



Florida Fair Elections Coalition Inc.
112 W. New York Ave., Suite 201, P.O. Box 317
Deland, Florida 32721
386-736-8086
www.FloridaFairElections.org

Florida Fair Elections Coalition has read the proposed rule IS-2.004 and has the following comments:

IS-2.004 implements law 101.292, 101.292(3), 101.294, 101.295 and 101.5604, F.S.

It is clear that the proposed rule is designed to make tests such as those conducted in Leon County in 2005 difficult if not impossible, even though those tests have been to the public good. The Leon County tests, which have become known nationally as the "Hursti hacks" after the computer consultant, Harri Hursti, who conducted them, have resulted in numerous other security vulnerabilities being discovered in voting systems. The ability to exploit the security vulnerabilities exposed in the Leon County and other security tests calls into question the accuracy and security of elections both in Florida and around the country.

The Supervisors of Elections are required to perform tests under Florida Statutes § 101.5612 which this rule covers (even though this statute is not listed as one of the laws implemented in the preface to the proposed rule). The constitutionality of the State taking over the Supervisors of Elections functions is questionable under the Florida Constitution and certainly, Florida Statutes § 101.58 should not be read to allow the Department of State to totally usurp the normal functions of supervisors of elections without cause.

The proposed rule gives too much power to voting machine vendors. Florida voting systems should be used and tested in an independent manner for the benefit of Florida voters. Section (5) (b) of the proposed rule states: "A voting system may be used in any manner approved by the vendor and must be approved in writing by the Division." (Emphasis added). Voting machine suppliers are required by the Florida Voting System Standards to provide a "User's Manual" to describe safe and efficient use of the voting system. However, it is unreasonable to expect a vendor to willingly cooperate in security testing after a sale – the vendor has everything to lose and nothing to gain. Granting power to the vendor to approve the way the voting system is used is absurd. There are limits to vendor discretion. The taxpayers have purchased the system and its use and security should be determined by them through their elected officials.

Requirements of the Test Plan

While it is questionable that state law authorizes the Department of State to control the testing of voting systems by a supervisor of elections, there are also some dubious concepts and requirements in the testing plan itself.

The first item listed as a minimum requirement for a test plan is protection of vendors' intellectual property. This might give the wrong impression that the intent of requiring such a plan is to protect the vendors.

Also, any test plan should not place the vendor in charge of the test process as this is a conflict of interest. This rule effectively does that.

There are no substantive standards listed for approval of the plan. This might allow the Department of State to arbitrarily withhold approval of any plan submitted.

The reason to demand that any assessment be conducted by an individual certified by one of three listed professional associations is unclear. There are other credible professional

associations and many credible computer scientists who do not have these affiliations. Also, one point of security testing would seem to be to determine what level of expertise is needed to exploit the systems' vulnerabilities. If a 15-year old with a laptop can alter election results, we need to know it. The fact is that Paul Craft, former Chief of the Bureau of Voting Systems Certification, apparently could not meet this requirement (no professional affiliations are listed on his resume

It is illogical for the state to assume responsibility for test procedures and then assign responsibility to the supervisors of elections for any problems those tests might uncover. It is a statutory function of the Division of Elections to promulgate minimum security procedures. Once major security vulnerabilities have been uncovered, it is incumbent on the state to step forward with meaningful solutions. The supervisors of elections do not have the power over vendors or others in the chain of custody that the state enjoys. The following wording, or something similar, would be more appropriate:

The Supervisor shall implement all available procedures to mitigate adverse effects resulting from any problem discovered during testing and may make recommendations to the State. The State, after reviewing the assessment, shall conduct necessary investigations into the problem, shall issue technical advisories and regulations designed to mitigate any adverse effects, and should decertify the voting system if appropriate.

It is redundant to state that the Supervisor's procedures will be replaced if and when the State issues procedures as that is the law

The current procedures for testing security during the certification process are not always followed and the procedures that do exist are inadequate. The State should have meaningful procedures in place for security testing and follow them. Adequate security testing would have uncovered the security vulnerability exploited in the Hursti hacks. At the very least, the State should have recognized the existence of an interpreter and interpreted code in the Diebold AccuBasic language and responded appropriately. The Hursti Hack should have failed because the State had eliminated that security vulnerability. Instead, it is still possible and will remain possible in the 2006 elections. However, the responsibility of the State to detect and mitigate security vulnerabilities does not excuse each supervisor of elections from his or her duty to make our elections as safe as possible. The State could help them by providing rational guidelines, not mandates, for test plans and by choosing the least burdensome methods of safeguarding intellectual property.

Editing Notes

Section 1(f) defines voting system as a method of casting ballots and processing "voters." This should read "votes." If "voters" was meant a more accurate term for voting than processing should be used

The proposed rule appears to skip from section 3 to 5 without a section 4.

Summary

While appropriate regulations are welcomed and encouraged, their aim should be the security of voting systems. Does the Department of State want to protect the integrity of Florida elections or does it want to minimize the possibility that real threats will be found? Electronic voting systems pose problems and risks to our democratic process that have never before been contemplated. All states are struggling with ways to prevent the exploitation of security vulnerabilities and any new findings should be welcomed, not condemned.

The task of the Department of State is to make these systems safe not pose obstacles to their testing for safety. The proposed rule constitutes just such an obstacle.

Matthews, Maria I.

From: Anita Lapidus [nitalapidus9@cfl.rr.com]
Sent: Monday, June 12, 2006 11:41 AM
To: Matthews, Maria I.
Cc: 'Susan Pynchon'
Subject: FFEC_Comments_on_Proposed_Rule_IS_2[1].015_FINAL.pdf
Attachments: FFEC_Comments_on_Proposed_Rule_IS_2[1].015_FINAL.pdf

Dear Maria:

Attached please find the Comments of Florida Fair Elections Coalition on the proposed 1S1.015. Please include them in the record of today's proceedings. Thank you in advance for your cooperation in this matter.

Sincerely

Anita Lapidus
General Counsel
Florida Fair Elections Coalition.

6/12/2006



Florida Fair Elections Coalition Inc.
112 W. New York Ave., Suite 201, P.O. Box 317
Deland, Florida 32721
386-736-8086
www.FloridaFairElections.org

Florida Fair Elections Coalition has read the proposed rule IS-2.015 and has the following comments:

IS-2.015 implements law 101.015, F.S.

101.015(3) states:

The Department of State shall adopt rules to achieve and maintain the *maximum degree of correctness*, impartiality, and efficiency of the procedures of voting, including write-in voting, and of *counting, tabulating, and recording votes* by voting systems in this state. (Emphasis added).

The amendment to the proposed rule, far from achieving the maximum degree of correctness in counting, tabulating and recording votes, does not even achieve a minimum degree of correctness.

Supervisors of Elections Responsible for Establishing Security Procedures

F.S. 101.015(4)(a) states that the Department of State shall adopt rules establishing minimum security standards for voting systems. However, prior to the proposed amendment, the Department of State has left the establishment of minimum security standards up to the individual supervisor of elections. The proposed amendment proscribes very few security procedures other than those included in a *Technical Advisory*¹ issued by the Florida Division of Elections on March 3, 2006. Most security procedures are still left up to the individual supervisor of elections. Furthermore, the security procedures contained in the March 3rd *Technical Advisory* that have been transcribed to the proposed rule are inadequate.

Proposed Amendment Fails to Meet Recommendations of the VSTAAB Report

The proposed amendment represents an attempt to follow the recommendations made in a report by the California Voting System Technology Advisory Board (VSTAAB) in a report dated February 14, 2006² but fails in numerous key aspects.

The VSTAAB report makes it clear that the security vulnerabilities recently discovered in the Diebold voting systems are so huge that guarding election media (including memory cards) is a "short-term mitigation strategy"...good only "for local elections in the short term." Yet this rule incorporates these security procedures as if they represent solutions suitable for major elections.

Vital Recommendations Ignored

Last December, California Secretary of State Bruce McPherson asked an advisory board of computer scientists to conduct a security review of the memory card components for both Diebold's AccuVote-OS (optical scan) and AccuVote-TSx (touchscreen with voter verified paper audit printer) voting systems. The report, "Security Analysis of the Diebold AccuBasic

¹ Technical Advisory Memorandum issued March 3, 2006 by Dawn Roberts, Director of the Florida Division of Elections

² "Security Analysis of the Diebold AccuBasic Interpreter," by David Wagner, David Jefferson and Matt Bishop of the California Voting Systems Technology Assessment Advisory Board and by Chris Karlof and Naveen Sastry of the University of California, Berkeley, February 14, 2006

Interpreter", which was released by the California's Voting System Technical Assessment and Advisory Board (VSTAAB) in February, confirmed numerous security vulnerabilities in Diebold's electronic voting systems.

The VSTAAB report is a far more comprehensive and in-depth analysis than the other documents mentioned in the introduction to the proposed rule. The other reports, which were prepared by Cyber Laboratory³, do not adequately report the huge security vulnerabilities in the Diebold system. *It is understandable that Cyber Lab would be protecting Diebold since Diebold is its client.* In fact, the Cyber report states on its cover that it was prepared for Diebold Election Systems, Inc. Furthermore, Cyber sought to minimize any criticism that might be leveled at it for its failure to discover that this system did not comply with the federal 2002 Voting System Standards. Therefore, the VSTAAB report is the far more credible report since it was independently commissioned by the California Secretary of State and was prepared by a team of 5 renowned computer scientists, 2 from the University of California at Berkeley and 3 from California's Technical Advisory Board.

Following is a summary of key points in the VSTAAB report that make it clear that the "security" procedures recommended in the proposed amendment to Rule IS-2.015 do nothing to assure that votes are being tabulated and counted accurately:

The original goal of the VVSTAB scientists was to verify the results of an earlier test of voting system security that was performed in Leon County, Florida. The Leon County test, conducted by Finnish computer programmer Harri Hursti, definitively proved that election results can be altered on a Diebold voting system, without detection, by using a single memory card. The computer scientists, who conducted only a limited review of the Diebold source code, not only confirmed the validity of the "Hursti Hack" - they also discovered 16 other serious security "bugs," each of which could be exploited to alter election results without detection.

And the report cautions that "these are just the bugs we were able to find; there are quite possibly others we did not notice..." Elsewhere in the report they reiterate, "There may, of course, be additional bugs, or [different] kinds of bugs, that we did not find." The report also confirms that the Diebold TSx (touch-screen) has many of the same vulnerabilities as the optical scan system that was tested in Leon County.

The scientists wrote, "Clearly there are serious security flaws in the current state of the AV-OS [optical scan] and AV-TSx [touch-screen] software." The scientists state that the security vulnerability exploited by Harri Hursti cannot be cured by rewriting the source code. They do note that the 16 additional bugs that were uncovered, while serious and able to change election results without detection, are "easily fixable" through a code re-write, but they go on to say that the real problem lies with the AccuBasic language and the "interpreted code" used by the Diebold system. The report notes: "Interpreted code in general is prohibited by the 2002 Federal Election Commission Voting System Standards and its successor standard, the Election Assistance Commission's Voluntary Voting System Guidelines due to take effect in two years. In order for the Diebold software architecture to be in compliance, it would appear that the AccuBasic language and interpreter have to be removed, or the standard will have to be changed."

Since there are valid security reasons why interpreted code is not allowed under the current standards, the only legitimate action would appear to be to the removal of the AccuBasic language and interpreter. This would essentially involve redesigning the entire Diebold system.

According to the report, "Once the attacker can replace the running code of the machine, the attacker has full control over all operations of the machine. Some of the consequences of this kind of compromise could include:

³ "Diebold Election Systems, Inc. Source Code Review," February 23, 2006, Prepared for Diebold Election Systems, Inc. by Cyber, Inc.

"The attack could manipulate the electronic tallies in any way desired. These manipulations could be performed at any point during the day. They could be performed selectively, based on knowledge about running tallies during the day. For instance, the attack code could wait until the end of the day, look at the electronic tallies accumulated so far, and choose to modify them only if they are not consistent with the attacker's wishes."

"The attack could print fraudulent zero reports and summary reports to prevent detection."

"The attack could modify the contents of the memory card in any way, including with the electronic vote counts and electronic ballot images stored on the card."

"The attack could erase all traces of the attack to prevent anyone from detecting the attack after the fact. For instance, once the attack code has gained control, it could overwrite the malicious AccuBasic object code (.abo file) stored on the memory card with legitimate AccuBasic object code, so that no amount of subsequent forensic investigation will uncover any evidence of the compromise."

"It is even conceivable that there is a way to exploit these vulnerabilities so that changes could persist from one election to another. For instance, if the firmware or software resident on the machine can be modified or updated by running code, then the attack might be able to modify the firmware or software in a permanent way, affecting future elections as well as the current election. In other words, these vulnerabilities mean that a procedural lapse in one election could potentially affect the integrity of the subsequent election."

In addition the report notes:

"It is conceivable that the attack might be able to propagate from machine to machine, like a computer virus. For instance, if an infected memory card is inserted into an infected voting machine, then the compromised voting machine could replace the AccuBasic object code on that memory card with a malicious AccuBasic script. At that point, the memory card has been infected, and if it is ever inserted into a second uninfected machine, the second machine will become infected as soon as it runs the AccuBasic script."

"In addition, most of the bugs we found could be used to crash the machine. This might disenfranchise voters or cause long lines. The bugs could be used to selectively trigger a crash only in some machines, in some geographic areas, or based on certain conditions, such as which candidate received more votes. For instance, it would be possible to write a malicious AccuBasic script so that, when the operator prints a summary report at the end of the day, the script examines the vote counters and either crashes or continues operating normally according to which candidate is in the lead." And this just scratches the surface of what the report reveals. It is imperative that all election officials and all potential purchasers of voting systems read this report in its entirety. There are many more important points that must be understood, including the weaknesses in the Diebold system involving inadequate or nonexistent encryption codes that allow easy access to the entire system including the GEMS central tabulator."

The tests done by the California scientists prompted Florida's Division of Elections to issue a *Technical Advisory* intended to address the security vulnerabilities that were found. However, the Florida *Technical Advisory* is completely inadequate in that it only mandates the short-term mitigation strategies recommended by the VSTAAB report for local elections, while ignoring the recommendations for removing the AccuBasic language and interpreter before using the machines in any statewide election. To quote directly from the VSTAAB report, "While these strategies do not completely eliminate all risk, we expect they would be capable of reducing the risk to a level that is manageable for local elections in the short term. In the longer term, or for statewide elections, the risks of not fixing the vulnerabilities in the AccuBasic interpreter become more pronounced. Larger elections, such as a statewide election, provide a greater incentive to hack the election and heighten the stakes. Also, the longer these vulnerabilities are left unfixed, the more opportunity it gives potential attackers to learn how to exploit these vulnerabilities. *For statewide elections, or looking farther into*

the future, it would be far preferable to fix the vulnerabilities discussed in this report.”

The proposed amendment does not state that the proposed security procedures for election media are short-term mitigation strategies. The proposed amendment makes no attempt to assure the accuracy of statewide elections.

There is one noteworthy statement in the Florida Technical Advisory that says the security vulnerabilities discovered and the recommendations made in the VSTAAB report are applicable not only to Diebold, but apply “to all voting systems deployed in Florida.” That would include ES&S and Sequoia!

Another important point made in the VSTAAB report is that “successful attacks can only be detected by examining the paper ballots.” (pg. 2)

Unfortunately, 15 Florida counties have no paper ballots and the remainder of Florida’s counties, which still use marksense (paper) ballots for all but their disabled voters, are adversely affected by a Florida law passed in 2005 that prohibits the full manual recounting of paper ballots in any race under any circumstance. This means that no Florida jurisdiction has the ability to confirm that machine counts are correct or to discover whether tampering has occurred. The Florida Department of State has no way to conform to the requirements of 101.015(3) to provide “the maximum degree of correctness...in counting, tabulating and recording votes.”

How can the Division of Elections assure that election media has not been tampered with when there is no ability to examine (recount) paper ballots?

New Security Vulnerabilities Revealed

It should be noted that since the VSTAAB report was issued on February 14, 2006, a new security vulnerability has been discovered in the Diebold TSX touch-screen voting system that has been called the worst defect ever discovered in a voting system. This defect would allow an attacker to take over a voting system and maliciously alter the results of not only that election but all future elections – all without detection. The attached article from the *New York Times*⁴ and a column from *Newsweek*⁵ summarize this new vulnerability, which is not addressed at all by the proposed amendment.

Short-term Mitigation is Inadequate

The proposed amendment fails even as a short-term mitigation strategy for local elections. Section (5)(g)(2)(a) provides inadequate protection for memory cards when it states that “no election media is [to be] left unattended or in an unsecured location *once it has been coded for an election.*” The VSTAAB report makes it abundantly clear that a memory card is vulnerable to tampering *before* it has been coded for an election, that a memory card is vulnerable to tampering at any point in its life-cycle, and that there is no way to determine if such tampering has occurred (VSTAAB pp 24-27).

It is obvious that all election media, including memory cards and paper ballots, require a secure chain-of-custody. However, the procedures proposed in the rule are merely stop-gap measures and should be acknowledged as such. The only rational response to the scientific evidence is a complete redesign of these machines and to develop thorough security testing as an integral part of the process, something that is not currently done.

The proposed amendment requires that two people are always in attendance around any election media, but in Section (5)(g)(3) states that “a security breach must be confirmed by

⁴ *New York Times*, “New Fears of Security Risks in Electronic Voting Systems,” by Monica Davey, Gretchen Ruethling and John Schwartz, May 12, 2006

⁵ *Newsweek*, “Will Your Vote Count in 2006?” by Steven Levy, May 29, 2006

more than one individual." If two people were in the secured storage area and one commits a security breach, it would not be adequate for the second person to report it since the report must be confirmed by someone else. If both people leave cards unattended and that breach is discovered by a third person, shouldn't one person finding the abandoned cards be enough? And wouldn't the requirement of dual confirmation stifle whistle-blowing by one individual? It would also mean that the State is choosing to err on the side of less caution. It is not a rational or supportable choice.

At the very least, all Diebold memory cards (and their equivalent for other electronic voting systems) should be replaced to start over with "clean" media, although this in itself is no guarantee that tampering may not occur at some stage in the life of the memory card. Based on the VSTAAB report, it is clear that any or all existing memory cards could already have been compromised.

Effect of Proposed Amendment

The proposed amendment seems far more concerned with protecting "proprietary" vendor information than with protecting votes. Why is Florida allowing its citizens to cast votes on secret software with no means to assure that those votes are being counted accurately and no means to determine if election tampering has occurred?

The Department of State has not met the goal stated in the "Notice of Proposed Rule Development," e.g. that "the purpose of this rule amendment is to provide *comprehensive* security procedures to ensure the *highest level* of voting system protection."

The proposed rule amendment does not meet the requirements of F.S. 120.52(8)(e), incorporated by reference under F.S. 20.10(3), because it is arbitrary and capricious and is not supported by the facts. It is being adopted without reason, since the document it purportedly responds to recommends far more comprehensive mitigation strategies.

Adopting the amendment to Rule IS-2.015 would be a disservice to the State of Florida because it could lead to the false assurance that security vulnerabilities in the State's electronic voting systems have been effectively dealt with, when in fact nothing could be further from the truth.

Summary

It is obvious that all election media, including memory cards and paper ballots, require a secure chain-of-custody. However, the procedures proposed in the rule are merely stop-gap measures and should be acknowledged as such. The only rational response to the scientific evidence is a complete redesign of these machines and the development of thorough security testing as an integral part of the process, something that is not currently done. Until that time, based on the credible evidence presented, the accuracy and security of Florida's elections are in doubt.

M E M O R A N D U M

TO: Supervisors of Elections
FROM : Dawn K. Roberts, Director
DATE: March 3, 2006
SUBJECT: Technical Advisory

Purpose:

This advisory concerns enhancements to voting system security procedures that each supervisor of elections must address immediately. Provided within this technical advisory are guidelines that clarify the requirements for meeting the minimum security standards of 1S-2.015 (5)(g), (k) and (n).

Background and Scope:

Florida's voting systems standards and certification program are recognized as the most stringent in the nation. Supplementing this rigorous certification process are the detailed security procedures that each county supervisor of elections must establish and follow. Indeed, the success of a certified voting system is largely dependant upon the security employed.

As a matter of practice, Florida's voting systems standards and certification program are reviewed by the Division's Bureau of Voting Systems Certification on a continuous basis. The Bureau recognizes that as technology evolves so must our security procedures surrounding the operations of our voting systems. As we identify new procedures and guidelines that are necessary, it is paramount that county Supervisors amend their security procedures.

In addition to the Division's ongoing internal examination of security procedures, we have recently reviewed the State of California's Voting Systems Technology Assessment Advisory Board's (VSTAAB) Security Analysis of the Diebold AccuBasic Interpreter and Ciber Laboratory's Source Code Review and Functional Testing reports. The Florida Division of Elections believes that potential system vulnerabilities identified in these reports can be addressed through enhanced security safeguards. In general, these recommendations are applicable to all types of election media including compact flashes, PCMCIA cards, memory packs, PEBs, and paper ballots. **This technical advisory therefore applies to all voting systems deployed in Florida.**

Note that the use of the word "procedure" within the context of this technical advisory means a macroscopic description of a process that defines the duties, responsibilities, and activities of an individual or a group of individuals. While explicit step-by-step task specific work instructions necessary for implementation are not required to be included in

your revised security procedures when submitted to the Division of Elections for approval, such instructions must be incorporated into your county's overall security plan to ensure the highest level of system protection.

Recommendations and Guidelines

Pre-election Steps for Voting Systems:

Threat model and mitigating strategy:

When developing a security procedure, one should determine the key elements within a system and develop threat models against those elements. For example, consider a threat model that consists of a "knowledge based" attack focused on a scanner memory card or any other type of election media. This "knowledge based" attack assumes that the security perimeter surrounding this media can be breached to allow unfettered access or that an internal party utilizes their position of responsibility to gain such access to the media. The mitigating strategy to defend against such an intrusion includes one or more security layers focused on election media accountability and chain of custody. Therefore, the following guidelines serve as the minimum criteria for evaluating compliance to this security procedure element as it relates to electronic media.

- 1) Regardless of electronic media type (memory packs, compact flash cards, PC Cards [aka PCMCIA cards], PEBs, voter card encoders, supervisor cards, and key cards), all such media shall be permanently identified with a unique identification (e.g., serial number).
 - a. The supervisor of elections shall create and maintain an inventory of all electronic media.
 - b. The supervisor of elections shall create a process and maintain a procedure for tracking the custody of electronic media from their storage location, through election coding, through the election process, to their final post-election disposition and return to storage. This electronic media must be given the same level of attention that one would give to official ballots.
 - c. The chain of custody must utilize two or more individuals to perform a check and verification check whenever a transfer of custody takes place.
- 2) The supervisor of elections shall create and maintain a secured location for storing the electronic media when not in use, for coding an election, for creating the election media, for transferring and installing the election media into the voting device, and for storing these devices once the election parameters are loaded.
 - a. No election media shall be left unattended or in an unsecured location once it has been coded for an election.
 - i. Where applicable, coded election media must be immediately loaded into the relevant voting device, logged, and made secure or must be placed in a secured and controlled environment and inventoried.
 - b. For each election, the supervisor of elections shall seal each election media in its relevant voting device or container utilizing one or more uniquely identified tamper-resistant or tamper-evident seals.
 - i. A combined master identification of the voting device, the election media, and the seal(s) must be created and maintained.

-
- ii. For election media that are device independent (e.g., PEBs, voter card encoders) these devices should be stored in a secured, sealed container and must also be identified on a master log.
 - c. The supervisor of elections shall create a process and maintain a procedure for tracking the custody of these voting devices once these devices are loaded with an election definition. These voting devices must be given the same level of attention that one would give to official ballots.
 - d. The chain of custody must utilize two or more individuals to perform a check and verification check whenever a transfer of custody takes place.
 - 3) The supervisor of elections shall have in place a recovery plan that is to be followed should there be any indication of a security breach in the accountability and chain of custody procedures. Any indication of a security breach must be confirmed by more than one individual.
 - 4) The supervisor of elections shall have a training plan for relevant election officials, staff, and temporary workers that address these security procedures and the relevant work instructions.

Transport of Ballots and/or Election Materials:

Threat model and Mitigation Strategy:

Consider a threat where a malicious entity wishes to gain access to a memory card or any type of election media. This could occur at any time prior to opening the polls and with the election media in any state (i.e., pre-election, set for election, or post-election.) The mitigating strategy to defend against such an invasion includes one or more security layers that again focus on accountability and chain of custody. Therefore, the following guidelines serve as the minimum criteria for evaluating compliance to this security procedure element.

- 1) The supervisor of elections shall create and maintain a secured location for storing and transporting voting devices once the election parameters are loaded. This shall include procedures that are to be used at locations outside the direct control of the supervisor of elections, such as overnight storage at a polling location.
 - a. For each election, the supervisor of elections shall create and maintain an inventory of these items for each storage location. These voting devices must be given the same level of attention that one would give to official ballots.
 - b. The chain of custody must utilize two or more individuals to perform a check and verification check whenever a transfer of custody takes place or where the voting devices have been left unattended for any length of time. Particular attention must be given to the integrity of the tamper-resistant or tamper-evident seals.
- 2) The supervisor of elections shall have in place a recovery plan that is to be followed should there be any indication of a security breach in the accountability and chain of custody procedures. The plan must also address inadvertent damage to any seals or accountability/chain of custody documentation errors. These plans must be developed in a manner that enhances public confidence in the security and integrity of the election. Any

indication of a security breach, documentation errors, or seal damage must be confirmed by more than one individual.

- 3) The supervisor of elections shall have a training plan for relevant election officials, staff, and temporary workers that address these security procedures and the relevant work instructions.

Election Access to Voting Systems:

Threat model and Mitigation Strategy:

Consider a threat model to optical scanners, DRE touchscreens, central count scanners, and the election management system; the success of which relies on a known vulnerability in an election department's security protocols. Under this condition, perimeter security may be compromised where access to the voting system relies on default passwords and encryption keys or where such items are not changed frequently. The obvious mitigating strategy to defend against such an intrusion includes immediately changing the default passwords and encryption keys and to develop a plan and process for changing the access control built on some time-based or event-based characteristic. Therefore, the following guidelines serve as the minimum criteria for evaluating compliance to this security procedure element.

- 1) The supervisor of elections shall have a procedure that ensures that default or vendor supplied passwords, encryption keys, etc. have been changed.
 - a. The supervisor of elections must maintain these access control keys/passwords in a secured and controlled environment. Who has access to these items must be delineated in the relevant position descriptions.
 - b. Changes to the encryption keys and passwords are at the discretion of the supervisor of elections, but it is advisable that this discretionary authority should not be delegated. However, the individual(s) that implement the change must have this "authorization to change" responsibility delineated within their position description(s). *(Note the distinction relative to describing who can authorize a change, who implements a change, and who has access but cannot change the passwords and encryption keys.)*
 - c. Where appropriate, the degree of access should be defined within each relevant position description and maintained at that level within the election management system and/or equipment. This applies where a voting system can limit an individual's access to certain menus, software modules, etc.
- 2) Access to any device, election media, or election management system that requires the use of an encryption key must be witnessed by one or more individuals authorized to use such information.
 - a. An access log should be developed and utilized.
- 3) The supervisor of elections shall have a training plan for relevant election officials, staff, and temporary workers that address these security procedures and the relevant work instructions.

Specific Authority: 101.015 F.S.

Rule: 1S-2.015 (5)(g), 1S-2.015 (5)(k), and 1S-2.015 (5)(n)

Security Analysis of the Diebold AccuBasic Interpreter

David Wagner David Jefferson Matt Bishop
Voting Systems Technology Assessment Advisory Board (VSTAAB)

with the assistance of:

Chris Karlof Naveen Sastry
University of California, Berkeley

February 14, 2006

1 Summary

This report summarizes the results of our review of some of the source code for the Diebold AV-OS optical scan (version 1.96.6) and the Diebold AV-TSx touchscreen (version 4.6.4) voting machines. The study was prompted by two issues: (1) the fact that AccuBasic scripts associated with the AV-OS and AV-TSx had not been subjected to thorough testing and review by the Independent Testing Authorities when they reviewed the rest of the code for those systems, and (2) concern over vulnerabilities demonstrated in the AV-OS optical scan system by Finnish investigator Harri Hursti in Leon County, FL. Mr. Hursti showed that it is possible for someone with access to a removable memory card used with the AV-OS system to modify scripts (small programs written in Diebold's proprietary AccuBasic language) that are stored on the card, and also to modify the vote counts stored on the card, in such a way that the tampering would affect the outcome of the election and not be detected by the subsequent canvass procedures.

The questions we addressed are these:

- What kinds of damage can a malicious person do to undermine an election if he can arbitrarily modify the contents of a memory card?
- How can the possibility of such attacks be neutralized or ameliorated?

The scope of our investigation was basically limited to the above questions. We did not do a comprehensive code review of the whole codebase, nor look at a very broad range of potential security issues. Instead, we concentrated attention to the AccuBasic scripting language, its compiler, its interpreter, and other code related to potential security vulnerabilities associated with the memory cards.

We found a number of security vulnerabilities, detailed below. Although the vulnerabilities are serious, they are all easily fixable. Moreover, until the bugs are fixed, the risks can be mitigated through appropriate use procedures. Therefore, we believe the problems as a whole are manageable.

Our findings regarding the scope of possible attacks on the AV-OS optical scan and AV-TSx touchscreen systems can be summarized as follows:

-
- *AccuBasic is a limited language:* The AccuBasic language itself is not a powerful programming language, but a very restricted one, narrowly tailored to one task: calculating and printing reports before and after an election. From a security point of view this is very desirable; minimal functionality generally means fewer opportunities for error or security vulnerability. In particular, *when its interpreter is properly implemented* (see below) an AccuBasic program cannot modify votes or ballot images; it can read vote counters (AV-OS) or ballot images (AV-TSx), but it cannot modify them.
 - *The AccuBasic interpreter is well-structured:* The code in the AccuBasic interpreters for both machines is clean, well-structured, and internally documented. We were able to understand it with little difficulty despite the lack of external documentation.
 - *Memory card attacks are a real threat:* We determined that anyone who has access to a memory card of the AV-OS, and can tamper it (i.e. modify its contents), and can have the modified cards used in a voting machine during election, can indeed modify the election results from that machine in a number of ways. The fact that the the results are incorrect cannot be detected except by a recount of the original paper ballots.
 - *Harri Hursti's attack does work:* Mr. Hursti's attack on the AV-OS is definitely real. He was indeed able to change the election results by doing nothing more than modifying the contents of a memory card. He needed no passwords, no cryptographic keys, and no access to any other part of the voting system, including the GEMS election management server.
 - *Interpreter bugs lead to another, more dangerous family of vulnerabilities:* However, there is another category of more serious vulnerabilities we discovered that go well beyond what Mr. Hursti demonstrated, and yet require no more access to the voting system than he had. These vulnerabilities are consequences of bugs—16 in all—in the implementation of the AccuBasic interpreter for the AV-OS. These bugs would have no effect at all in the absence of deliberate tampering, and would not be discovered by any amount of functionality testing; but they could allow an attacker to completely control the behavior of the AV-OS. An attacker could change vote totals, modify reports, change the names of candidates, change the races being voted on, or insert his own code into the running firmware of the machine.
 - *Successful attacks can only be detected by examining the paper ballots:* There would be no way to know that any of these attacks occurred; the canvass procedure would not detect any anomalies, and would just produce incorrect results. The only way to detect and correct the problem would be by recount of the original paper ballots, e.g. during the 1 percent manual recount.
 - *The bugs are classic, and can only be found by source code review:* Finding these bugs was only possible through close study of the source code. All of them are classic security flaws, including buffer overruns, array bounds violations, double-free errors, format string vulnerabilities, and several others. There may, of course, be additional bugs, or kinds of bugs, that we did not find.
 - *AV-TSx has potential cryptographic protection against memory card attacks:* A majority of the bugs in the AV-OS AccuBasic interpreter are also present in the interpreter for the AV-TSx touchscreen system. However, the AV-TSx touchscreen has an important protection that

the AV-OS optical scan does not: the key contents of its removable memory card, including the AccuBasic scripts, are digitally signed. Hence, if the cryptographic keys are managed properly (see next bullet), any tampering would be quickly detected and the attack would be unsuccessful. All of the attacks we describe, and Hursti's attack as well, would be foiled, because the memory card by itself would in effect be cryptographically tamperproof.

- *But the implementation of cryptographic protection is flawed:* There is a serious flaw in the key management of the crypto code that otherwise should protect the AV-TSx from memory card attacks. Unless election officials avail themselves of the option to create new cryptographic keys, the AV-TSx uses a default key. This key is hard-coded into the source code for the AV-TSx, which is poor security practice because, among other things, it means the same key is used in every such machine in the U.S. Worse, the particular default key in question was openly published two and a half years ago in a famous research paper, and is now known by anyone who follows election security, and can be found through Google. The result is that in any jurisdiction that uses the default keys rather than creating new ones, the digital signatures provide no protection at all.
- *All the bugs are easy to fix:* In spite of the fact that the bugs we have identified are very serious, all of them are very local and very easy to fix. In each case only a couple of lines of code need to be changed. It should take only a few hours to do the whole job for both the AV-OS and AV-TSx.
- *No use of high assurance development methods:* The AccuBasic interpreter does not appear to have been written using high-assurance development methodologies. It seems to have been written according to ordinary commercial practices. In the long run, if the interpreter remains part of the codebase, it and the rest of the codebase should be revised according to a more rigorous methodology that would, among other things, likely have prevented the bugs we found.
- *Interpreted code is contrary to standards:* Interpreted code in general is prohibited by the 2002 FEC Voluntary Voting System Standards, and also by the successor standard, the EAC's Voluntary Voting System Guidelines due to take effect in two years. In order for the Diebold software architecture to be in compliance, it would appear that either the AccuBasic language and interpreter have to be removed, or the standard will have to be changed.
- *Bugs detailed in confidential companion report:* In a companion report we have listed in great detail all of the bugs we identified, the lines at which they occur, and the threats they pose. Because that report contains Diebold proprietary information, and because it details exactly how to exploit the vulnerabilities we discovered, that report must be confidential.

Clearly there are serious security flaws in current state of the AV-OS and AV-TSx software. However, despite these serious vulnerabilities, we believe that the security issues are manageable by a reasonably careful combination of short- and long-term approaches. Here are our recommendations with regard to mitigation strategies.

In the short term, especially for local elections, the security problems related to AccuBasic and the memory cards might be managed according to guidelines such as these:

-
- *Strong control over access to memory cards for the AV-OS:* The AV-OS optical scan is vulnerable to both the Hursti attack and attacks based on the AccuBasic interpreter bugs we found. It would be safest if it is not widely used until these bugs are fixed, and until a modification is made to ensure that the Hursti attack is eliminated. But if the AV-OS is used, strong procedural safeguards should be implemented that prevent anyone from gaining unsupervised or undocumented access to a memory card, and these procedures should be maintained for the life of all cards. Such controls might include a dual-person rule (i.e. no one can be alone with a memory card); permanent serial numbers on memory cards along with chain-of custody documentation, so there is a paper trail to record who has access to which cards; numbered, tamper evident seals protecting access to the cards whenever they are out of control of county staff; and training of all personnel, including poll workers, regarding proper treatment of cards, and how to check for problems with the seals and record a problem. Any breach of control over a card should require that its contents be zeroed (in the presence of two people) before it is used again.
 - *Require generation of new crypto keys for the AV-TSx:* The AV-TSx is not vulnerable to any of these memory card attacks *provided* that the default cryptographic key used for signing the contents of the memory card is changed to a new, unguessable key and kept secure. If the key is changed then these threats are all eliminated, at least for the short term. If this is not done, however, then the AV-TSx is no more secure than the AV-OS.
 - *Control access to GEMS:* Access to GEMS should be tightly controlled. This is a good idea for many reasons, since a malicious person with access to GEMS can undermine the integrity of an election in many ways. In addition, in a TSx system, GEMS holds a copy of the cryptographic key used for signing the contents of the memory cards, and in both systems the GEMS server may hold master copies of the AccuBasic scripts loaded onto the memory cards.

In the longer term, one would want to consider a number of additional measures:

- *Fix bugs:* Certainly the bugs in the source code of the interpreters for both the AV-OS and AV-TSx should be corrected with all deliberate speed, the Hursti vulnerability should be fixed, and the code re-examined by independent experts to verify that it was properly done.
- *Defensive and high assurance programming methodology:* The source code of the interpreters should be revised to introduce systematic defensive programming practices and high assurance development methods. In particular, eliminate in the firmware, insofar as possible, any trust of the contents of the memory card.
- *Protect AccuBasic code from tampering:* The AccuBasic object code could be protected from tampering and modification, either by (a) storing AccuBasic object code on non-removable storage and treating it like firmware, or by (b) protecting AccuBasic object code from modification through the use of strong cryptography (particularly public-key signatures).
- *Don't store code on memory cards:* The architecture of the AV-OS and the AV-TSx could be changed so they do not store code on removable memory cards.

-
- *Remove interpreters and interpreted code:* The architecture of the AV-OS and the AV-TSx could be changed so they do not contain any interpreter or use any kind of interpreted code, in order to bring the codebase into compliance with standards.

2 Introduction

Scope of the study. This report summarizes the results of our review of the source code for the Diebold AV-OS optical scan (version 1.96.6) and the Diebold AV-TSx touchscreen (version 4.6.4) voting machines. This investigation, requested by the office of the California Secretary of State, was to evaluate security concerns raised by the use of AccuBasic scripts (programs) stored on removable memory cards in the two systems and offer options for their amelioration. The study was prompted by vulnerabilities demonstrated in the optical scan system by Finnish investigator Harri Hursti in Leon County, FL. Mr. Hursti showed that under certain circumstances it is possible for someone with access to a memory card to modify the scripts and modify the vote counts in a way that would not be detected by the subsequent canvass procedure, and would normally only be detectable by a recount of the paper ballots.

Our study does not constitute a comprehensive code review of the entire Diebold codebase. We had access to the full codebases for the AV-OS and AV-TSx, but we did not even attempt a comprehensive review of the entire codebase. Our attention was focused fairly narrowly on Diebold's proprietary AccuBasic scripting language, the compiler for that language, the interpreter for its object code, the AccuBasic scripts themselves, and the related protocols and procedures, both for the AV-OS (optical scan) and AV-TSx (touchscreen) voting systems.

In particular, we did not have the source code for the Diebold GEMS election management system, and our security evaluation does not cover GEMS at all. It is widely acknowledged that a malicious person with unsupervised access to GEMS, even without knowing the passwords, can compromise GEMS and the election it controls. This report does not address those threats, however.

Our analysis was based only on reading the source code we were given. We did not have access to a real running system (although we were able to compile and execute modified versions of the compiler and interpreter on a PC). Nor did we have any manuals or other documentation beyond that present in comments in the code itself. We had access to the source code for a period of approximately four weeks for this review.

The threat model. Different jurisdictions around the country have somewhat different procedures for conducting an election with the Diebold AV-OS and AV-TSx systems, but all include the following steps:

1. Before the election, the removable memory cards are initialized through the GEMS election management system with the appropriate election description information for the precinct the machine will be used in, and with the AccuBasic object code scripts to be used, and with other information detailed below.
2. The initialized cards are then inserted into the voting machines (optical scan or touchscreen); the compartment in which the card sits is locked and sealed with a tamper-evident seal of some kind.
3. The voting machine with its enclosed card is transported to the precinct poll site where it is stored over night (or longer) until the start of the election.
4. At the start of the election, a script on the card is used to print initial reports, including the Zero Report, which should indicate that all the vote counters are zero (in the AV-OS) and file of voted ballots is empty (in the AV-TSx).

5. All during election day, voted paper ballots are scanned and the appropriate counters on the removable memory card are incremented (AV-OS), or the voted ballots themselves are stored electronically on the memory card (AV-TSx), and electronic audit log records are appended to a file on the card.
6. At the end of election day, a script from the card is used to print final reports for the day, including vote totals.
7. Finally, one of two steps is taken, depending on the jurisdiction: either (a) the seal is broken and the memory card is removed and transported back to a central location for canvass using GEMS; or, (b) the entire voting machine is transported to the central location, where election officials break the seal, remove the memory card, and read its contents during the canvass.

The threats we are concerned about specifically involve modification of the contents of the memory card, especially the AccuBasic object code. In other words, somewhere along the line, in the procedure above, the attacker is able to get a memory card, arbitrarily modify its contents, and surreptitiously place it in a voting machine for use in an election, and do so without being immediately detected.

We assume the attacker's goal is either to change the election results undetected, or perhaps simply to disrupt the election (e.g. by causing voting machine crashes). We also assume that the attacker knows every detail of how the system works, and the procedural safeguards, and even has access to the manuals, documentation, and source code of the system. The attacker, therefore, is able to take advantage of bugs and vulnerabilities in the code. (It is standard to make these last assumptions, since it is almost impossible to keep code and related information secret from a determined attacker.)

We do not, however, assume that the attacker has any inside confederates, or has access to any passwords or cryptographic keys, or access to GEMS. We do not assume that the attacker has any access to paper ballots (AV-OS) or VVPAT (AV-TSx), nor even that he has any access to the voting system beyond the ability to insert a memory card undetected.

The process we followed. We were asked to perform a security review of the Diebold source code. As part of the review, we were provided access to the source code for the AV-OS and the AV-TSx machines. This included the source code for the AccuBasic compiler, for the AccuBasic interpreter in the AV-OS and the AccuBasic interpreter in the AV-TSx, for some AccuBasic scripts, and all other source code for the AV-OS and AV-TSx. There are two separate versions of the interpreter, one in the AV-OS and one in the AV-TSx; however, the two implementations are very similar.

We undertook a line-by-line analysis of the source code for the AV-OS AccuBasic interpreter. Three team members (Karlof, Sastry, and Wagner) read every line of source code carefully and checked for all types of security and reliability defects known to us. When we found a vulnerability in the AV-OS interpreter, we examined the corresponding portion of the AV-TSx interpreter to check whether the AV-TSx shared that same vulnerability.

After completing the line-by-line source code analysis, we applied a commercial static source code analysis tool to the AV-OS interpreter code. Code analysis tools perform an automated scan of the source code to identify potentially dangerous constructs. We obtained a copy of the Source Code Analyzer (SCA) tool, made by Fortify Software, Inc.; Fortify generously donated the tool to us for our use in this project at no cost, and we gratefully acknowledge their contribution. Two

of us (Bishop and Wagner) are members of Fortify Software’s Technical Advisory Board, and thus were already familiar with this tool. We manually inspected each of the warnings generated by the tool.

While our analysis uncovered several potential attacks on the system, we have not attempted to attack any working system. We performed our analysis mostly “on paper”; we did not have access to a genuine running system. We did, however, get a stubbed-out version of the code running on a PC, and were able to confirm that one of the attacks we discovered (the only one we tried) actually works.

In the end, we wrote our report in two parts. The *public* part is this document, which contains background, our findings and recommendations, and all of the explanatory information we have found to support them. The *confidential* part contains a detailed description of all of the bugs we found, the file names and line numbers where they occur, how they can be exploited, and what the consequences are. It is confidential because it contains both proprietary material and specific information about potential attacks on voting systems.

3 Background

3.1 Contents of the memory card

Both the AV-OS and AV-TSx systems use removable memory cards as key parts of their architectures. In both systems, the memory cards contain several kinds of information:

- the election description (a small database describing the races, candidates, parties, propositions, and ballot layout information for the current election);
- vote counters for every candidate and proposition on the ballot that store a count of the number of votes for that candidate (in the case of the AV-OS), or data records containing the cast ballot images (AV-TSx), along with various summary counters;
- byte-coded object programs (.abo files), which are normally created by writing scripts (programs) in the AccuBasic language and running them through the AccuBasic compiler.¹
- the internal electronic audit log
- an election mode field indicating whether the system containing the card is currently being used in a real election or not;
- a large number of other significant variables including strings, flags (for selecting options), various event counters, and other data describing the state of the election.

In fact, as far as we can tell, *the entire election-specific state of the voting machine* (the part that is retained *between* voting transactions) is stored on the memory card. It would take a much more comprehensive review of the software than we were able to conduct in order to verify this, but it appears to be the case.

¹AccuBasic object files (.abo files) are *normally* created by running AccuBasic programs through the compiler, i.e. that is the intent. But nothing prevents a programmer from directly writing .abo files, or modifying them, bypassing the AccuBasic language and the compiler entirely. Indeed, this is a route to several potential attacks. The AccuBasic interpreter makes no effort to verify that the AccuBasic object code has indeed been produced by the compiler.

All of this information on the memory cards is critical election information. If it is not properly managed, or if it is modified in any unauthorized way, the integrity of the entire election is possibly compromised. It is therefore vital, as everyone acknowledges, to maintain proper procedural control over the memory cards to prevent unauthorized tampering, and to treat them at all times during the election *with at least the same level of security as ballot boxes containing voted ballots*.

From one point of view, such an architecture makes good sense. In principle, it allows a memory card to be removed from a machine at almost any time (except during a short critical time window at the final completion of each vote transaction) without losing any votes or audit records, or any of the other context that has been accumulated. (Removal of a memory card during an election is procedurally forbidden under normal circumstances.) And it guarantees that when the memory card is removed at the end of the day, it contains *all* of the data needed for canvass, and for the resolution of most disputes, excepting only those that might depend on detailed forensic analysis.

Having all of the state on a removable memory card has a downside, however. It means an attacker with access to the card has potentially many other avenues of attack besides direct modification of the vote counts or the AccuBasic scripts; he can modify any other part of the election configuration or state as well. In our investigation, we did not attempt to enumerate all of these possibilities since it was clear that the only strong way to protect against all such attacks is to prevent any possibility of undetected tampering with the memory card in the first place.

When the AV-OS memory card is inserted into the AV-OS, it acts like an extension of main memory, and can be directly read and written via ordinary memory addressing, e.g. via variables and pointers. (Whether it actually is RAM, or is instead some other kind of memory-mapped storage device is not clear to us, but from a software point of view there is no difference.)

On the AV-TSx, however, the election state data is stored in a *file system* on the removable card. This means that the firmware cannot access it directly as main memory, but must use open/close/read/write calls to move data between files on the card and main memory. From a reliability and security point of view this is preferable to the architecture used on the AV-OS, since many kinds of common bugs (e.g. index or pointer bugs) can corrupt the data on a card that acts as main memory, whereas that is less likely for data packaged in a file system.

In the AV-OS, once the memory card is inserted into the voting machine, the byte-coded object programs become immediately executable by the AccuBasic interpreter in the firmware of the machine. However, on the AV-TSx the byte-coded object programs are cryptographically protected by the GEMS election management system. In effect, the GEMS server writes a sort of checksum² that depends on both the data and a secret cryptographic key to the memory card. When the memory card is inserted in an AV-TSx machine, the correctness of the checksum is validated and the machine refuses to enter election mode if the check fails³.

The cryptographic protection for the object code on the AV-TSx touchscreen machine is a significant improvement. It means that even if an attacker can get access to a memory card and modify the object code, unless he also has the cryptographic key to allow him to create a matching checksum for the modified object code, the checksum will not match when the card is inserted and the attack would be foiled. The integrity of the object code then boils down, for all practical purposes, to the secrecy of the cryptographic key (which we will discuss later).

²To be precise, it uses a cryptographic message authentication code (MAC).

³If the cryptographic message authentication code is invalid, a dialog box appears on the screen with the warning "Unable to load the election: the digital database signature does not match the expected value", and the machine does not enter election mode.

3.2 AccuBasic

The AccuBasic programming language is a Diebold-proprietary, limited-functionality *scripting language* (a kind of programming language). The *scripts* (programs) written in AccuBasic are intended to be used only for creating and printing reports on the printer units attached to the AV-OS or AV-TSx.

Once a script is written in AccuBasic (the *source code* version of the script), it is run through the AccuBasic *compiler*, which translates it into a form of *object code*. The object code is represented in another Diebold-proprietary language that seems to be unnamed but is generally referred to as *byte code* or an *.abo file*. It is the object code form of the scripts that is stored on the memory card, not the source form.

Normally all .abo files are produced in this way, i.e. by running AccuBasic source through the compiler. But it is important to understand that nothing prevents a programmer from bypassing the compiler and constructing a valid .abo file directly, or by editing an .abo file produced by the compiler. (Mr. Hursti did just that, modifying the portion of the script responsible for printing the zero report.) A .abo file produced in either of these nonstandard ways might not be producible by the compiler at all from any AccuBasic source file. However, they will still be executable by the interpreter without any error, and this fact can be the basis for powerful attacks that can take advantage of bugs in the interpreter. The AccuBasic interpreter makes no attempt to validate the .abo files, i.e. to ascertain that that they were in fact produced using the compiler.

The AccuBasic software for the AV-TSx is slightly different from that on the AV-OS. This is due primarily to the differences in the environment on the two systems. For example, the AV-TSx gets yes/no user input through the touchscreen, whereas the AV-OS gets it from physical buttons. Also, AV-OS memory cards contain vote counters only, whereas the AV-TSx cards store full ballot records. The memory card on the AV-OS is memory-mapped, whereas the same information is stored in a file system on the AV-TSx memory card. The AccuBasic interpreter for the AV-TSx is implemented in C++, whereas the interpreter in the AV-OS is written in C. The AV-OS interpreter contains 1838 lines of C code (not counting blank lines, comments, or global declarations), while the AV-TSx contains 2614 lines of C++ code (again, excluding blank lines, comments, and declarations). However, it is clear that the AccuBasic interpreter in the AV-TSx was originally just a translation from C to C++ of the one in the AV-OS, and they have subsequently diverged only slightly. The differences between the two AccuBasic interpreters are generally small enough that, except where noted, our generalizations about AccuBasic and its implementation apply equally to both versions.

AccuBasic is in one sense a general purpose language, in that it is able to do arbitrary numerical and string calculations.⁴ But in another sense, *when its interpreter is properly implemented*, it is a very restricted language in that, while it can *calculate* anything, it can only *control* a very limited part of the functionality of the voting machine. For example, an AccuBasic script can read the vote counters (or ballot images) and the election description from the memory card, and it can read a few other internal values as well (such as the date and time); but it cannot modify any of them. And it can invoke only a few functions from the rest of the codebase outside the interpreter, specifically, those needed for assembling information for, and for the printing of, reports on the machine's screen and printer. It is not possible (again, *when the AccuBasic interpreter is properly implemented*) for AccuBasic object code to:

⁴The language uses integer and string data types, and permits assignments, substring extraction and assignment, conditionals, loops, a limited number of defined subroutines, subroutine calls (without arguments), and recursion. It is theoretically capable of computing any computable function.

-
- modify the vote counts (AV-OS) or the ballot images (AV-TSx);
 - forge any votes or fail to record any votes;
 - modify the election description information; or
 - modify any paper ballots.

On the other hand, even when perfectly implemented, it is always possible for an erroneous or malicious AccuBasic script to:

- print false reports, or
- crash the voting machine (e.g., by going into an infinite loop).

These latter points are not flaws in the design of AccuBasic language or interpreter. Any other software, e.g. the machine's firmware, could have similar bugs. However, the fact that the scripts are on removable memory cards—and thus potentially exposed to tampering—makes these possibilities important. Mr. Hursti's attack on the AV-OS depended critically on his ability to modify the Zero Report script so that it falsely indicated that all counters were zero when in fact they were not. And in some jurisdictions, e.g. Florida, the reports printed by the AV-OS are the legal results of the election, so printing a false report amounts to falsifying the results of the election.

The intent of the AccuBasic language, compiler, and interpreter is that AccuBasic scripts should be usable *exclusively* for creating and printing reports on the voting machine's printer, without modifying the voting machine's behavior in any other way. With the exception of some serious bugs (described in our findings below) we found that this is indeed the case. In spite of its name, which is reminiscent of the powerful scripting language Visual Basic, we found that AccuBasic is a very limited, special purpose language; this is the right approach if one is to use an interpreted language at all.

Aside from the bugs (described below) the AccuBasic interpreters for both the AV-OS and AV-TSx are very well written and documented. We had no difficulty understanding the code and reviewing it.

4 Findings

Finding 1 *There are serious vulnerabilities in the AV-OS and AV-TSx interpreter that go beyond what was previously known. If a malicious individual gets unsupervised access to a memory card, he or she could potentially exploit these vulnerabilities to modify the electronic tallies at will, change the running code on these systems, and compromise the integrity of the election arbitrarily. (The original paper ballots for the AV-OS, of course, cannot be affected by tampering with the memory cards.)*

The AccuBasic interpreters, in both the AV-OS and AV-TSx, have a number of serious bugs—defects in the source code—that render the machines vulnerable to various attacks. (This goes well beyond what Mr. Hursti demonstrated; his attacks did not exploit any of these vulnerabilities.) These vulnerabilities would not affect the normal behavior of the machine, and would not be discovered during testing. But they could be exploited by an attacker with unsupervised access to a memory card. Many of these vulnerabilities are present in both the AV-OS and AV-TSx;

the AV-TSx code is basically a translation of the AV-OS code from C to C++, and most of the vulnerabilities were preserved in the translation.

The vulnerabilities arise because the AccuBasic interpreter “trusts” the contents of the AccuBasic object code (.abo files) stored on the memory card, and implicitly assumes that this AccuBasic object code has been produced by a legitimate Diebold AccuBasic compiler. As discussed earlier, this assumption is not necessarily justified. Anyone with unsupervised access to the AV-OS memory card could freely modify its contents, including the .abo file stored on the memory card. The same is true of the AV-TSx memory card, if the cryptographic keys are not updated from their default values (see Finding 4 below).

Types of vulnerabilities. The vulnerabilities include several instances of the classic buffer over-run vulnerability, as well as vulnerabilities with a similar effect. This kind of vulnerability would allow someone who could edit the AccuBasic object code on the memory card to completely control the behavior of the voting machine. The instant that the AccuBasic interpreter on the AV-OS or AV-TSx attempts to execute the malicious AccuBasic object code, the machine will be compromised.

Table 1 contains an overview of the 16 vulnerabilities we found in the AV-OS, and their impact. Also, Table 2 contains a similar overview of the 10 vulnerabilities we found in the AV-TSx. Note that we have excised any information that might help to exploit these vulnerabilities from those tables. We have relegated all such information to a separate Appendix, which contains additional detail: for each vulnerability, the Appendix lists the source code line number where the vulnerability appears, along with information about how the vulnerability might be exploited in the field.

These vulnerabilities were found primarily by line-by-line review of the source code, performed by three of us reading every line of the interpreter code together as a team. After we had completed a careful line-by-line security analysis, we then applied the Fortify Source Code Analyzer (SCA) tool and examined the warnings it produced. Given the care with which we performed the manual code review, we had not expected a static bug-finding tool to find any further bugs. Consistent with our expectations, the first warning we inspected from the tool referred to an exploitable security vulnerability we had already found. However, to our considerable surprise, the second warning from the tool turned out to reveal a vulnerability that we had missed as part of our manual code inspection (namely, Vulnerability V2). (The remainder of warnings we examined pointed to bugs and vulnerabilities that we had already found.)

In all cases the specific bugs we found are local and easy to fix. One concern, however, is that these are just the bugs *we* were able to find; there are quite possibly others we did not notice, and that automated bug-finding tools (which are always imperfect) would not notice either. Code review is difficult. It is hard to be confident that one has found all bugs (and indeed, our experience with the Fortify SCA tool highlighted this fact), and if we used another tool or if another person were to examine the code, they might find other vulnerabilities.

None of the vulnerabilities we found would have been found through standard testing, so testing is not the answer. This is a long-term problem with the use of interpreted code on removable memory cards, and with the failure to use defensive programming and other good security practices when implementing the interpreter.

These vulnerabilities have not been confirmed by verifying that they work against a full working system. (We did not have access to a running system.) We have used our best judgement to assess which bugs are likely to be exploitable, but it is possible that some bugs we classified as

vulnerabilities may in fact not be exploitable. Conversely, there may be other vulnerabilities that we failed to identify because of the lack of a working system.

To double-check our analysis, we chose one vulnerability more or less at random and verified that we were able to exploit it in a simulated test environment. We were able to compile and execute a slightly modified version of the AV-OS AccuBasic interpreter, as well as the AccuBasic compiler, on a PC. We then developed an example of AccuBasic object code (an .abo file) that would exploit this vulnerability. We verified that, when using the interpreter to interpret this object code on our PC, we were able to trigger a buffer overrun and successfully exploit the vulnerability. This provides partial confirmation of our analysis, but it is certainly not an authoritative test. We did not attempt to perform an exhaustive test of all 16 vulnerabilities.

Impact. The consequence of these vulnerabilities is that any person with unsupervised access to a memory card for sufficient time to modify it, or who is in a position to switch a malicious memory card for a good one, has the opportunity to completely compromise the integrity of the electronic tallies from the machine using that card.

Many of these vulnerabilities allow the attacker to seize control of the machine. In particular, they can be used to replace some of the software and the firmware on the machine with code of the attacker's choosing. At that point, the voting system is no longer running the code from the vendor, but is instead running illegitimate code from the attacker. Once the attacker can replace the running code of the machine, the attacker has full control over all operation of the machine. Some of the consequences of this kind of compromise could include:

- The attack could manipulate the electronic tallies in any way desired. These manipulations could be performed at any point during the day. They could be performed selectively, based on knowledge about running tallies during the day. For instance, the attack code could wait until the end of the day, look at the electronic tallies accumulated so far, and choose to modify them only if they are not consistent with the attacker's desired outcome.
- The attack could print fraudulent zero reports and summary reports to prevent detection.
- The attack could modify the contents of the memory card in any way, including tampering with the electronic vote counts and electronic ballot images stored on the card.
- The attack could erase all traces of the attack to prevent anyone from detecting the attack after the fact. For instance, once the attack code has gained control, it could overwrite the malicious AccuBasic object code (.abo file) stored on the memory card with legitimate AccuBasic object code, so that no amount of subsequent forensic investigation will uncover any evidence of the compromise.
- It is even conceivable that there is a way to exploit these vulnerabilities so that changes could persist from one election to another. For instance, if the firmware or software resident on the machine can be modified or updated by running code, then the attack might be able to modify the firmware or software in a permanent way, affecting future elections as well as the current election. In other words, these vulnerabilities mean that a procedural lapse in one election could potentially affect the integrity of a subsequent election. However, we would not be able to verify or refute this possibility without experimentation with real systems.

	Type	Impact
V1	Array bounds violation	Overwrite any memory address within $\pm 2^{15}$ bytes of the global context structure with a 2-byte value that the adversary has partial control over. Might allow attacker to inject malicious code and take complete control of the machine. Might allow overwriting vote counters.
V2	Format string vulnerability	Crash the machine; read the contents of memory within a narrow range
V3	Input validation error	Choose any location on the memory card and begin executing it as .abo code; could be used to conceal malicious .abo code in unexpected locations, or to crash the machine.
V4	Array bounds violation	Memory corruption; crash the machine.
V5	Double-free() vulnerability	Overwrite any desired 4-byte memory address with any desired 4-byte value. Allows attacker to inject malicious code and take complete control of the machine.
V6	Array bounds violation	Memory corruption: overwrite any memory address up to 2^{16} bytes after the global context structure with a 2-byte value that the adversary has no control over. Might allow overwriting vote counters.
V7	Buffer overrun	Memory corruption; crash the machine
V8	Buffer overrun, integer conversion bug	Memory corruption: overwrite up to 2^{15} consecutive bytes of memory starting at global context structure. Might allow attacker to inject malicious code and take complete control of the machine. Might allow overwriting vote counters. Information disclosure: read any memory location $\pm 2^{15}$ bytes away from global context structure. Crash the machine.
V9	Buffer underrun	Memory corruption: overwrite up to 2^{15} consecutive bytes of memory extending backwards from the global context structure. Might allow attacker to inject malicious code and take complete control of the machine. Might allow overwriting vote counters. Information disclosure: read any memory location within this window. Crash the machine.
V10	Buffer overrun	Overwrite return address on the stack. Allows attacker to inject malicious code and take complete control of the machine.
V11	Array bounds violation	Information disclosure: read from potentially any memory address. Crash the machine.
V12	Array bounds violation	Write any 2-byte value to any address up to 2^{16} bytes after the global context structure. Might allow attacker to inject malicious code and take complete control of the machine. Might allow overwriting vote counters.
V13	Array bounds violation	Information disclosure: Read any 2-byte value from any address up to 2^{16} bytes after the global context structure.
V14	Pointer arithmetic error	Crash machine. Could begin interpreting random memory locations as though they were .abo code.
V15	Unchecked string operation	Machine might crash or become unresponsive
V16	Unchecked string operation	Overwrite stack memory. Might allow attacker to inject malicious code and take complete control of the machine.

Table 1: 16 security vulnerabilities we found in the AV-OS.

	Type	Impact
W1	Array bounds violation	Overwrite any memory address with a 4-byte value that the adversary has partial control over. Allows attacker to inject malicious code and take complete control of the machine.
W3	Input validation error	Choose any memory location and begin executing it as .abo code; could be used to conceal malicious .abo code in unexpected locations, or to crash the machine.
W6	Array bounds violation	Overwrite any memory location with any desired value. Allows attacker to inject malicious code and take complete control of the machine.
W7	Buffer overrun	Memory corruption; crash the machine
W8	Buffer overrun, integer conversion bug	Corrupts memory until the machine crashes.
W10	Buffer overrun	Overwrite return address on the stack. Allows attacker to inject malicious code and take complete control of the machine.
W11	Array bounds violation	Information disclosure: read from potentially any memory address. Crash the machine.
W12	Array bounds violation	Writes any 4-byte value to any address. Allows attacker to inject malicious code and take complete control of the machine.
W13	Array bounds violation	Information disclosure: read a 4-byte value from any address.
W14	Pointer arithmetic error	Crash machine. Could begin interpreting random memory locations as though they were .abo code.

Table 2: 10 security vulnerabilities we found in the AV-TSx. Note that in many cases, the same vulnerability appears in both the AV-OS and AV-TSx interpreters, so we have used parallel numbering (e.g., the bug V6 in the AV-OS interpreter also appears in a very similar form as bug W6 in the AV-TSx interpreter).

-
- It is conceivable that the attack might be able to propagate from machine to machine, like a computer virus. For instance, if an uninfected memory card is inserted into an infected voting machine, then the compromised voting machine could replace the AccuBasic object code on that memory card with a malicious AccuBasic script. At that point, the memory card has been infected, and if it is ever inserted into a second uninfected machine, the second machine will become infected as soon as it runs the AccuBasic script.

It is difficult to confidently assess the magnitude of this risk without experimentation with real systems. That said, given our current understanding of how memory cards are used and our current understanding of the vulnerabilities⁵, we believe the risk of this kind of attack is low (at least in the near term). This kind of virus would only be able to spread through “promiscuous sharing” of memory cards, which means that propagation would probably be fairly slow. If typical practice is that memory cards are wiped clean before the election, programmed, sent to the polls, and then returned for reading at the GEMS central management system, then there does not seem to be much opportunity for one infected memory card to infect many machines.

- On the AV-TSx, the attack could print fraudulent VVPAT records. Since VVPAT records are considered the authoritative record during a recount, this might enable election fraud even if the VVPAT records are manually recounted. For instance, the attack could print extra VVPAT records during a quiet time when no voter is present (however, we expect that this might be noticed by poll workers, as the TSx printer is fairly noisy). As another example, when a voter is ready to print the VVPAT record, the attack code could print two copies of the voter’s VVPAT record and hope that the voter doesn’t notice. The attack might print duplicate VVPAT records only for voters who have voted for one particular candidate, thereby inflating the number of VVPAT records for that favored candidate. Alternatively, it might fail to print VVPAT records for voters who vote for a disfavored candidate (but of course, this could easily be detected voters who know to expect the machine to print a VVPAT record).

We believe the risk of false VVPAT records is lower than it might at first seem. See below for further discussion.

- The attack could affect the correct operation of the machine. For instance, on the AV-OS, it could turn off under- and over-vote notification. It could selectively disable over-vote notification for ballots that contain votes for a disfavored candidate, or selectively provide false over-vote notifications for ballots that contain votes for a favored candidate. On the AV-TSx, it could show the voter a wrong or incomplete list of candidates during vote selection; it could change selections between the time when they are initially selected and when they are shown on the summary screen; and it could selectively target a subset of voters, based on how they have voted or on other factors. Once the machine is running native code supplied by the attacker, its operation can be completely controlled by the attacker.

In addition, most of the bugs we found could be used to crash the machine. This might disenfranchise voters or cause long lines. These bugs could be used to selectively trigger a crash

⁵We have assumed as part of this analysis that the GEMS central management system, and TSx machines running in accumulator mode, do not execute AccuBasic scripts as part of reading memory cards. We were not able to verify or refute this assumption; however, we have no reason to believe it is inaccurate. Of course, if this assumption is inaccurate, our analysis of the risk would be affected.

only on some machines, in some geographic areas, or based on certain conditions, such as which candidate has received more votes. For instance, it would be possible to write a malicious AccuBasic script so that, when the operator prints a summary report at the end of the day, the script examines the vote counters and either crashes or continues operating normally according to which candidate is in the lead.

Unfortunately, the ability of malicious AccuBasic scripts to crash the machine is currently embedded in the architecture of the interpreter. Any infinite loop in the AccuBasic script immediately translates into an infinite loop in the interpreter (which causes the machine to stop responding, and is indistinguishable from a crash), and any infinite recursion in the AccuBasic script translates into stack overflow in the interpreter (which could corrupt stack memory or crash the machine).

The impact on the paper ballots (AV-OS). It is important to note that even in the worst case, the paper ballots cast using an AV-OS remain trustworthy; in no case can any of these vulnerabilities be used to tamper with the paper ballots themselves.

The impact on the VVPAT records (AV-TSx). As mentioned above, on the AV-TSx it is conceivable that these vulnerabilities might enable an attacker to print false VVPAT records. We assess the magnitude of this risk here. There are two cases:

- If the bugs are not fixed, and if proper cryptographic defenses are not adopted (see Finding 3), and if a malicious individual gains unsupervised access to the memory code:

In this case, it is hard to make any guarantees about the integrity of the VVPAT records. Attack code might be able to introduce fraudulent VVPAT records, compromising the integrity of both the electronic tallies and the paper records.

We were unable to identify any realistic scenario where this would enable an attacker to cause fraud on a large enough scale to affect the outcome of a typical election without being detected. If the attack tries to insert many fraudulent extra VVPAT records, then the 1% recount should detect that the VVPAT records do not match the electronic tallies or that many precincts have more VVPAT records than voters who signed in (on the roster sheets), which would reveal the presence of some kind of attack and (presumably) trigger further investigation. If the attack tries to defraud many voters by failing to print a valid VVPAT record, then we suspect at least some of these voters will notice and the attack is likely to be detected. Also, mounting a large-scale attack would appear to require tampering with many memory cards or with the GEMS election management system, which restricts the class of adversaries who would have the opportunity to mount such an attack.

Nonetheless, if such an attack is detected, it may be difficult to decide how to recover from the attack. In this scenario, both the electronic tallies and the paper records are untrustworthy, so in the worst case the only recourse may be to hold another election.

- If the bugs are fixed:

In this case, we do not see any realistic threat to the integrity of the VVPAT records.

In principle, if a malicious individual is able to introduce a malicious AccuBasic script, one might imagine a possible attack vector where the AccuBasic code prints false VVPAT records. However, in practice we do not see any viable threat here. AccuBasic scripts do have the capability to print to the AV-TSx printer, and this printer is shared for both printing reports (e.g., the zero tape, the summary report) during poll opening/closing, and for printing VVPAT records during the election. In theory, one might be able to envision a malicious AccuBasic script that, after it finishes printing the zero tape, continues running, waits some period of time, and then prints some text designed to look like a VVPAT record in hopes that this will be spooled into the security canister along with other VVPAT records. In practice, we believe that poll workers are unlikely to be fooled by this. As far as we can tell, the AV-TSx is single-threaded, so if the AccuBasic script does not relinquish control, the TSx will not show a startup screen welcoming voters to begin voting. It does not seem particularly likely that a poll worker would print and tear off a zero tape, feed the paper into the security canister, walk away before the machine has displayed a welcome screen, and fail to notice the machine printing and scrolling the tape into the security canister when there is no voter present. It is hard to imagine how this could be used for any kind of large-scale attack without being detected in at least some fraction of the polling places where the attack occurs. Therefore, we consider this risk to be minimal, if the bugs in the AV-TSx AccuBasic interpreter are fixed.

Finding 2 *Everything we saw in the source code is consistent with Harri Hursti's attack on the AV-OS.*

Our analysis of the source code is consistent with Harri Hursti's findings that (a) the AccuBasic script on the AV-OS memory card can be replaced with a malicious script, (b) the vote counters on the AV-OS memory card can be tampered with and set to non-zero values, and (c) it is possible to use a malicious AccuBasic script to conceal this tampering by printing fraudulent zero reports or summary reports. Our source analysis confirmed that a malicious AccuBasic script is able to print to the printer (on both the AV-OS and the AV-TSx), display messages on the LCD display (on the AV-OS), and prompt for user responses (on the AV-OS). Our analysis also confirmed that the AV-OS fails to check that the vote counters are zero at the start of election day. We also confirmed that the AV-OS source code has numerous places where it manipulates vote counters as 16-bit values without first checking them for overflow, so that if more than 65535 votes are cast, then the vote counters will wrap around and start counting up from 0 again. (It is a feature of 16-bit unsigned computer arithmetic that large positive numbers just less than 65536 are effectively the same as small negative numbers)⁶. There is little doubt in our minds that Hursti's findings about the AV-OS are accurate. Even if the bugs we found in the AccuBasic interpreter are fixed, Hursti's attacks will remain possible.

⁶We discovered that the code does contain a check to ensure that it will not accept more than 65535 ballots. On the surface, that might appear adequate to rule out the possibility of arithmetic overflow. However, as Hursti's attack demonstrates, the existing check is not, in fact, adequate: if the vote counter started out at some non-zero value, then it is possible for the counter to wrap around after counting only a few ballots. This is a good example of the need for defensive programming. If code had been written to check for wrap-around immediately before every arithmetic operation on any vote counter, Hursti's technique of loading the vote counter with a large number just less than 65536 would not have worked.

The AV-TSx also appears to be at risk for similar attacks. The AV-TSx memory card also contains an AccuBasic script and appears to be vulnerable to similar kinds of tampering, unless the cryptographic keys have been updated from their default values (see below for a discussion).

Finding 3 *The AV-TSx (but not the AV-OS) contains cryptography designed to protect the contents of the AV-TSx memory card from modification while it is in transit. This mechanism appears to be an acceptable way to protect AccuBasic scripts from tampering while the memory card is in transit, assuming election officials update the cryptographic keys on every AV-TSx machine.*

The AV-TSx uses a cryptographic message authentication code (MAC), which ensures that it is infeasible for anyone who does not know the secret cryptographic key to tamper with the data stored on the memory card. The use of the cryptographic MAC in the AV-TSx appears to be an acceptable way to protect AccuBasic object code (.abo files) from tampering while the memory card is in transit, provided that election officials update the cryptographic keys on every AV-TSx. On the other hand, if the cryptographic keys are not updated, then the cryptographic mechanism does not protect against tampering with the contents of the memory card, for the following reasons.

The AV-TSx contains a default set of cryptographic keys. There is a procedure that election officials can use to change the keys stored on any particular AV-TSx machine. However, if this procedure is not performed on an AV-TSx machine, then that AV-TSx continues to use its default keys.

The default keys provide no security. They appear to be the same for all TSx machines in the nation, and in fact were discovered and published two and a half years ago (see Finding 4 below). Unfortunately, if no special steps are taken, the AV-TSx silently uses these insecure keys, without providing any warning of the dangers. Therefore, election officials will need to choose a new key for the county and update the keys on every AV-TSx machine themselves. Fortunately, there is a process for updating the keys on the AV-TSx by inserting a special smartcard into the AV-TSx machine.

So long as this process is followed, the cryptographic message authentication code (MAC) should provide acceptable security against tampering⁷. Because the AccuBasic script (.abo file) is stored on the memory card, the cryptography protects the AccuBasic script from being modified. If the cryptographic keys have been set properly, this defends against attacks like Harri Hursti's against the TSx: it prevents a malicious individual from *successfully* tampering with the AccuBasic script or the ballots stored on the memory card, even if the individual has somehow gained unsupervised access to the memory card, because the cryptographic check built in to the TSx firmware will fail and the TSx will *print a warning message and refuse to proceed further*.

The cryptographic MAC on the TSx appears to cover almost everything stored on the memory card data file. It covers election parameters, vote counters, the AccuBasic script (.abo file), and some other configuration data. The only exceptions we are aware of is that the file version number and the election serial number do not appear to be covered by the cryptographic MAC or by any checksum. These exceptions seem to be harmless.

In effect, the cryptography acts as the electronic equivalent of a tamper-resistant seal. If the contents of the memory card is tampered with, the cryptography will reveal this fact.

⁷We assume that the cryptographic keys are not stored on the memory card, but are stored on non-removable storage. We were not able to verify this assumption from the source code alone, but we have no reason to believe otherwise.

We stress that, like a tamper-resistant seal, the cryptography *only* defends against tampering while the memory card is in transit. The cryptography does *not* protect against tampering with AccuBasic scripts while they are stored on the GEMS server. In the Diebold system, the cryptographic protection is applied by the GEMS server when the memory card is initialized. The GEMS server stores the cryptographic keys and uses them to compute the cryptographic MAC when initializing a memory card; later, the AV-TSx uses its own copy of the keys to check the validity of the MAC. Of course, anyone who knows the cryptographic key can change the contents of the card and re-compute the MAC appropriately. This means that anyone with access to the GEMS server will have all the information needed to make undetected changes to AV-TSx memory cards. Also, AccuBasic scripts (.abo files) are stored on the GEMS server and downloaded onto memory cards as needed. If the copy of the .abo files stored on the GEMS server were corrupted or replaced, then this could affect every AV-OS machine and every AV-TSx machine in the county. In other words, if the operator of the GEMS server is malicious, or if any untrusted individual gains access to the GEMS server, all of the machines in the county could be compromised. The AV-TSx cryptography provides no defense against this threat; instead, it must be prevented by carefully guarding access to the GEMS server.

The cryptographic algorithm used in the AV-TSx, while perhaps not ideal, appears to be adequate for its purpose. The AV-TSx uses the following MAC algorithm:

$$F_k(x) = \text{AES}_k(\text{MD5}(x)),$$

where $\text{AES}_k(\cdot)$ denotes AES-ECB encryption of a 128-bit value under key k . This choice of MAC algorithm is probably not what any cryptographer would select today, but it appears to be adequate. In August 2004, cryptographers discovered a way to find collisions in MD5, which prompted many cryptographers to suggest using some other hash algorithm in new systems. Fortunately, these collision attacks do not appear to endanger the way that AV-TSx uses its MAC, because chosen-plaintext attacks do not appear to pose a realistic threat. In contrast, the discovery of second pre-image attacks on MD5 would probably suffice to break the AV-TSx MAC algorithm, but fortunately no practical second pre-image attacks on MD5 are known. Consequently, given our current knowledge, the AV-TSx MAC appears to be acceptable.

In the long run, it would probably make sense to migrate to a more robust MAC algorithm (e.g., AES-CMAC). Even better, a cryptographic public-key signature (e.g., RSA, DSA) would appear to be ideal for this task. With the current scheme, anyone who can gain access to and reverse-engineer an AV-TSx machine can recover the cryptographic key and attack the other memory cards in the same county; while a public-key signature would eliminate this risk. Nonetheless, for present purposes the current scheme appears to be strong enough that it is not the weakest point in the system.

Finding 4 *The AV-TSx contains default cryptographic keys that are hard-coded into the source code and that are the same for every AV-TSx machine in the nation. One of these keys was disclosed publicly in July, 2003, yet it remains present in the source code to this day.*

We mentioned above that the AV-TSx contains a set of default keys that are used if the cryptographic keys have not been explicitly updated. We found that these default keys are hard-coded in the source code and are the same for every AV-TSx machine in the nation.

The presence of hard-coded keys in the TS was first disclosed in a famous scientific paper by Kohno, Stubblefield, Rubin, and Wallach in July, 2003. Their paper also revealed the value of the

key—namely, F2654hD4—to the public. Subsequent reports from Doug Jones revealed that this design defect dates back to November, 1997, when he discovered the same hard-coded key and reported its presence to the vendor. These authors pointed out that use of a hard-coded key that is the same for all machines is very poor practice and opens up serious risks. It would be like a bank using the same PIN code for every ATM card they issued; if this PIN code ever became known, the exposure could be tremendous. It had been our understanding that all of the vulnerabilities found in those investigations two years ago had been addressed. It is hard to imagine any justification for continuing to use this key after it had been compromised and revealed to the public. This is a serious lapse that we find hard to understand considering how widely publicized this vulnerability was.

This also illustrates the reason that cryptographers uniformly recommend against hard-coded keys. If those keys are ever compromised or leaked, the compromise can affect every machine ever manufactured, and it can be difficult to change the key on every affected machine.

The AV-TSx would be more secure if it were changed to avoid use of default keys, i.e. if election officials were *required* to generate and load a county-specific cryptographic key onto the AV-TSx before its first use, and if the AV-TSx were to refuse to enter *election mode* if no key has ever been loaded.

Finding 5 *The AV-OS stores the four-digit supervisor PIN on the memory card. The PIN is stored in an obfuscated format, but this obfuscation offers limited protection due to its reliance on hard-coded magic constants in the source code.*

On the AV-OS, the four-digit PIN is derived as a specific function of a field stored on the memory card and of some constant values that are hard-coded into the source code. These magic constants are the same for every AV-OS machine across the nation, which is the rough equivalent of the hard-coded keys found in the AV-TSx. Thus, the AV-OS contains a design defect that is roughly similar to one in the AV-TSx.

Anyone with access to the AV-OS source code can learn these magic constants. Likewise, anyone who has unsupervised access to an AV-OS machine and the ability to perform reverse engineering could learn these magic constants. Once the magic constants are known, anyone who gains access to a memory card can read its contents and predict its four-digit PIN. Likewise, if they had unsupervised access to the memory card, they could set the four-digit PIN to any desired value by setting the field stored on the memory card appropriately. The use of the same magic constant values for every AV-OS machine in existence poses the risk that, if these constant values are ever disclosed, the security of the PIN protection would be undermined.

At present, we believe the security risks of this design misfeature are probably minor and limited in extent, because even knowledge of the PIN only provides a limited degree of additional access. There are worse things that an individual could do if she gained unsupervised access to an AV-OS memory card. Nonetheless, we caution election administrators not to place too much reliance on the four-digit PIN on the AV-OS.

Finding 6 *The AccuBasic interpreter was fairly cleanly structured and was organized in a way that made the source code very easy to read.*

The source code for the AccuBasic interpreter was written in a way that made it easy for us to understand its intent and operation and analyze its security properties. The code was split into

many small functions whose purpose was clear and that performed one simple operation. There were comments explaining the purpose of each function and explaining tricky parts of the code. The clarity of the interpreter source code was about as good as any commercial code we have ever reviewed.

The interpreter is structured as a recursive descent parser, so that the program's call stack mirrors the stack of the associated context-free automaton. In addition, there is a global variable holding the global interpreter context: e.g., AccuBasic registers, AccuBasic variables, and various loop indexes. This was a reasonably elegant way to structure the implementation.

There were some ways that the implementation could have been improved. The code didn't use defensive programming, which would have helped tremendously to harden it against many malicious attacks. Also, the source code didn't document the relevant program invariants and pre/post-conditions. We were forced to work these out by hand (e.g., that certain parameters were never NULL, that the global string register would never contain a string more than 255 bytes long, and so on), and it would have helped if these had been documented in the source code. Nonetheless, on the whole the interpreter source code was structured in a way that simplified the source code review task.

Finding 7 *The AccuBasic language is not a general-purpose system; it is narrowly tailored for its purpose.*

The AccuBasic language is *not* a full, general-purpose scripting language in the same category as, say, Visual Basic, in spite of the similarity of names. Instead, it is very modest in scope, with strongly circumscribed capabilities. If you are going to use an interpreted language at all in a context where security is important, this is the right way to do: one should include only the absolute minimum functionality in the language necessary to do the job it is designed for, and AccuBasic seems to meet that goal. In particular, we note that:

- AccuBasic is computationally complete in the sense that it can *compute* anything, but its interactions with the rest of the codebase are very limited. The parts of the firmware and operating system that it can invoke makes it basically useful *only* for printing reports, which is the intent.
- The AccuBasic interpreter cannot invoke most of the functions available in the firmware. It cannot read or write memory outside the its own stack. It can only invoke a handful of benign services necessary for its report-writing function, e.g. reading (but not writing) the vote totals or ballot file, accepting yes/no input from the user, writing to the printer, LCD screen, or touchscreen, appending an event to the audit log file, and reading the date and time.
- In particular, the AccuBasic interpreter has only read-only access to the vote counters or ballot file, so that AccuBasic scripts can construct reports from them, but cannot modify them.

In the short, the design of the AccuBasic language appears to us to be appropriate for its purpose.

Finding 8 *The AccuBasic interpreter cannot be invoked while the AV-OS or AV-TSx are executing the core election functionality, i.e., while they are accepting votes during the middle of election day.*

The AV-OS. We determined the AV-OS does not invoke the interpreter during the tallying of live election ballots. The AV-OS invokes the interpreter during pre-election procedures, such as printing test ballot zero reports and tallies, printing election zero reports, and printing labels for duplicate memory cards and audit reports. The AV-OS also invokes the interpreter to print post-election reports after the “ender” card is read.

The AV-TSx. We determined the AV-TSx does not invoke the interpreter while it is in “election” mode. The AV-TSx can invoke the interpreter under five circumstances:

1. Printing a zero report on machine initialization.
2. The “Print Election Results” button on the pre-election menu page for printing pre-election test results.
3. Printing election totals after a poll worker presses the “End Voting” button on the election menu page.
4. The “Print Election Results” button on the post-election menu page.
5. The “Print Results” button on the the accumulator menu page.

None of these can occur during the middle of the day while the TSx is in the process of interacting with voters and accepting votes.

These observations are also positive design points. The interpreter is not only very limited in its functionality, but it is very limited in the window of time during an election that it runs, which is what one wants when security is important.

Finding 9 *The AccuBasic interpreter does not appear to have been written using high-assurance software development methodologies.*

The AccuBasic interpreter appeared to be written using commercial standards of software development. This means it is not high-assurance software, nor was it developed following high-assurance methodologies.

High-assurance methods are often used for software systems where security is of utmost importance, most notably for military applications (e.g., software used to process classified documents). At a high level, these methods are similar to those used to build safety-critical software systems, where failure of the software can lead to loss of life (e.g., software found in avionics control systems, nuclear reactors, manned space flight, train control systems, automotive braking systems, and other similar settings).

In high assurance software development, one first determines explicitly what requirements the software and/or system must meet. One then designs the system, demonstrating throughout that the design meets the requirements. The method used to demonstrate this depends upon the degree of assurance desired. One then implements the system, and again justifies that the implementation meets the design. Indeed, one should be able to point to each requirement and show *exactly* what code is present as a result of that requirement. Finally, the operating instructions and procedures for the system and software must also meet the requirements.

We saw no evidence that the AccuBasic interpreter was developed in this way. Indeed, the problems we found argue against it. We should note that we did not see *anything* beyond the code—no