Software Review and Security Analysis of the Diebold Voting Machine Software

Supplemental Report

David Gainey, Michael Gerke, and Alec Yasinsac

Security and Assurance in Information Technology Laboratory
Florida State University
Tallahassee, Florida

August 10, 2007

For the Florida Department of State

Software Review and Security Analysis of the Diebold Voting Machine Software

Supplemental Report

1 Introduction

On May 14th 2007, the Florida Department of State (FLDoS) commissioned an independent expert review of Diebold Voting System software version 1.96.8, by a team led by Florida State University's (FSU) Security and Assurance in Information Technology (SAIT) Laboratory [1]. The team issued its final report on July 27, 2007 [2] (hereafter referred to as the SAIT Diebold Report without citation). Shortly thereafter, the Florida Secretary of State (FLSoS) declined to certify the system that employed software version 1.96.8 as submitted and required Diebold to make substantive repairs [3] in order to continue the certification process.

On August 7th, FLDoS asked SAIT Laboratory to conduct a supplemental review of the Diebold software version 1.96.9 that the vendor submitted through the Independent Test Authority, in response to the FLSoS letter. The request's exclusive goal is to determine if the specific, required software repairs were accomplished. This report is the culmination of the requested supplemental review.

1.1 The Analysis' Scope

The scope of the investigation was exclusively to determine if the four required repairs listed on the first attachment page in [3] are evident and complete in version 1.96.9. The four flaws identified for required repairs are:

1. The Signature Flaw [3, par. 3.5]
2. Attacker Can Hide Preloaded Votes [3, par. 3.8.1.4]
3. AccuBasic Scripts Can Be Misused [3, par. 3.9.3] (noted as paragraph 3.9.1 in [3]).
4. Unchecked String Operation: Allows Overwrite of Stack Memory [3, paragraph 3.9.5]

1.2 Security Disclaimer

This report reflects the narrow investigative scope requested by FLDoS. These results are not comprehensive in any sense, nor is this report an endorsement of the system's overall security. We examined only a small subset of the flaws from the SAIT Diebold Report. All other flaws identified in that report remain in the code base, including vulnerability to a sleepover attack that may allow an intruder to manipulate vote computation or worse. Significant, critical vulnerability remains in this code base independent of repairs documented in this report.

Our exclusive purpose is to provide a technical assessment regarding four specific flaws. We defer interpretation and application of these technical results to FLDoS.

1.3 The Review Process

As a supplemental review, the basic terms of the SAIT Diebold Report also apply to this document, including all definitions, disclaimers, and findings that are not specifically addressed herein.

Upon receipt of the new code version from FLDoS, the team electronically compared the modified firmware version (1.96.9) to the version reviewed in the SAIT Diebold Report (1.96.8). Our review focused on the detected code differences, which included approximately two hundred lines of code, including comments. This small code volume facilitated quick response and increases confidence in our results.

With the code differences in hand, we reviewed each required change in [3] and compared it to the code differences to determine if they impacted a required change. Our findings below detail instances

where the changes impacted the required changes and further report our judgment regarding the technical impact that the accumulated changes have on the required flaws overall.

1.4 The Software Review Team

David Gainey is a computer science graduate student, a member of SAIT Laboratory, and is a member of the technical staff at the Florida State University Office of Technology Integration. He was a member of the original SAIT Diebold review team [2] and has evaluated a variety of voting system software products.

Michael Gerke is a computer science graduate student and a member of SAIT Laboratory at Florida State University. He is also presently employed by the Florida Department of State, Division of Elections. He was a member of the original SAIT Diebold review team [2] and has evaluated a variety of voting system software products.

Alec Yasinsac is an Associate Professor of Computer Science at Florida State University, a co-Director of SAIT Laboratory, and is the lead Principal Investigator on this project. He also led the SAIT teams that conducted the Florida Congressional District Thirteen voting system software review [4] and the Diebold review [2] that this project supplements.

1.5 Limits of this Study

1.5.1 Laboratory Results

There is always risk in conducting laboratory analysis for systems that operate outside the laboratory environment. At best, laboratory analysis presents a clinical perspective, often allowing scientists and investigators to isolate issues that cannot be economically evaluated during normal testing, operational testing, or during normal operations, but does not necessarily accurately capture more broad interactions.

Computer software has many properties that lend it favorably to laboratory analysis. For example, software is naturally created in a laboratory environment where programmers intentionally isolate specified functionality into their coding assignments. Subsequent laboratory testing and demonstration is often used to convince purchasers that the software meets its functional and foundational requirements. Thus, laboratory analysis is inherent, and well-understood, in the software process.

Some have questioned the applicability of laboratory analysis with regard to voting system software. However, there is no doubt that flaws identified in the SAIT Diebold Report exist and are technically exploitable, as is reflected in the serious response by the FLSoS [3]. Moreover, the SAIT Diebold Report discussed some common elections procedures and scientifically identified prerequisites and mitigations of this vulnerability where it exists.

Unfortunately, there is no clear existing procedure for testing voting system exploits in a non-laboratory setting. Because election integrity is sacrosanct, no reputable scientist or election official would engage a realistic security test in a real election. Worse yet, there is no way to reliably determine the likelihood that software vulnerability prerequisites are met even in the face of rigorous election procedures.

There are many systems that share this testing-limited property; these systems are sometimes termed “moon shot” systems. Fortunately, there is a coding approach that is suitable for critical, testing-limited systems; this is termed “high assurance” development. Since voting systems cannot be subjected to dangerous security analysis while in operation, they should be developed for high assurance. Until this occurs, elections officials face an unnecessarily high risk and they must exercise significantly expanded election security procedures to mitigate known and unknown software vulnerability.

1.5.2 Completeness Limitations

Any study involving systems and source code of the complexity of the Diebold Optical Scan system raises questions of completeness: could the investigators have missed problems? In this report, we document our efforts and assumptions to allow others to evaluate the thoroughness of our study.

Moreover, it is well-known that testing is an inherently incomplete process and that no testing process can guarantee absence of flaws. Regarding even the isolated flaws that we evaluated, we cannot guarantee that the changes or repairs that removed, mitigated, or reduced these flaws did not cause regression faults. While we did look for regression faults, it is possible that undetected regression faults may be more dangerous than the flaws that they fixed. Thus, we do not offer guarantees, but only our best professional and academic opinions.

Our conclusions were guided solely by the source code examined. We did not generate binaries for this source code to compare against a verified build. If this source does not precisely match that used in executing modules, our results may not apply.

2 Findings

We group our findings by the flaw that they describe. Where flaws overlap, we cross-reference to ensure consistency and to provide context to related flaws.

The Team's primary findings are that one of the four targeted flaws is fixed and the three other flaws are significantly improved in the revision. As an example of what we mean by "significantly improved", one of the two identified buffer overflow flaws was fixed, while the remaining buffer overflow flaw has no known exploit.

2.1 The Signature Flaw

As described in the SAIT Diebold Report,

> *... the hand-coded RSA signature verification is insecure and signatures generated with the implemented method can be forged.*

The original code applies a standard SHA1[1] hash to efficiently protect data integrity, then applies an RSA public key signature [5] across the hash value to guarantee its authenticity. This is a commonly accepted practice and, absent two subtle implementation miscues, it can provide strong authenticity assurances. The flaw occurred in the previous version because the RSA signature is longer than the SHA1 hash and the length difference was not properly handled[2], allowing an attacker to sign, or forge a signature on, an arbitrary message without knowing the signing key.

The crux of the problem was that when the signature was verified, only the SHA1 part of the signature was checked, allowing an astute attacker to leverage the unchecked bits to generate effective forgeries. The attack is facilitated because the implementation used another commonly accepted practice of selecting a key exponent of three (3), which was recently shown to expose RSA signatures to some subtle attacks.

In the revision, the vendor corrected the signature verification by modifying the code to verify the entire signature. They extended their repair by incorporating padding into the signature process in the commonly accepted way (appending the hash algorithm, etc.), so the process now controls and checks the entire signature content.

This algorithm fixes the signature flaw, thus preventing forgeries. We analyzed the code in some detail and it appears to be properly implemented.

---

[1] See FIPS PUB 180-1

[2] http://nvd.nist.gov/nvd.cfm?cvename=CVE-2006-4339

2.2 Attacker Can Hide Preloaded Votes

This original flaw, highlighted by the Hursti attack [6], involves improper counter management. Specifically, variables used to count votes were not properly protected and verified, allowing an attacker to preload votes by manipulating the counters on the memory card. Hursti's attack involved balancing incremented vote counters with other decremented (logically negative value) counters. In the modified code, the vendor implemented new counter routines that prevent counter faults.

The class of preloaded vote attacks identified under this flaw in the SAIT Diebold Report did not involve functional counter operation. Rather, these issues concern the fact that vote count fields on the memory card are not integrity protected. This allowed an attacker to edit fields to change votes with the only technological challenge being to appropriately set error detecting checksum values.

The vulnerability (as paraphrased from paragraph 3.8.1.4 of the SAIT Diebold Report) was:

1. Prepare a memory card with preloaded votes and insert it into the terminal after the zero report is printed. The modified count would be shown on the terminal's display.
2. Extract a memory card from the terminal after several votes had been registered, modify the counters to redistribute votes but also to maintain the same total vote count, and reinsert the memory card in the terminal if an attacker can gain suitable access to the terminal.
3. Prepared a memory card with a predetermined vote count and wait until the number of votes entered on the card were normally registered on the terminal. At that point, the attacker could replace the official memory card with the forged card and the terminal's display would reflect the correct, expected number of votes cast and the precinct count printout would show the injected votes.
4. Construct an interpreter script that disregards the counters on the memory card and prints the preloaded counts.

In the 1.96.9 revision, the subject ballot definition file, including the counter fields, is still not integrity protected. However, an attacker's ability to remove and reinsert a rogue memory card into the terminal is limited as a side-effect of the AccuBasic script signature repair (Section 2.1).

Under normal operations, card removal results in a properly reported system failure and forces a user initiated system shutdown. Since the script verification status is stored in volatile memory, card removal and reinsertion normally forces signature re-verification at the next script access, thus a rogue card would be detected when results were printed unless the card's script were properly signed.

Since the signature flaw repair now prevents signature forgery, this limits an attacker to: (1) Re-use a script from a properly signed, previously used memory card to create a rogue memory card or (2) Re-use the existing card after modifying the ballot definition file content, which requires a significantly longer attack-time terminal access period, or (3) Bypass the card removal shutdown process. We consider the first to be the most likely attack avenue and discuss the third further in Section 2.3 below.

We find that this flaw is significantly improved.

2.3 AccuBasic Scripts Can Be Misused

Unlike the ballot definition information on the removable memory cards, the AccuBasic script[3] is protected by the RSA/SHA1 signature. Thus, repairing the signature (see Section 2.1 above) reduces AccuBasic vulnerability. Signature verification now prevents an attacker from routinely inserting a rogue memory card with a custom script during terminal supervision gaps within the voting period.

However, due to the protocol applied to access the memory card, the vulnerability is not completely eliminated by the signature repair. The signature on the memory card script is only checked when the

[3] AccuBasic scripts are stored as ".abo" files in their source format. Since there is no file system on the memory cards, we refer to these programs as "scripts" when referring to them on the memory card.

script is first exercised, for example when the zero tape is printed. As noted in Section 2.2 above, under normal operations, card removal results in a properly reported system failure and a resulting user-initiated system shutdown. Since the script verification status is stored in volatile memory, card removal and reinsertion would normally force signature re-verification at the next script access. If an attacker could bypass the card removal shutdown process, they may be able to inject a rogue card with an unsigned script.

Clearly, script protection relies on subtle interactions between otherwise independent components, in this case, proper script verification depends on the card removal shutdown procedure always completing properly. This is a dangerous development practice that increases future regression fault likelihood.

We find that the noted technical flaw is significantly improved, in that while vulnerability still exists, we did not find an exploit.

2.4 Unchecked String Operation: Allows Overwrite of Stack Memory

As the SAIT Diebold Report indicated, while the vulnerable string operations for these two flaws still existed in the 1.96.8 code, the repairs made to correct other flaws mitigated the vulnerability these flaws introduced in versions prior to 1.96.8.

In the resubmitted version (1.96.9), one of these buffer overflow flaws is fixed, while the other is not. We are not aware of any exploit for the other, unfixed buffer overflow vulnerability.

3 Conclusions

As requested by the Florida Department of State, we analyzed the vendor's changes made in response to the FLSoS certification letter. We determined that the Signature Flaw repairs were accomplished and that the Preload Votes, Unchecked String Operations, and AccuBasic Script Misuse flaws were significantly improved.

We further note that the changes reflect a weak security approach. While these procedures appear to have fixed one of the required flaws and reduced vulnerability in the others, it is precisely these types of procedures that led to the original flaws. Presence of these procedures suggests that additional flaws will emerge in the application as it is now written and are even more likely to occur as regression faults in future versions.

We conclude by re-stating that this report does not constitute a comprehensive security analysis. We limited our investigation to four specific flaws. In spite of repairs made, significant security vulnerability continues to exist in the code base.

This system's overall security properties must be considered in terms of the complete elections environment to determine the practical impact of all remaining flaws.

4 References

1 "Software Review and Security Analysis for Diebold Voting Machine Software.", Joint Florida Department of State and Florida State University Statement of Work, May 14, 2007

2 Ryan Gardner, Alec Yasinsac, Matt Bishop, Tadayoshi, Kohno, Zachary Hartley, John Kerski, David Gainey, Ryan Walega, Evan Hollander, and Michael Gerke, "Software Review and Security Analysis of the Diebold Voting Machine Software", Final ReportFor the Florida Department of State, July 27, 2007, http://election.dos.state.fl.us/pdf/SAITreport.pdf

3 Kurt Browning, Florida Secretary of State Letter to Mr. David Byrd, Diebold Election Systems, dated July 31, 2007, http://election.dos.state.fl.us/pdf/SAITbrowningLetter.pdf

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

5 R.L Rivest, A. Shamir, L. M. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems", CACM, Vol. 21, No. 2, Feb 1978, pp. 120-126

6 The Black Box Report, SECURITY ALERT: Critical Security Issues with Diebold Optical Scan Design, July 4, 2005.