown
Upload Peb to CF	unknown	unknown
Test Printer	unknown	unknown
Test Modem	unknown	unknown
Upload Firmware	unknown	unknown
Load System Files	unknown	unsafe (wavs unsafe)
Enable Audio Ballot	unsafe (wavs unsafe)	unsafe (wavs unsafe)
Set Volume	unknown	unknown
Force Coded Ballot Entry	unknown	unknown
Logic and Accuracy	unsafe	unsafe
Print with Writeins	unsafe	unknown
Print without Writeins	unknown	unknown
Transfer Results to Peb	unknown	unknown
Machine Service	unsafe	unknown