

Security Analysis of the eVACS Open-Source Voting System

Ananya Das
das@cs.ucdavis.edu

Yuan Niu
niu@cs.ucdavis.edu

Till Stegers
stegers@cs.ucdavis.edu

December 16, 2005

Abstract

The electronic Voting and Counting System (eVACS) is an open-source software used in an electronic voting trial the Australian Capital Territory, and has been recommended for use in future elections. In this paper, we report results from a review of the eVACS code and design, supported by static analysis tools. While no “hot exploits” have been found, several bad practices were identified.

1 Introduction

Electronic voting is touted as a way to reduce errors in counting and storage during the election. It is also a way to mitigate the cost of printing ballots in multiple forms to accommodate for different languages. E-voting has also been touted as being far more accessible to disabled voters. Several countries have already adopted electronic voting, including Australia, Brazil, Estonia, and the United States.

Australia developed and implemented eVACS while the public controversy about insecure voting machines in the United States ensued as a result of flaws encountered by voters using electronic systems, and academic papers like [13] that pointed out serious vulnerabilities. The integrity of an election depends on the proper functionality of all its components. To that end, the voting software needs to be thoroughly tested and evaluated before it is used. This paper presents an audit of the 2004 eVACS source code (available for download from the ACT Electoral Commission website) and the election procedures.

The electronic Voting And Counting System (eVACS) is software developed by Software Improvements Pty Ltd for a trial of electronic voting in the Australian Capital Territory (ACT). eVACS was developed as an open source project so that the software is transparent and readily available to scrutineers, candidates, and any other interested parties. eVACS has been used in the Australian Capital Territory for three elections to date (ACT 2001 Legislative

Assembly Election, 2003 Casual Vacancy, ACT 2004 Legislative Assembly Election) and is recommended for use in future elections. In 2004 alone, 68% of all votes cast during pre-polling were electronic votes [5].

eVACS consists of two major components: a system for casting votes via and a system for counting votes by computer. The electronic voting component is used in polling places to directly record votes and store them in a central database. The electronic counting component handles the data entry and counting of all votes (paper and electronic).

In the remainder of this paper we describe our review of the eVACS 2004 code. We first provide an introduction to previous work done regarding e-voting in Section 2. Next, we outline the procedures for an ACT election in Section 3.1. Next, we give a technical overview of the system in Section 3.2. We state our goals and methodology before the code review in Section 4.1. We present our findings and suggestions for mitigation of software related flaws in Section 4.3 and procedural flaws in Section 4.4. Finally, we summarize our findings and state our conclusions in Section 5.

2 Related Work

2.1 E-voting Systems

Voting Software SAVIOC (Safe Accurate Voting on Inexpensive Old Computers) proposes a minimalist approach that can turn any IBM into a voting station [11]. The entire program is stored on a floppy disk. EVM 2003 (Electronic Voting Machine) is another open source project which was developed by the Open Voting Consortium [15].

Protocols & Paper Trails Voteegrity, David Chaum's scheme, is difficult to explain but simple to use. Essentially, the voter receives one half of a receipt, while the other half is posted on a public bulletin. Without both halves, the receipts are meaningless. VoteHere uses Andrew Neff's protocol for verifiable mixing.

Internet voting An Internet voting system has been deployed successfully in Estonia, and is currently under use. The United States attempted to employ SERVE (Secure Electronic Registration and Voting Experiment), an Internet voting project, to allow overseas soldiers to vote, but canceled the project after negative evaluations from a team of security experts.

2.2 Audits & Evaluations

Evaluations of US Systems Computer scientists have long been among the most vocal critics of electronic voting systems, but it was not until the controversial Hopkins report, "Analysis of an Electronic Voting System" that security of e-voting systems became a "hot topic". This audit of the Diebold

AccuVote-TS DRE system was possible because Diebold had failed to secure its CVS repository [13]. With the advent of voting systems designed with security in mind, Karlof, Sastry, Wagner of UC Berkeley published an analysis of two promising cryptographic protocols to show that cryptographic security does not guarantee total security since human errors may still compromise the system [12].

Evaluations of eVACS The voting component of eVACS uses the Hare-Clark vote counting system to produce the results of the election. The Logic & Computation group at the National ICT of Australia reviewed the implementation of the Hare-Clark system using formal methods. They noted that the source code is difficult to understand without documentation and the control flow is complex. More importantly, their tests revealed that unless there were as many seats as candidates, the eVACS counting code reports one less candidate than the number of seats to be filled [4]. In 2004, BMM Test Labs conducted an audit of the eVACS 2004 source code. They confirmed that the Hare-Clark algorithm was correctly implemented. Their audit found no major problems in the general modules. [9]

3 Design of eVACS

3.1 Procedural overview

Before the Election All Australian citizens are required by law to vote. Those who cannot vote on the day of the election may vote during the three weeks of pre-polling. Polling materials, such as bar code cards, ballot materials, and certified lists of voters are the responsibility of the OIC (officer in charge) at any polling place. Manuals explicitly state that these materials cannot be stored at the polling place location. [7, 8] All materials must be distributed at the start of any polling day. The computers used for voting are booted with a CD-ROM containing configuration files and the eVACS software.

During the Election Voters receive a bar code card after being checked off the certified list of voters. They proceed to a cardboard booth with a monitor, bar code reader, and a keypad. The keypad has only a few active keys: the directional keys, Select, Undo, Finish, and Start Again. [6] After swiping the bar code card, they should be able to start voting. Voting is done by selecting candidates in the order of preference. When this is complete, voters verify the choices they make and swipe the bar code a second time to confirm the vote. If voters choose not to vote, they may register a blank, or informal vote. Voters should wait for the confirmation screen before leaving the voting station. Before exiting the polling place voters should discard their bar code cards in a ballot box.

After the Election At the end of a polling day, all voting machines are turned off. The OIC is required to record the number of paper ballots and bar codes used. Two copies of the voting database are burned to a CD-ROM from the server. A program produces a hash of all data stored to each CD to verify that results were not altered during the burning process or whether the CD was switched at any time. The server is then turned off and locked away (during pre-polling) or removed (on election night). Data from the CDs are loaded into the eVACS counting server in a central tally room and votes are tallied. When this is complete, the results are published. Daily results from data entry of paper ballots are also tallied and published until all votes have been counted. Scrutineers may observe the entire process.

3.2 Technical Overview

Once the bar code card is swiped, the client initializes the audio component and displays the welcome screen. It then sets up the bar code reader, reads the bar code from the card, and sends the encoded form of the bar code string to the server. The server calls a cgi script (`authenticate`) to check the validity of the string. The script accesses the database to compare the string with a list of authentic bar codes. If the the string is validated as an authentic bar code, the server notifies the client to prompt the voter for her votes. The voter makes her selections and presses the 'finish' button. The client now stores the votes and prompts the voter to swipe her card again. By re-swiping her card, the voter has committed her vote. The client verifies that this bar code matches the bar code that was swiped at the start of the vote. When the bar code is verified, the client sends an encoded form of the bar code, and the votes cast to the server. The server calls another cgi script (`commit_vote`) which accesses the central database. The script saves the votes in the database, thereby committing the votes. The script also invalidates the bar code by setting the 'used' field in the bar code to true. Finally, the server calls another cgi script (`commit_vote`) which saves the votes and updates the bar code in a backup database.

Once all the paper ballots have been submitted and the electronic votes have been cast, the counting component is manually executed by an election worker. The counting component counts all votes (paper and electronic) and outputs the results of the election.

Database eVACS uses two databases: a central database and a backup database. Before election day, each database contains a list of authentic bar codes. This list is used for authentication of bar code cards. Also when a bar code has been used, the 'used' field in the bar code table is set to true. Before election day, the database also contains a list of candidate names. When a vote is cast for a particular candidate, a count is incremented.

4 Audit of eVACS

We conducted a review of the eVACS version [2] available on the website of the ACT Electoral Commission. Although review and audit reports [5, 9], and the procedural manuals [7, 8] were provided, no technical documentation was available. The code frequently refers to a “DDS” document which is not publicly available.¹.

4.1 Audit Goals and Methodology

We audited the code base (excluding shell scripts and tests) contained in the archive `evacs.zip` [2] obtained from the ACT website. A complete audit of all 33000 lines of code was impossible due to time constraints. Problems we were looking for included the following:

- buffer overflow vulnerabilities
- flaws in authentication protocols
- bad coding practices
- insecure use of CGI scripts
- insecure use of systems calls
- privacy leaks

Similar to the audit performed by BMM International, our audit did not review whether eVACS actually provides its expected functionality. This has been addressed to some extent in the reviews by the ACT Electoral Commission [5] and ICT Australia [4]. In particular, the Electoral Commission reports to have conducted extensive regression testing. We also did not scrutinize the test cases provided. In order to make a fully qualified judgment on whether a specific setup of eVACS is suitable for use in a general election, the following components need to be scrutinized as well:

1. Physical setup of all computers used and the network used
2. Configuration of database, operating system, and eVACS (this includes the automatic setup scripts mentioned in[5])
3. Source code of all software used (operating system, compiler and CD recording application)

To address items 2 and 3 above, it would be helpful if the Electoral Commission would publish a disk image of the eVACS CD used in elections. We understand

¹The ACT Electoral Commission is not authorized by Software Improvements Pty Ltd, who seem to be the copyright owners, to distribute copies of the document. Software Improvements did not respond to a request to supply us with the document.

that a complete review of the operating system code might require more resources than available to secure the election. However, it would definitely be feasible to at least scan the employed software for known vulnerabilities and check whether there are any discrepancies between the sources used and the authoritative code base.

To audit the code, we used manual inspection, supported by two static analysis tools: David Wheeler’s Flawfinder [17] and the Fortify Source Code Analysis Suite [10].

4.2 Threat model

When auditing the code, we anticipated the following attack vectors:

- benign but erroneous application code
- malicious voters
- malicious poll workers (with the exception of the OIC)
- intruders on the local network
- compromised voting terminals.

Judging from the code base, it seems that eVACS development presumed the local network to be trusted, as well as the operating system and CD recording software used.

4.3 Technical Findings

4.3.1 Good practices

The eVACS developers can be complimented in that we did not find any of the following common security flaws:

1. SQL injections
2. user-exploitable buffer overflows
3. vulnerabilities in CGI scripts

The use of the widely deployed OpenSSL cryptographic library is also a good practice.

4.3.2 Bad Practices

Most of the problems we found with the system were due to bad software engineering practices. Although, to our knowledge, none of these are exploitable, they still present a danger to future releases of the software. This is particularly troublesome if new developers not familiar with implicit assumptions made by the original eVACS developers attempt to modify the code.

Parameter checking If performance requirements allow (and they would for the present system), it is good practice to declare whether a function requires the caller to check the arguments or whether it is the callee’s responsibility, and have the callee perform argument checking in either case. In eVACS, most functions perform only casual error checking. For instance, problems arising from corrupted data structures such as an unintentionally circular list are not checked for. For a highly reliable system, extensive error checking should be performed, and should be reflected in the tests.

Return values It is also good practice to check informative return values, in particular of critical functions. In our audit, we counted at least 46 calls² after which the return value was not checked, 40 of which were calls to `malloc`. These constitute 46 potential denial of service vulnerabilities. In addition, this also suggests that, from a software engineering perspective, only limited testing had been done.

Memory leaks In 14 codepaths, we found that dynamically allocated memory was not freed. The amount of wasted memory is small, but could potentially add up. For instance, some memory allocated by the server when responding to a client’s request to send a ballot is not released, and this function is called (at least) once for every voter³.

Code duplication It is apparent that the eVACS developers frequently copied code fragments between different source files. Using the plagiarism detection tool MOSS [3], we were able to quickly identify large identical sections in different files. This practice should be avoided for several reasons. One, bugs in copied code may be overlooked and not fixed in every occurrence. Two, pasted code usually requires many small changes (such as renaming a variable throughout), an error prone process.

Use of preprocessor macros The source code is full of hard-coded numbers and strings, such as the number of seats in a race (5 or 7), all SQL commands, CGI variable names, etc. These should be `#defined`.

Another example where the use of informative constants would improve maintainability and readability is provided by the technique of disabling source fragments using the directive

```
#if 0
// ...statements ...
#else
// ...statements ...
#endif
```

²If code was duplicated in several files, we counted it twice

³This count disregards voters who enter the booth without attempting to vote.

We found 35 instances of “comments” using this technique in the code base. A reader different from the author of the lines in question is left without a clue as to why the code was disabled. We suggest to define a meaningful and documented macro instead of using 0 directly, such as (at least)

```
/* 0 = production mode, 1 = debugging mode */  
#define DEBUG 0
```

Communication Without access to the database configuration files, we were unable to determine whether eVACS uses any authentication and/or encryption when communicating with the database. For instance, PostgreSQL provides built-in encryption and authentication using the Secure Sockets Layer (SSL). Using encryption is strongly suggested to thwart vote sniffing on the local network; an integrity check is also desirable since in the current authentication protocol, nothing binds the bar code to the vote cast.

Password Handling To tally up the number of voters ranking each candidate highest, the program `display_first_preferences` is installed on the tallying server. Execution of this program is restricted to authorized users, presumably because it is desired to keep the intermediate results secret until polling night. For a user to be authorized, the password she enters is `crypt`d using the constant salt "ev" and then compared to the encoding of the password stored in the database. Above the call to `crypt`, the following comment is placed:

```
Encrypt the password, the extra security from implementing  
a random salt is not needed here.
```

This is bad security design: Why make a system deliberately less secure, in particular when security comes at almost no cost? Without a random salt, the password is vulnerable to dictionary attacks. The developers that placed the above comment in the code would probably reply that this requires attackers to have physical and logical access to the server, and so break the system anyway. But possibly the attacker’s account on the server is different from the one which is used to run the voting server software. Depending on whether scrutineers should be allowed to see the password, it might be advisable not to echo it to the console [14] as it is done now.

The current implementation of `display_first_preferences` overwrites the array holding the password after it has been passed to `crypt` and is no longer needed. However, this is done using a simple `for`-loop in the same function, which might be flagged as “dead store removal” by the compiler and thus be omitted from the executable. We were not able to confirm this when using the GNU Compiler Collection 4.0 with the highest optimization flag, `-O3`. Nevertheless, it might become an issue when other compilers are used. More about this type of attack and countermeasures can be found in [16, section 11.4].

The password is also not `mlocked` to prevent it from being swapped to disk, probably because this usually requires superuser privileges. It seems that this could be remedied by use of the Linux Security Modules [1] framework.

Vote Reconstruction A peculiar security feature of eVACS is that the voter’s keystrokes are logged and sent along with her vote to the server. The server then determines the vote from the keystroke sequence what the vote must be and rejects it if there is a mismatch. This feature only guards against accidental changes of votes, but not against attackers such as compromised voting terminals or network-based attacks.

Documentation The quality of documentation varies, but is mostly replaced by a reference to the “DDS” document. Without having this document at hand, both auditing and continuing the development of eVACS becomes unnecessarily difficult.

4.4 Procedural findings

Selling an unused bar code card The eVACS system is not designed to resist vote buying/selling. When a voter has completed the voting process, she is required to drop her bar code card in a ballot box. However, there is no check by an election worker that the returned card is an authentic card. Malicious Voter *A* can create a replica of one of these cards. This fake card does not need to be scannable, it only needs to look like an authentic bar code card. Upon entering the polling place, voter *A* will receive an actual bar code card. She will enter the voting booth and pretend to vote. When leaving, she will drop the fake card into the ballot box. She now has hold of an unused authentic bar code card which she can sell to malicious voter *B*. When voter *B* enters the polling place, he will receive a new bar code card. In the privacy of his voting booth, he can use the two cards to vote twice.

Although at each polling place, used cards are counted at the end of the election, and a check is done to verify that the number of cards returned is equal to the number of votes cast at the polling place [8], this check would not prevent malicious voters from buying or selling votes. Even if it was found that the number of used cards exceeded the number of votes cast, there is no way to determine which of these votes were cast by a voter who maliciously obtained multiple cards. The only solution to this problem would be to cancel all the votes acquired from this polling place and require all the voters to vote again.

One way to prevent this would be to implement a card reader which verifies that the returned card has actually been used before it is dropped in the ballot box. Performing this check has another benefit: it allows election workers to verify that each voter has actually voted. Voters should be reassured that their votes are not recorded on the cards and the election worker who swipes their cards will have no knowledge of their votes.

Forging a bar code Upon entering the polling place, each voter who chooses to vote electronically receives a voting card with a bar code printed on it. Each bar code consists of the SHA-1 hash of the concatenation of three strings: a random string, a code for the electorate, and a code for the polling place. A

4-bit checksum is appended to this hashed string. Finally, the entire string is encoded with a simple mapping to ASCII characters.

The security of the eVACS system heavily relies on the difficulty of forging of bar codes. If bar codes can be forged with little effort, malicious voters can create fake bar codes cards to vote multiple times. As long as the bar code on the fake card matches a bar code listed as unused in the central database, the voter can use the card to cast a vote. Since a bar code card does not record the vote cast on it, even if a fake card was used to cast a vote, then later discovered to be fake, it would be impossible to determine which votes were cast by this card.

Therefore the use of a cryptographically secure random number generator is crucial when generating bar codes. Unfortunately, the bar code generator is not included in the public eVACS release we analyzed.

Unreliability of bar code readers In the pre-poll voting manual issued by the ACT, the section for *Instructions on Voting by Computer* provides the following note:

Bar codes sometimes need to be swiped more than once to get a correct read – the bar code reader will beep when the bar code is read correctly. Please let electors know this when they are issued with their bar code. You may need to allocate staff to help voters with their bar codes and electronic voting [8].

While the voter is prompted to swipe her card to confirm her vote, a confirmation page, showing all of the voter's selections is displayed on the computer screen. If the bar code reader fails to beep at this time, and an election worker must be summoned to assist the voter, the voter's privacy is clearly jeopardized.

One solution to this problem would be to provide voters a switch that allows them to turn off and on their computer screens. Each voter should be made aware that she must switch off her screen if an election worker needs to enter her voting booth. However if the error message is displayed on the screen, then the election worker must be able to see the screen in order to help.

Accessing the Local Network The eVACS system makes a fundamental assumption that the local network at each polling place is secure. Their threat model, therefore, does not incorporate attacks that could compromise the local network. For example, votes are not encrypted by any means. An adversary who has access to the local network may be able to sniff network packets and learn the password needed to access and alter the central database. The adversary can then change the number of votes cast for candidates. The adversary can also retrieve a list of authentic and unused bar codes, and use this list to generate bar codes to use in fake cards. One way to prevent this would be to employ an encryption protocol such as SSL. The routers used should be secured properly so that no intruders can gain access to the network and no adversary should be able to set up or switch it with a wireless router.

5 Conclusion

5.1 Summary

We have presented an overview of the eVACS system and the results of our audit with regard to flaws and possible countermeasures. While we found no “hot exploits,” we did find some vulnerabilities and bad software engineering practices. Our findings show a fragile system with a large trusted computing base. The lack of documentation makes extensibility and maintenance difficult. While we commend the developers for publishing the source code, the lack of documentation makes it difficult to understand the developers’ intentions.

Within the code, there is a much repetition of entire code blocks. Even more dangerous are unchecked return values and memory leakage resulting from a failure to free allocated memory. We noticed a habit of hard coding number and string values. Curiously, we found little or no use of cryptography to protect or verify the transfer of information.

eVACS is not designed to prevent vote buying and selling. Voters can sell their votes by returning fake cards and selling unused authentic cards. Furthermore, faulty machinery in the voting booths may require election workers to enter booths, thereby risking voter privacy.

The eVACS system assumes that the local network is secure. Therefore, they use no encryption while transferring data through the network. If an adversary could access the network, she could alter records stored in the main database and the backup database. Even if an adversary could make changes to only one database, there would be no way to determine which database held the correct information.

Although the use of scrutineers throughout the process supports the enforcement of procedures, scrutineers may not be able to detect all malicious behavior.

5.2 Future Work

There are still a variety of ways to analyze the vulnerabilities of the eVACS system. In our analysis of the system, we were unable acquire the configuration files to run the source code. Running the source code would provide an in-depth understanding of the control flow of the system and insight into its main components (client, server, and counting). A study of the documentation for the source code would also be informative means to understanding the system. Perhaps the most comprehensive approach to understanding the security vulnerabilities of eVACS is to attempt to attack a set-up of the system. This would require the voting cards and the hardware (bar code reader, computer screen, and keypad) employed by eVACS.

5.3 Acknowledgments

We would like to thank Matt Bishop, Hao Chen, and Scott Ritchie for useful discussions about this work. We would also like to thank the ACT's Deputy Electoral Commissioner Alison Purvis for supplying us with information about eVACS and the voting process.

References

- [1] "Linux Security Modules Project," Website.
<http://lsm.immunix.org/>
- [2] "electronic Voting and Counting System," (2004), MD5 hash a184981edcfb5c1e379c082129ff9723.
<http://www.elections.act.gov.au/evacs2004.zip>
- [3] "MOSS – A System for Detecting Software Plagiarism," (2005).
<http://www.cs.berkeley.edu/~aiken/moss.html>
- [4] Abate, P., J. Dawson, R. Goré, M. Gray, M. Norrish and A. Slater, *Formal methods applied to electronic voting systems*, 2003, website.
<http://users.rsise.anu.edu.au/~rpg/EVoting/>
- [5] Australian Capital Electoral Commission, "Electronic Voting and Counting System Review," (2005),
<http://www.elections.act.gov.au/adobe/2004ElectionReviewComputerVoting.pdf>.
- [6] Australian Capital Territory Electoral Commission, *The electronic voting process*, 2004, website.
<http://www.elections.act.gov.au/EvoteTR.html>
- [7] Australian Capital Territory Electoral Commission, "Polling place management procedures manual," (2004), manual for the 2004 Legislative Assembly election.
- [8] Australian Capital Territory Electoral Commission, "Pre-poll voting procedures manual," (2004), manual for the 2004 Legislative Assembly election.
- [9] BMM International Pty Ltd, "Audit of eVACS Software Code," (2004), issue 1.1, available through the ACT Electoral Commission.
- [10] Fortify Software Inc., "Fortify Source Code Analysis Suite," version 3.1.0086.
- [11] Gaston, C., *Savioc voting systems*, 2005, website.
<http://www.savioc.com>
- [12] Karlof, W., Sastry, *Cryptographic voting protocols: A systems perspective* (2005).

- [13] Kohno, R. W., Stubblefield, *Analysis of an electronic voting system* (2004).
- [14] Messier, M. and J. Viega, "Secure Programming Cookbook for C and C++," O'Reilly, 2003.
- [15] Open Voting Consortium, *The electronic voting machine project*, 2004, website.
<http://evm2003.sourceforge.net/>
- [16] Wheeler, D., "Secure Programming for Linux and Unix HOWTO," (2003), version 3.010.
<http://www.dwheeler.com/secure-programs/>
- [17] Wheeler, D., "Flawfinder," (2004), version 1.2.6.
<http://www.dwheeler.com/flawfinder/>