Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

The return of software vulnerabilities in the Brazilian voting machine

Diego F. Aranha[a,b,c,*], Pedro Y.S. Barbosa[d], Thiago N.C. Cardoso[e], Caio L. Araújo[f], Paulo Matias[g]

[a] Department of Engineering, Aarhus University, Finlandsgade 22, Bygning 5125, 8200 Aarhus N, Denmark
[b] Institute of Computing, University of Campinas, Brazil
[c] DIGIT Centre for Digitalisation, Big Data and Data Analytics at Aarhus University, Denmark
[d] Department of Systems and Computing, Federal University of Campina Grande, Brazil
[e] Hekima, Brazil
[f] Center of Informatics, Federal University of Pernambuco, Brazil
[g] Department of Computing, Federal University of São Carlos, Brazil

ARTICLE INFO

Article history:
Received 6 March 2019
Revised 12 June 2019
Accepted 18 June 2019
Available online 20 June 2019

Keywords:
Electronic voting
Direct recording electronic
Software vulnerabilities
Security
Integrity

ABSTRACT

This paper presents a detailed and up-to-date security analysis of the voting software used in Brazilian election based on results obtained by the authors in a recent hacking challenge organized by the national electoral authority. During the event, multiple serious vulnerabilities were detected in the voting software, which when combined compromised the main security properties of the equipment, namely ballot secrecy and software integrity. The insecure storage of cryptographic keys, hard-coded directly in source code and shared among all machines, allowed full content inspection of the software installation memory cards, after which two shared libraries missing authentication signatures were detected. Injecting code in the libraries allowed the execution of arbitrary code in the machine, violating the integrity of the voting software. Our progress is richly described, to illustrate difficulties and limitations in the testing methodology chosen by the electoral authority, and to inform how teams participating in future challenges can optimize their performance. We trace the history of the vulnerabilities to a previous security analysis, providing some perspective about how the system evolved in the past 6 years. As far as we know, this was the most in-depth compromise of an official large-scale voting system ever performed under such severely restricted conditions.

1. Introduction

Brazil is one of the largest democracies in the world, with 147 million voters in the 2018 general elections[1]. Elections are conducted every 2 years in the month of October, alternating between municipal elections (members of the city council and mayors) and general elections (members of the lower house, senators, governors and the president).

* Corresponding author at: Department of Engineering, Aarhus University, Finlandsgade 22, Bygning 5125, 8200 Aarhus N, Denmark.
E-mail addresses: dfaranha@eng.au.dk (D.F. Aranha), pedroyossis@copin.ufcg.edu.br (P.Y.S. Barbosa), thiagoncc@gmail.com (T.N.C. Cardoso), caioluders@gmail.com (C.L. Araújo), matias@ufscar.br (P. Matias).

[1] Based on statistics from (in Portuguese): http://www.tse.jus.br/eleicoes/estatisticas/estatisticas-eleitorais.

https://doi.org/10.1016/j.cose.2019.06.009
0167-4048/© 2019 Elsevier Ltd. All rights reserved.

Direct Recoding Electronic (DRE) voting machines were first introduced in the country in 1996, after frequent reports of fraud during transport and tabulation of paper ballots, and elections became fully electronic in the year 2000. Voter-verified paper records were briefly experimented with in 2002, but deemed too costly and cumbersome by the electoral authority. Since then, multiple bills were introduced in Brazilian Congress to reintroduce paper records, the last one demanding deployment in 2018, but ultimately suspended on the grounds of unconstitutionality by the Supreme Court, with the argument that paper records violate ballot secrecy (Superior Electoral Court, 2017b).

As a response to the frequent calls for increased transparency, the Superior Electoral Court (SEC) started organizing hacking challenges in 2009. These are restricted events, where pre-approved external and independent researchers can examine the security mechanisms implemented within the system, find vulnerabilities and provide suggestions for improvement. After the first hacking challenges in 2009 and 2012, the event became mandatory and now happens from six months to one year before the elections (Superior Electoral Court, 2015).

Under a turbulent and polarized political environment, the debate around the security and transparency of Brazilian machines is growing in popularity. Access to voting equipment by researchers is still notably restricted, with the hacking challenges thus presenting a unique opportunity to perform independent security analysis. The first such analysis was performed in 2012, and a full report was published afterwards by Aranha et al. (2014). In the occasion, the authors were able to mount an attack against ballot secrecy based on insecure random number generation, and also document many other security issues. As the main concerns, the authors detected insecure storage and massive sharing of cryptographic keys, insufficient verification of software integrity, and a faulty software development process conducted under an inadequate adversarial model.

Our contributions. We continue and update the efforts towards a security analysis of the Brazilian voting machine, by presenting our findings collected in the 2017 edition of the hacking challenge. In summary, we were able to:

(i) Recover the symmetric cryptographic key protecting the memory cards that install software before the elections, allowing decryption and full inspection of their contents. Because this symmetric key is shared among all machines, recovering it in a real election would have nationwide impact.
(ii) Find two software libraries missing digital signatures for authentication or any other kind of integrity check, allowing direct injection of arbitrary code in their functions. Consequently, it was possible to indirectly modify the programs linked against them, such as the official voting application.
(iii) Exploit the code injection capabilities to break ballot secrecy for selected votes, by manipulating cryptographic keys generated on-the-fly; to receive commands from a regular keyboard; and to manipulate logging records generated by the voting software.
(iv) More importantly, inject code to manipulate the strings presented by the software to the voter in real time and advertise a hypothetical candidate or political party.

The code injection capabilities were later extended to manipulate the outcome of a simulated election, since the vulnerabilities detected until that point were already sufficient. Substantial progress was made toward this goal by erasing all votes in an electronic ballot, which triggered an empty ballot consistency error in the voting software. The attack was then adapted to manipulate votes stored in memory, but the event was interrupted before we could validate this version of the payload in the real hardware.

Paper outline. The remaining sections are organized as follows. In Section 2, we describe the standard Brazilian election procedure, the software ecosystem of the Brazilian voting machine, and some of its security mechanisms. In Section 3, we summarize the history and discuss the format, limitations and results in previous hacking challenges. Section 4 continues by describing our planning steps in the latest edition, while Section 5 reports on the actual progress. Section 6 reviews related work and analyzes our findings in context, tracing a historical perspective with relation to other security analysis of the same system together with potential countermeasures. Section 7 concludes.

2. Background

A surprising characteristic of Brazilian elections is that the full election administration is under control of a single institution. The Superior Electoral Court (SEC) is responsible for performing voter registration, designing election procedures, recruiting election officials, organizing the logistics on election day, deciding and implementing what election technology will be used, and any other remaining operational tasks. As part of the judicial branch of government, the SEC also resolves all legal disputes regarding elections. It is presided by a Supreme Court judge, who accumulates a seat to report on electoral issues involving constitutional matters.

Brazil witnessed rampant fraud on paper ballot elections during the transition to a democracy in the 80s, motivating the SEC to consider the deployment of electronic devices for collecting votes. In 1982, the move to digital systems started by employing electronic transmission of election results. A few years later, voter registration data was migrated to digital storage and, in 1991, the first personal computers were used as voting equipment in small referendums.

The first DRE voting machines were introduced in 1996 at a small scale, in only 56 municipalities. The machines were manufactured by Unisys and equipped with an Intel 80386 processor. In terms of software, the machines employed a DOS-compatible operating system called VirtuOS, manufactured by the Brazilian company Microbase. Later models were subsequently introduced almost every other year and manufacturing was transferred at some point to Procomp, the Brazilian subsidiary of Diebold Incorporated.

The later models maintained the same initial design and interface, but adopted more recent hardware components and software. For example, machines from 2002 to 2006 were

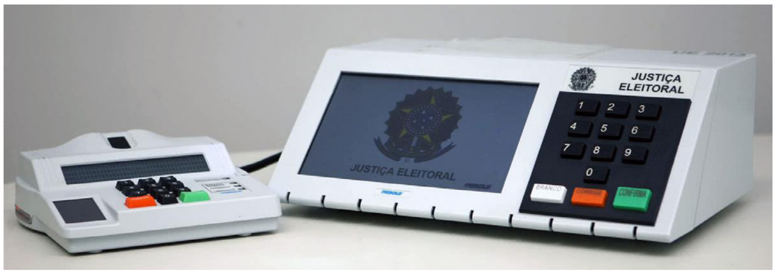


Fig. 1 – The two terminals of the Brazilian paperless DRE voting machine. In the left, the election official terminal has a fingerprint recognition device visible in the top; while the voter terminal is on the right. The cable in the back connecting the two terminals allows software executed in the election official terminal to authenticate voter data (registration number or fingerprint) stored in the voter terminal. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

deployed with the Microsoft Windows Compact Edition operating system, and newer models exclusively run the GNU/Linux operating system. The latest model, introduced in 2015, includes a Hardware Security Module (HSM) of sorts called MSD (*Master Security Device*) for computing critical cryptographic operations, storing cryptographic keys and verifying software integrity during the boot process. Software-wise, initial versions of the voting software were also produced by Diebold-Procomp, but the SEC took ownership of the software development process in 2006. The team responsible for software development consists in a mix of in-house and contracted developers. In 2008, the SEC started enrolling voters for biometric identification using fingerprints and the effort recently reached half of the voters.

2.1. *Voting equipment*

The Brazilian voting machine, or "*urna eletrônica*" in Portuguese, consists of a classical DRE device without any type of voter-verified paper record. The machine is composed of an election official terminal, used to authenticate voters by their registration numbers or fingerprints, and a voter terminal where votes are cast via a keyboard reminiscent of a modern telephone. The full machine is shown in Fig. 1. Candidates and their political parties are selected by typing the corresponding numbers in the voter terminal. It is also possible to either cast a blank vote, change or confirm a vote by pressing the colored buttons (from left to right, respectively).

The two terminals are connected by a long cable, a questionable design aspect in terms of ballot secrecy. The cable provides access to the voter data stored by the voter terminal. This means that the voting machine simultaneously observes voter identification information and the votes being cast. Besides the keyboards for input, communication to and from the voting machine is possible via memory cards and a small printer. Memory components include internal and external flash cards responsible for storing the operating system, the components of the voting machine software, and data related to candidates and voters. During an election, the files storing the current state are redundantly stored in both the internal and external memories, so they can be recovered in case either is permanently damaged.

External flash cards are inserted in a slot in the back of the machine. A software installation card, called an *install card* (or "*flash de carga*"), is used to transfer official voting machine software to the internal memory before the elections. Another flash card, called *voting card* (or "*flash de votação*"), is inserted during elections in the same external slot for providing voter and candidate metadata. There is another slot in the back of the machine to attach a USB stick called Memory of Results (or MR – "*Memória de Resultados*"). The MR stores election results and other data that is made publicly available later. These external interfaces are protected by tamper-evident seals signed by electoral judges in a public ceremony.

2.2. *Official voting procedures*

Elections using the Brazilian voting machine follow procedures typical of DRE-based elections and a very similar workflow. The preparation steps performed before elections can be found below:

(i) *Development of software components*: contrary to other countries, election software is continuously maintained and updated by the SEC. Inspectors from political parties and other institutions can examine the source code under a Non-Disclosure Agreement (NDA) at the SEC headquarters for a few months before the elections and provide suggestions for improvement.

(ii) *Distribution of software components*: a specific version of the software is frozen and compiled in a public ceremony, to be later transmitted electronically to the local branches of the SEC a few days before the election. Hence each new election runs on a more recent version of the codebase. Upon receipt of the official voting machine software, staff in the local branches generate the install cards using desktop computers. These memory cards are then transported across the states to multiple places where voting machines are stored in-between elections. Each card installs up to 50 voting machines.

(iii) *Installation of voting machine software*: voting software is installed in the machines through the install cards using a software module called SCUE.[2] The machine boots from the install card and the system self-checks the integrity of its own files before they are copied to the internal memory. Afterwards, the machine can be initialized from internal memory, and a hardware test is performed on the first boot.

(iv) *Distribution of the voting machines*: uniquely identified voting machines are assigned to the corresponding polling places. Each assignment is digitally signed using a pairing-based Boneh–Boyen short signature scheme (Boneh and Boyen, 2008), instantiated with a 160-bit Barreto–Naehrig curve (Barreto and Naehrig, 2005), and the resulting signature is used as witness, called *correspondence code* (or "*resumo da correspondência*"). A database containing the assignments is published afterwards.

There are around half a million voting machines in operation, all running the same software. In the election day, a uniform procedure is executed across all polling stations, simplified below for clarity:

1. Voting machine prints the *zero tape* (or "*zerésima*") between 7 A.M. and 8 A.M. This is an official public document attesting that no votes were supposedly computed for any candidates before the start of the elections.
2. The election official opens the voting session at 8 A.M. by typing a command in the election official terminal.
3. The voters provide identification information and are authorized to cast votes in the machines.
4. The election official closes the voting session at 5 P.M., local time, if no additional voters remain in the queue.
5. The voting machine prints the *poll tape* (or "*Boletim de Urna*"), containing per-machine totals for each candidate. Copies of this physical document are signed by election officials, distributed among inspectors from the political parties and should be fixed on an accessible location in the polling place.
6. The voting machine records electronic versions of the authenticated public products of the election. They consist of a digital version of the poll tape containing the partial results; a chronological record of events registered by the machine (LOG); and the Digital Record of the Vote (DRV), a shuffled list of the actual votes. These files are digitally signed and stored in the MR.
7. The election official violates the seal and retrieves the MR containing the public products.
8. The election official boots a networked desktop computer in the polling place using *JE Connect*, a dedicated LiveUSB with a GNU/Linux distribution. This system establishes a secure connection with the tabulation infrastructure using a Virtual Private Network (VPN).
9. The election official attaches the MR to this computer and transmits the public products of the election to the centralized tabulation system.
10. The central tabulator combines all the partial results and the official result of the election is declared.

The whole process of transmission and tabulation usually takes a few hours. After three days, digital versions of the poll tapes received by the tabulation system are made available in the SEC website for independent checking, and other election products can be obtained by the political parties through a formal process. Malicious manipulation of the tabulation phase can only be detected by manual comparison of printed and digital versions of the poll tapes (Aranha et al., 2016).

2.3. *The software ecosystem*

The whole software codebase has a complexity in the order of tens of million lines of code, including all components. The version made available in the latest hacking challenge was tagged with the string "*The Hour of the Star*" (or "*A Hora da Estrela*") in homage to a novel authored by Brazilian writer Clarice Lispector.

The source code for the voting machine software alone has 24 million lines and is organized as a customized GNU/Linux distribution (UENUX) for the Intel 32-bit architecture. Besides the typical userland libraries, it includes the official voting application (VOTA – "*Votação Oficial*"), the voting machine software installation system (SCUE – "*Sistema de Carga da Urna Eletrônica*"), a forensic tool to recover damaged data (RED – "*Recuperador de Dados*"), a system for manually typing partial results in case a voting machine malfunctions (SA – "*Sistema de Apuraação*"), and an application manager (GAP – "*Gerenciador de Aplicativos*"). Low-level software includes a customized bootloader (based on Syslinux[3]), kernel and drivers. The bootloader boots from a nonstandard offset and the BIOS is modified to take this into account. Among the customizations in the GNU/Linux kernel are the inclusion of additional device drivers and modification of some standard security mechanisms. In December 2017, the codebase was shifting from the 2.6 to the 3.18 branch of the kernel.

Encryption. There are several security mechanisms implemented in the voting machine software, where the main security goal is to enforce integrity checking and authentication of the hardware and software. This is attempted by combining multiple layers of encryption, to prevent inspection and extraction of sensitive information by outsiders, and several authentication primitives. As examples of low-level mechanisms, the bootloader contains an AES-256 encryption key to decrypt the kernel image under the ECB mode of operation during initialization of the system. The kernel has keys embedded to encrypt/decrypt individual files in a MINIX file system under the AES-XTS mode; and to implement a repository of cryptographic keys encrypted under AES-256 in CBC mode. The latter set of keys, here called *authentication keys*, include private keys to digitally sign the public products of an election, public keys and certificates for signature verification, and secret keys to authenticate poll tapes using SipHash (Aumasson and Bernstein, 2012). Fig. 2 presents how the encryption layers are organized, with modules in the top containing the keys to decrypt modules immediately below.

[2] From the Portuguese "*Sistema de Carga da Urna Eletrônica*", or Voting Machine Installation System.

[3] Syslinux bootloader: http://www.syslinux.org.

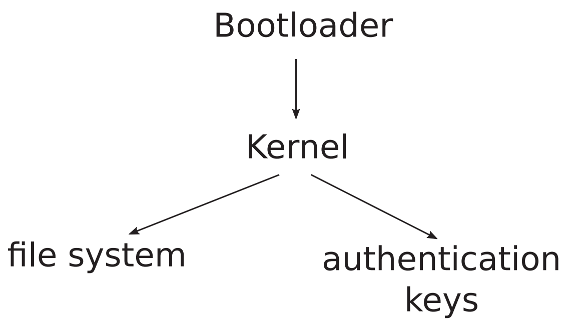


Fig. 2 – Chained layers of encryption in the install card. Arrows denote possession of encryption keys to decrypt the next layer.

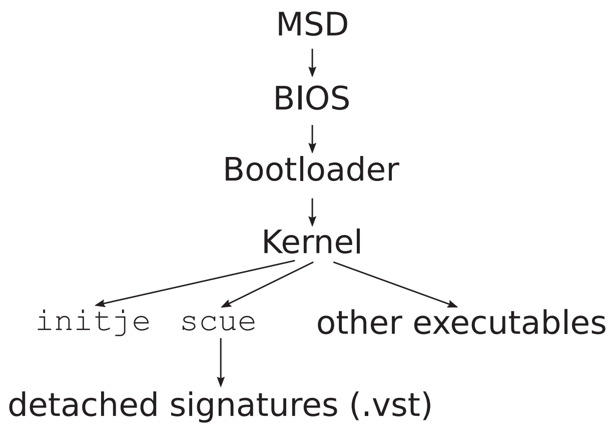


Fig. 3 – Voting machine chain of trust, starting with the MSD. Arrows denote signature verification. Detached signatures for installation files are verified by the SCUE module.

Authentication. In terms of authentication, the BIOS, bootloader and kernel images are digitally signed using ECDSA, instantiated with the NIST P-521 curve. Fig. 3 presents the chain of trust. The chain starts with the MSD and each component authenticates the next in sequence until userland applications are successfully loaded and executed.

Kernel modules, executable binaries and shared libraries are digitally signed with RSA-4096 and the signatures appended to the corresponding files. The corresponding public key is embedded in the kernel and used by the loader for verification to prevent tampering with these files. All files in the install and voting cards, and the public files resulting at the end of an election are digitally signed, with signatures computed by the MSD. These detached signatures are stored in VST files, this time using an Elgamal signature scheme instantiated with the NIST P-256 elliptic curve. This module was designed and implemented by technicians from the cryptography sector (CEPESC – "*Centro de Pesquisa e Desenvolvimento para a Segurança das Comunicações*") of the Brazilian Intelligence Agency (ABIN – "*Agência ncia Brasileira de Inteligência*"). The poll tapes have QR codes encoding partial results in a machine-readable format, to facilitate automated comparisons with published results (Aranha et al., 2016), and these are digitally signed using the Ed25519 digital signature scheme (Bernstein et al., 2011).

Random numbers. Many of the deployed cryptographic algorithms for encryption and authentication require random numbers, and there are multiple algorithms implemented across the codebase. The Elgamal signatures computed by the MSD rely on the `xorshift` family (Marsaglia, 2003) of pseudo-random number generators (PRNG). The mechanism for shuffling votes inside the DRV file is implemented by sampling random table positions from a combination of two other generators: reading directly from `/dev/urandom` or from a customized PRNG based on a 32-bit variant of the 64-bit version of an obscure algorithm called Sapparot-2 (Levin, 2005). Shuffling is performed within the DRV to disassociate the order in which votes are cast from the order they are stored, attempting to protect ballot secrecy.

Additional software components are *InfoArquivos* and *RecArquivos*, parts of the system for transmission of results and tabulation; and GEDAI (or "*Gerenciador de Dados, Aplicativos e Interface com a Urna Eletrônica*"), a software subsystem to manage and generate install/voting cards and empty MR sticks in the Windows platform. It is worthy noting that the computers running GEDAI execute a security software suite at the operating system level in an attempt to prevent tampering during the generation of install cards. This suite is called SIS (or "*Subsistema de Instalação e Segurança*") and is provided by the Brazilian company Modulo Security.

3. The hacking challenges

The Public Security Tests of the Brazilian voting system, or TPS – "*Testes Públicos de Seguranç a*" in Portuguese, allow independent experts to evaluate the security mechanisms implemented in the voting system. Experts attempt to violate the classical security goals of any voting system, namely ballot secrecy and integrity, and are asked to later suggest proper improvements to restore the security of the affected mechanisms. The format and scope of the event evolved with time, and the challenges recently became mandatory as an official event in the election calendar.

3.1. *History*

In the first TPS, organized in 2009, researchers did not have access to the source code of the software and the chosen model was a competition with money prizes. As a result, the winning strategy by Sergio Freitas da Silva was a black-box attack against ballot secrecy using a radio receiver to capture keyboard emanations (Superior Electoral Court, 2009). Afterwards, the electoral authority reportedly shielded the keyboard as mitigation.

In 2012, independent experts had the first opportunity to examine voting machine software source code without NDA restrictions. The format of the challenge was slightly tweaked and financial awards were discontinued. The reduced limitations were enough to bring out the first vulnerability reports concerning the software. In particular, the winning attack strategy was an accurate in-order recovery of the votes cast, successfully mounted in a realistically-sized simulated polling station with 950 votes. The attack was based only on public data published in the DRV file, and superficial knowledge about how the votes were stored in the file. The main attack vector was a vulnerability involving a call to `srand(time(0))` to seed the pseudorandom generator, with a subsequent printing of the timestamp in the zero tape. With knowledge of the timestamp and the order of voters in the voting queue, it would be possible to break ballot secrecy for an entire polling station. Alternately, by obtaining the time an important vote was cast (by a judge or some other public official), it would be possible to discover the position of the voter in the queue using the LOG file and break ballot secrecy for the selected voter. Other detected design flaws included massive sharing of cryptographic keys, insecure storage of secret key material and inadequate choice of algorithms. Aranha et al. (2014) provide a detailed first-hand account of the event, vulnerabilities and aftermath.

The challenges resumed only in 2016, with the introduction of an NDA that ended up alienating a large portion of the local technical community. The scope was also extended to include software components from the tabulation system and the GEDAI system for generating install cards. This edition saw the first successful attack against the integrity of results, presented again by Sergio Freitas da Silva (Superior Electoral Court, 2016). He demonstrated how checksums implemented in the poll tapes did not provide authentication, allowing anyone with knowledge of the underlying algorithm to compute correct checksums for fake results. The manipulated results could then be transmitted to the central tabulator using a subsystem used to transmit results whenever the voting machine malfunctions.

After substantial pressure from the technical community, the NDA was considerably relaxed in the subsequent year, to allow participants to publicly discuss their findings after coordinated disclosure of any vulnerabilities detected during the event. The source code for the MSD firmware was also included for inspection. Although it has been progressively deployed in the past ten years, the fingerprint identification system is still out of scope.

3.2. *The 2017 edition*

This work reports our findings collected during the 2017 edition of the TPS. It was comprised of five main phases: registration and pre-approval, code inspection, submission of testing plans, the actual trials, and the reporting of results. Rules were described in an official call for participation published in the SEC website (Superior Electoral Court, 2017a). Multiple committees were involved in the organization, such as the organizing committee composed of SEC staff and an independent overseeing committee to monitor progress of the participants.

During the registration phase, between August 10 and September 10, individual researchers and teams up to five members submitted their identification information and institutions they officially represented. Although the SEC states that any Brazilian citizen over 18 years is eligible, there is a screening process in place: only after the SEC verifies the documents and pre-approves the applicants, they become able to inspect source code.

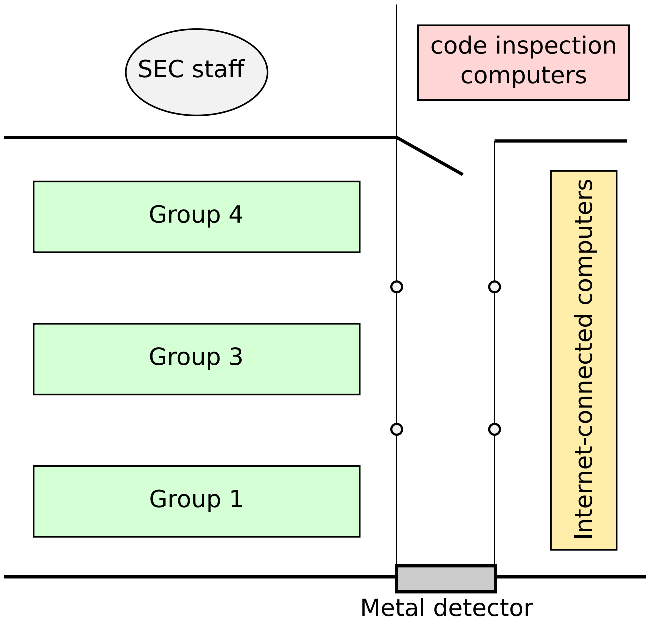


Fig. 4 – Layout for the room where the TPS was conducted, highlighting the areas for Internet access, code review and the stands where groups worked in the trials. Group 4 was formed by merging Group 2 with individual participants from the Brazilian Federal Police.

The code inspection phase started with an opening talk by the SEC staff describing the rules of the challenge and an overview of the system and its implemented security mechanisms. Teams with pre-approved registration whose members signed the NDA were allowed to spend four days at the SEC headquarters in Brasília, between October 3 and 6, inspecting the source code. Participants were allowed to use only computers and tools provided by SEC and take notes on paper. A metal detector prevented entrance of memory devices for copying the codebase, but there was no rigorous control about what pieces of paper entered or left the inspection environment. The inspection computers were offline, but Internet-connected computers were available in a different section of the room (see Fig. 4 for a layout). The rules explicitly stated that researchers *would not have access to cryptographic keys* (Superior Electoral Court, 2017a).

In the next phase, each individual or team had to submit at least one testing plan to be formally approved as a participant. A testing plan must explain the intended attack in some detail, what portion of the attack surface of the system was targeted, the possible outcome in case of success, and the potential impact in the electoral process. All attacks described in a testing plan had to be within the scope defined by the organization. The SEC then decided what testing plans were compatible with the rules, selecting which teams were allowed to take part in the event, with a maximum of 25 participants. At the end, after merging participants from different groups, a total of 16 participants were finally approved, consisting of 4 individual researchers and 3 teams (labeled Groups 1, 3 and 4).

During the trials, teams had five days to execute the previously submitted plans, between November 27 and December 1, from 9 A.M. to 6 P.M. The first day was reserved for preparing the environment and tools. The other four remaining days were used for executing the testing plans against the system, again using SEC computers. The execution of each step of the plans was closely followed by SEC personnel, and every action and result were recorded in a log. If a team decided that some aspect of a testing plan had to be changed, a modification had to be submitted and approval obtained by the overseeing committee. In the beginning of this stage, access to paper became tightly controlled and numbered sheets were distributed to all participants. All pages were checked at the end of each day to prevent information from being exfiltrated through paper. Additionally, leaving the code inspection area to other parts of the room with notes containing pieces of source code was not allowed.

The whole process was painfully bureaucratic and many forms had to be filled and signed by the team leader along the way: submission of modifications or entirely new testing plans for approval; software requisition, to ask software to be installed in the testing computers; authorization for external materials to enter the testing environment; authorization for installing software brought by the participants; official inquiry to the organization committees, to ask technical questions and clarify operational issues; notification of vulnerabilities and mitigation; and experimental conclusions in the last day. For each new request, a sequential number had to be obtained and reserved in a centralized control sheet.

Two reports were published after the event. The first one, written by the overseeing committee, contains the results obtained by the teams and suggestions for improving the event as a whole (Committee, 2017). The second report was written by SEC staff and discusses the vulnerabilities found by researchers and the measures taken by the SEC for mitigation (Superior Electoral Court, 2018). The latter was published in May 2018, officially ending the event.

4. Planning

In this section we detail the progress during the first few phases of the hacking challenge, encompassing both the code inspection and the attack methodologies formulated by the team based on the information collected during the inspection and later submitted as testing plans.

4.1. *Code inspection*

In accordance with the event schedule, we were able to inspect the source code of the voting system. Inspection was performed in the computers provided by the SEC, with Ubuntu 14.04 and some software already pre-installed, such as the Eclipse development environment, a Python interpreter and the standard command line tools. We were not able to enter or leave the code inspection area with any electronic material nor any digital storage media (not even read-only media). For example, we could not bring the source code of a vanilla Linux kernel to check for differences with the SEC custom kernel.

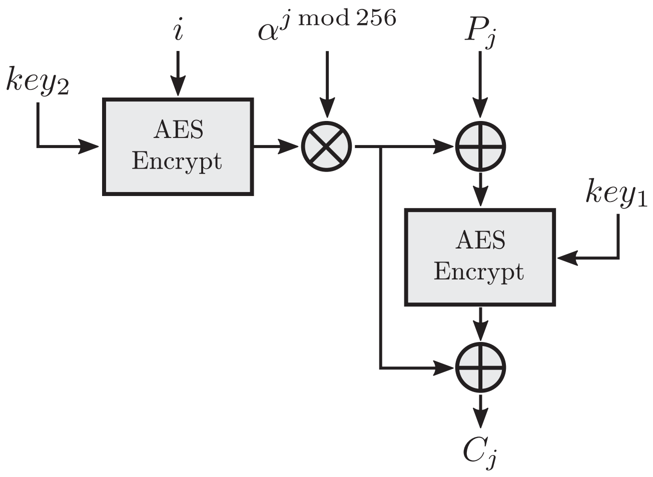


Fig. 5 – Install card encryption scheme based on AES-XTS (adapted from Stallings, 2016).

```
function GET_KEY
    $key_2 \leftarrow \{\}$
    $offset_1 \leftarrow 512 + 7$
    $offset_2 \leftarrow 512 + 128$
    $b \leftarrow get\_byte(offset_1)$
    for $n \leftarrow 0$ to 32 do
        $key_2[n] \leftarrow \text{GET\_BYTE}(offset_2 + n) \oplus b$
    return $key_2$
```

Listing 1 – Obtaining the AES-XTS *key*$_2$ from a install card.

We were not able to install other software in these computers, like more powerful text editors, a C compiler, neither any of our preferred tools.

The main tools we used for searching vulnerabilities were the `grep`[4] command and the `nano`[5] text editor. We quickly realized that some symbols and files were missing, such as the bootloader source code. We were surprised to find out that the SEC staff restricted access to cryptographic keys by *attempting to remove all key material from the source code*, thus presenting a modified and incomplete codebase to the participants.

Using `grep`, we found many potential vulnerabilities. An important one was related to the file system encryption scheme used for the install cards. It employed an encryption scheme based on 256-bit AES-XTS, but the keys were hard-coded into the kernel. To encrypt, AES-XTS uses two keys, as presented in Fig. 5. Below we describe each of the variables observed during the inspection of the encryption scheme:

- *key*$_1$: First part of the AES-XTS key. It is 256 bits long and its value was hard-coded into the kernel.
- *key*$_2$: Second part of the AES-XTS key. It is also 256 bits long and was computed in accordance to Listing 1, where the

[4] GNU Grep: https://www.gnu.org/software/grep.
[5] GNU Nano text editor: https://www.nano-editor.org.

GET_BYTE(n) function returns the n-th byte from the first partition of the install card.
- i: The initialization vector. During the code inspection, we noticed that i was chosen as the *inode* number of the file being encrypted in the file system.
- α is the primitive element of $GF(2^{128})$, where GF denotes Galois Field.
- P_j: The j-th block of plaintext. All blocks except possibly the last one have a length of 128 bits.
- C_j: The j-th 128-bit block of ciphertext.

The SEC implementation of AES-XTS deviates from the standard in that it computes $\alpha^{j \bmod 256}$ instead of α^j. This actually weakens the algorithm, since internal state is now restarted at every 4096-byte block. As we were unable to find any technical justification for this change, we suspect it was an attempt at obfuscation. Decryption follows the same diagram presented in Fig. 5, the only difference is that the ciphertext now serves as input (in place of P_j) and the plaintext is obtained as output (in place of C_j).

Despite cryptographic keys being removed by the SEC staff, the file system encryption key key_1 was still visible in the more recent 3.18 branch of the kernel, due to an operational flaw during the sanitization of the codebase. At this point, we also knew it was possible to extract key_1 by reverse engineering the kernel (after it is decrypted by the bootloader).

4.2. *Submitted testing plans*

We employ the GQM paradigm (*Goal, Question, Metric*) (Basili, 1992) as a simple way of defining objectives and evaluating progress in the hacking challenge. GQM defines an evaluation method in three levels: conceptual (objetives), operational (Research Questions) and quantitative (metrics). The objective specified in GQM has three attributes: purpose, object of study, interested party and context. As an example, the objective compatible with the TPS has the purpose of *analyzing the security* of the *electronic voting system* when it is being *used by voters* in the context of a *real election*.

All the attacks to be mounted had to follow the scope and specification defined for the TPS. The team submitted four testing plans, all approved by the SEC, and listed below in the form of Research Questions (RQs). The testing plans also illustrate how large is the attack surface of the system.

RQ1: Is it possible to extract cryptographic keys from the FC? Since we did not know if the key_1 found during code inspection was correct, this testing plan consisted of reverse engineering the encryption process to extract cryptographic keys using only the install card used for distributing and installing the voting software. The recovered keys could later be used to decrypt sensitive files or authenticate files containing fake election results.

RQ2: Is it possible to violate ballot secrecy by attacking the vote shuffling mechanism? This testing plan aimed at verifying if the vulnerability discovered in the 2012 hacking challenge was indeed fixed.

RQ3: Is it possible to mount an attack through a malicious USB flash drive? The voting machine has two USB ports in the voter terminal and two others in the poll office terminal. This testing plan consisted of employing programmable USB devices, such as the Raspberry Pi Zero and FaceDancer, to impersonate the MSD to collect sensitive information or bypass the chain of trust.

RQ4: Is it possible to remotely execute code in the centralized tabulator? The web platform of the tabulation system was part of the scope of the tests for the first time. Access to the web platform is allowed only through a VPN, however the VPN credentials can probably be extracted from the JE Connect LiveUSB designed by the SEC to set up the VPN connection from inside insecure networks. Compromising the server could allow for tampering with tabulation results. Although the outcome could probably be corrected afterwards by checking the paper version of poll tapes, the attack would potentially slow down the tabulation process and undermine trust in the electoral process.

For each RQ, the metric considered was the number of vulnerabilities explored and demonstrated during the TPS. Hence, a successful exploit would be sufficient to provide positive support to the corresponding RQ.

5. Execution

In the beginning of the first day, we spent a few hours filling paperwork and some time recognizing the testing environment and the equipment available. The routine of filling forms, realizing that we missed some important software package or dependency, and asking for authorization of new incoming software would repeat in the following days. We quickly realized that the single computer provided by the SEC would not be sufficient for the whole team to work and asked for more computers. This was another opportunity to review the source code, since it had been a few weeks since the end of the code inspection phase. We decided to switch all computers to Kali Linux[6] and took notes on what to bring on the next day in terms of software and equipment.

We also made progress decrypting the install cards. Because we were not absolutely certain that the value of key_1 found in the source code or that the fixed position for key_2 were correct, we quickly wrote a small Python decryption script in the code inspection computers invoking command-line OpenSSL[7] to decrypt each individual block. Due to the unavailability of real install cards at that point and a C compiler in the code inspection computers, we tested the program on an encrypted stub file we found in the codebase. The program was painfully slow due to the constant spawn of child processes, but sufficient to validate our hypothesis. Handling padding was a nuisance, because additional bytes were added to fill the last block and written to the file system, but the reported file sizes did not take those into account.

Decrypting the install card. After the decryption script worked successfully in the code inspection computers, we copied the decryption key to the testing computer a few bytes at a time and reimplemented the decryption process as a program by adapting code from another AES-XTS implementation (Teuwen, 2017). In our first decryption attempt, we

[6] Kali Linux – Linux Distribution: https://www.kali.org.
[7] OpenSSL Cryptography and SSL/TLS Toolkit: https://www.openssl.org.

chose to decrypt an Extensible Linking Format (ELF) file called `initje`, a modified version of the Unix `init` daemon, because we knew how the header format (also known as *magic number*) looked like. In this first attempt, the decryption was successful, *i.e.*, we obtained a file with `\x7fELF` as the initial bytes. After that, we wrote a script to decrypt/encrypt files in the install card, and proceeded to inspect it looking for additional vulnerabilities in the authentication chain.

As described in Fig. 3, the process starts with the verification of the BIOS signature by the MSD and finishes with validation of individual files using the detached signatures stored in the files with extension VST. These files contain paths of files along with digital signatures of their contents, in a custom binary format described in ASN.1 syntax. During installation and initialization of the voting machine, the VST files are checked and the signatures verified.

We first observed that extraneous files could be added to the install card without triggering any security alert. If a vulnerability existed in the authentication chain, it would be possible to violate software integrity and consequently execute arbitrary code provided by the attacker, representing a full compromise of the system. This motivated the submission of another testing plan, also presented as an RQ:

RQ5: Is it possible for an external attacker to execute arbitrary code in the voting machine?

By inspecting the VST files, we noticed that two shared libraries used by the voting system (`libapilog.so` and `libhkdf.so`) did not have corresponding detached signatures. These were used, respectively, for logging and an HMAC-based Extract-and-Expand Key Derivation Function (HKDF). We checked that this was indeed an attack vector by first replacing the opcodes of the functions in the two libraries by an `exit` system call, observing the installation was interrupted as expected. We also observed which infected function was the first one to be called and injected code to print the custom message “`FRAUD!`” in the terminal using the `write` syscall during the system initialization. Extending the attack, a simple read-echo loop was created in order to show the possibility of using a regular USB keyboard, writing messages from `stdin` to `stdout`. This possibility of arbitrary code execution was the main attack vector used throughout the rest of the challenge. Given the time restrictions imposed by the event, we decided at the end of the second day to not proceed further with *RQ2*, *RQ3*, *RQ4* and focus on the more promising *RQ1* and *RQ5*.

Manipulating the LOG. To illustrate other possibilities of exploiting our code injection, we performed more attacks in a simulated polling station. In the first one, the constant string `INFO` of the logging library was replaced by the `XXXX` string, showing the possibility of modifying events in the LOG. We verified the success of the attack by observing the modifications in the corresponding file stored in the MR after the election was over.

Violating ballot secrecy. In another attack, the HKDF library was infected to force the derived cryptographic keys to have known values. The HKDF algorithm was used as the key derivation algorithm for encrypting the DRV. This file contains every vote cast in the voting machine and is randomly shuffled in order to avoid the identification of votes based on sequential observations. We replaced the opcodes of the key generation function with the opcodes of the following instructions: `xor eax, eax; ret`. This code just returns the function without any errors. As the function was supposed to store the generated key in a C++ vector passed as argument, this code is equivalent to just returning a zeroed key, since the `std::vector` constructor initializes the content with zeros when `count` is the only argument supplied.

With the library always producing a zeroed key, the DRV file could be trivially decrypted. During the election, a temporary version of this file is stored both in internal memory and in the voting card. Since the voting card can be manually removed from the voting machine without serious consequences, this file could be copied and decrypted, with differences computed at every vote, thus allowing for the violation of ballot secrecy. We demonstrated this attack to the SEC staff in the context of attacking selected votes (by judges, politicians or other public officials), since extracting cards during the election might raise suspicion. The attack against ballot secrecy could have been further improved by injecting code which automates the manual activities and stores consecutive versions of the file.

Tampering with screen contents. With the intent of compromising the memory space of the voting application itself (VOTA), we tried to statically analyze its binary using a disassembler, in order to look for addresses of interesting code excerpts or global variables in the compiled application. However, the binary was packed with UPX[8] and could not be unpacked by the standard UPX command line tool, consuming additional time. It turned out to be a simple matter of UPX getting confused by the digital signature appended to executables. Removing the signature allowed us to normally unpack binaries using the standard UPX tool.

With the unpacked VOTA application binary at hand, we noted that it lacked common exploit mitigation techniques, because it was not in a Position Independent Executable (PIE) format. This simplified our exploitation because targeted contents were stored in fixed addresses, eliminating the need to compute addresses from the process memory mappings. However, both of the unsigned libraries which could be used as attack vectors were linked against multiple executables, whereas we wanted to compromise just the VOTA application. We chose to insert our payloads in the HKDF library, which was linked to only two of the executables – SCUE and VOTA. In order to check whether the library code was running inside the memory space of VOTA, we read a 32-bit word from some address which existed both in SCUE and VOTA but contained different values in each application. If it detected our code was not running inside VOTA, it would just jump and skip the payload.

In order to verify whether we would be able to use the `mprotect` syscall to change the permissions of read-only memory pages, we wrote a payload to modify VOTA’s version string, located in its `.rodata` section. We modified the original string to “The Hour of the Threat” (in Portuguese, “*A Hora da Treta*”). The test was successful and the new string could be found in the installation log and inside the Memory of Results (MR).

[8] UPX: Ultimate Packer for eXecutables – https://upx.github.io.

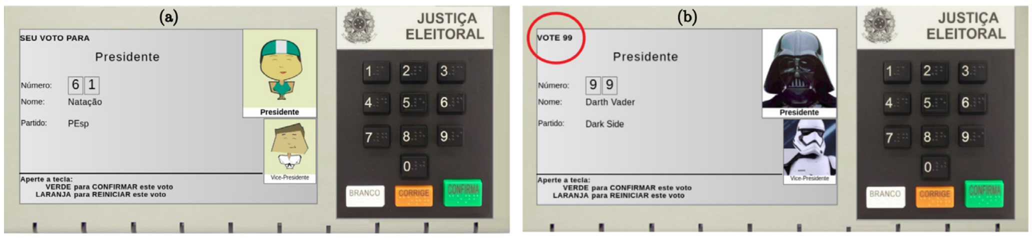
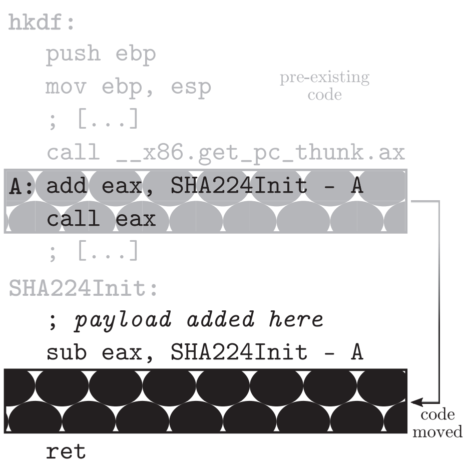


Fig. 6 – Reproduction of the in-memory modification of one of the strings contained in the VOTA application. (a) Original DRE. (b) Compromised DRE – the voting software now advertises the choice of a hypothetical candidate.



```
hkdf:
   push ebp
   mov ebp, esp
   ; [...]
   call __x86.get_pc_thunk.ax
A: add eax, SHA224Init - A
   call eax
   ; [...]
SHA224Init:
   ; payload added here
   sub eax, SHA224Init - A

   ret
```

Fig. 7 – A simple library code infection technique we employed to furtively compromise `libhkdf.so`**.**

In order to escalate this simple payload to a useful and visible attack, we decided to modify a string in the voter screen, which is clearly visible during the voter's interaction with the equipment. The selected string was "YOUR VOTE GOES TO" (in Portuguese, "*SEU VOTO PARA*"). A reproduction of the screen after this attack is shown in Fig. 6(b) – at the top left corner, a message appears asking the voter to cast a vote for candidate number "99".

Once the ability to modify any desired memory page was demonstrated, we improved the HDKF library infection technique. Until this moment, we simply overwrote the `hkdf()` library function to derive a symmetric key. Because of this, our payloads always caused the DRV key to be zeroed, similar to the attack against ballot secrecy. Although an apparently corrupt DRV file would hardly raise any suspicion, since it is only inspected when an audit is performed, we implemented a method to avoid this drawback by preserving HKDF functionality. As illustrated in Fig. 7, we replaced the SHA-224 implementation (which was present in the library but not used by any of the applications) with our payload. Then, we overwrote just an excerpt of the `hkdf()` code with instructions for redirecting the execution flow to the `SHA224Init` function, whose address we compute into the `eax` register taking into account that, unlike the main executable, the library contains Position Independent Code (PIC) and is loaded at a random address. After running the payload, the `SHA224Init` function restores `eax` to its original value, runs a copy of the original excerpt moved from `hkdf()`, and returns. Since the `ret` instruction pops the stack, an excerpt containing instructions that manipulate the stack would present issues, however we fortunately did not face this problem.

Tampering with votes. Continuing to analyze the VOTA binary with a disassembler in the last few remaining hours, we found a method called `AddVote` (in Portuguese, "*`AdicionaVoto`*") with signature `void(uint8_t office, int type, std::string &vote)`. This method was called only when the vote to be cast was already rendered on the screen and the voter pressed the DRE's confirm button. In other words, it was possible to modify its behavior to change votes before storing them, without the voter ever noticing anything abnormal. To compromise `AddVote()`, we wrote a Python script to generate a payload to infect the method with the code below. This code loads in `eax` a reference to the `std::string` which holds the vote, then loads to `edi` a pointer to the string's characters, and finally writes a '9' character to two subsequent addresses (starting at `edi`):

```
mov eax, [ebp+0x14] ; std::string&
mov edi, [eax]  ; char*
mov al, '9'
stosb
stosb
```

Testing a new payload in the DRE took more than 30 minutes due to the sanity self-checks required by the installation process. Therefore we decided a member of our team would test a simpler payload, which just replaced the `AddVote()` method with the `ret` instruction opcode, while another member would simulate the complete payload in the computer, to ensure it was correct before loading it into the DRE. Since the simple payload prevented votes from being stored in the electronic ballot (a `std::vector` containing votes for all the election offices), we observed the following behavior when it was loaded in the DRE: the voter could type and confirm votes for all races, however just after pressing the confirm button for the last one, a consistency check triggered an alert message in the DRE screen stating that the in-memory ballot was empty. Although the complete payload worked correctly when simulated in the computer without requiring any further bug fixes, the tests were interrupted on time at 6PM, and we were not allowed to proceed testing it on the DRE voting machine.

6. Discussion

DRE voting machines are largely criticized in academic literature, mainly due to their design and implementation flaws, and because these electronic voting systems do not allow for external audits/recounts in case the election outcome is disputed. Models manufactured by Diebold were especially subject of multiple security analysis (Calandrino et al., 2007; Feldman et al., 2007). The impact of the vulnerabilities discovered ranged from manipulation of election results to viral infection of voting equipment. Most of the problems were a direct result of insecure engineering practices and the enormous complexity of the voting software, comprising around a million lines of source code. Other works focused on comparably simpler voting systems found similar problems, such as the software attacks on Dutch Nedap DRE machines by Gonggrijp and Hengeveld (2007), and hardware attacks against the Indian EVMs discussed by Wolchok et al. (2010). Paper records are not a panacea either, and hybrid systems with electronic and physical records, such as the one used in parts of Argentina, were also found vulnerable to realistic attacks when analyzed by independent researchers (Amato et al., 2015).

Most of the publicly available literature regarding the Brazilian system has scope limited to official voting procedures, election legislation and informal analysis. The Brazilian Computer Society (the national equivalent of the Association for Computer Machinery – ACM) commissioned a report in 2012 which found important concerns about the security and transparency guarantees of Brazilian electronic elections (van de Graaf, 2002). The report suggests many improvements to the election workflow, but no software vulnerabilities were discussed, although the authors had the opportunity to observe some software development inside the SEC.

Until recently, only inspectors from political parties could examine the source code during a time period before the elections. For this, they must sign an NDA which prevents any public disclosure of the problems observed in the code. These limitations explain the lack of public literature about the security of Brazilian DRE machines. The report published by Aranha et al. after the TPS 2012 (Aranha et al., 2014) was the first technical document containing a detailed analysis of the security mechanisms implemented in the voting system. However, the report focuses more on the vulnerabilities of the ballot shuffling mechanism and how the researchers were able to exploit them under the restrictions of the hacking challenge, although some discussion is dedicated to the insecure storage of cryptographic keys and inherent limitations of the software integrity checking mechanism. Our work should help to fill this gap and to accurately update the state of Brazilian voting technology to the international technical community. We split the discussion between the software integrity and ballot secrecy properties and finish arguing how our results invalidate official security claims published by the SEC.

6.1. *Software integrity*

Decrypting the install cards was all that was needed to discover two shared libraries without detached signatures, and thus amenable for arbitrary code injection attacks. Circumventing the encryption mechanism thus gave disproportionate capabilities to an attacker, an unexpected effect. Although we obtained the encryption key directly from the source code, which greatly accelerated our progress, we claim that an external attacker would also be able to recover the encryption key embedded in the bootloader and proceed with decrypting the kernel image. On possession of a decrypted kernel image, the attacker would become capable of both decrypting the install cards and removing the second encryption layer protecting the application-level authentication keys. The latter provides a huge amount of power to the adversary, who becomes capable of forging files and corresponding digital signatures protecting poll tapes, software components, the LOG and the DRV. Interestingly, in the last day of the hacking challenges, a group composed of forensic experts from the Brazilian Federal Police was able to recover the kernel image in plaintext through reverse engineering by moving the bootloader to a standard address and running it inside a virtual machine emulator. This way, there is strong positive supporting evidence for RQ1*: as it was possible to extract cryptographic keys from the install card*.

We conclude that integrity of the software and results depend ultimately on the secrecy of a symmetric key embedded into the bootloader. This key is trivially accessible by the entire voting software development team, and visible to external attackers because it is stored in plaintext inside the install cards. In fact, this mechanism does not amount to encryption *per se*, but only to a much weaker form of *obfuscation*. In retrospect, these vulnerabilities are not exactly new. The report by Aranha et al. (2014) already pointed out how the file system encryption keys were insecurely stored in the source code in the 2012 version of the voting software. At that time, the same key and initialization vector (IV) were shared among all voting machines, for encrypting files using AES-256 in CBC mode. The only improvements we observed in the 2017 version were switching to variable IVs and adopting the XTS mode.

After the install card was decrypted, two shared libraries were found without signatures, supporting RQ5*: as it was possible to inject and execute arbitrary code*. The two shared libraries without detached signatures were not signed because generating the list of files to be signed is apparently not an automated process. In their report, the SEC development team states that the libraries still had kernel-level RSA signatures appended to the files, but a sign mismatch bug in the verification code (and its unit test) prevented the manipulation to be detected (Superior Electoral Court, 2018). This suggests a development process in need of urgent revision of its critical procedures.

The staff also states that file system encryption keys will not be hard-coded anymore in future versions of the software, but computed on-the-fly with help of the BIOS, increasing the level of obfuscation (Superior Electoral Court, 2018). However, the keys will still be the same for all voting machines and will remain predictable to an insider attacker with access to the BIOS contents or the development process. This design choice was justified by the SEC after observing that not all voting machines have the MSD device available, which limits the possibility of using it for storing cryptographic keys. Because of the imposed requirement that any voting machine must be able to replace any other on election day, they consider risky to have different versions of the software in operation. This suggests further that the overall security of the system is dictated by the oldest model in operation, in this case the 2007 model not equipped with the MSD.

We recommend the SEC to revise its development process, adopting best practices by automating critical procedures and implementing negative testing countermeasures. The list of files to be signed should not be generated by manually hard-coding file names in a script, and testing of signature verification should not only be evaluated under ideal circumstances (correct key and message). We further recommend the install card encryption keys (and other cryptographic keys) to be segregated in the minimal unit possible (polling place, neighborhood or city), to reduce overall impact in case of leakage. In the longer term, a key distribution architecture should be implemented and all cryptographic keys should be moved inside the MSD security perimeter.

6.2. *Ballot secrecy*

The DRV file stores a table separated into sections, where each section is devoted to a different race. This table shuffles the votes cast by the voters to disassociate the order of the voters and their votes. It was introduced by law to replace the paper trail after the failure in implementing paper records in 2002. The SEC claims it supposedly permits independent verification of election results (Superior Electoral Court, 2014), but the file is produced by the same software which tallies the votes. Any successful attack against the tallying software can also compromise the integrity of the DRV. For this reason, Aranha et al. (2014) concludes that the DRV file does not serve any purpose besides violating ballot secrecy if implemented insecurely. Preventing attacks against the DRV relies on secure random number generation.

There are multiple such algorithms being used across the code base to satisfy the randomness needs of the plethora of cryptographic primitives for encryption and authentication deployed in the voting software. The Elgamal signatures computed by the MSD rely on the weak `xorshift` family (Marsaglia, 2003) of pseudo-random number generators (PRNG). The mechanism for shuffling votes inside the DRV file, a critical component for ballot secrecy, was implemented through a combination of two other generators: reading directly from `/dev/urandom` or from a customized PRNG based on a 32-bit variant of the 64-bit version of the obscure Sapparot-2 algorithm. The generator alternates between the two algorithms in case any of them fails. In its original version, the Sapparot-2 algorithm is clearly not suited for cryptographic applications, as explicitly advertised by the author (Levin, 2005).

Although a significant improvement over the previous shuffling mechanism implemented with `srand()/rand()`, the modifications we observed are clearly not sufficient. Even if the new version of the implemented mechanism appears much harder to exploit due to frequent mixing of operating system entropy in the internal state, the inadequate choice of algorithms after five years of development is surprising. The replacement algorithm was not vetted by the cryptographic community and does not satisfy minimal security requirements for such a critical file. The recommendations in the previous report were not fully adopted, since the file layout still lacks defense-in-depth protections by removing unused slots in the DRV table and the PRNG remains nonstandard (Aranha et al., 2014). We reinforce the same recommendations, assuming that the DRV must still be produced to satisfy legal requirements: (i) remove unused slots corresponding to absentees to prevent exhaustive search in the seed space; (ii) adopt stronger standardized PRNG algorithms or read from `/dev/urandom` directly, if collected entropy is of enough quality. In the longer term, we further recommend the DRV file to be eliminated, and the law mandating it to be changed.

6.3. *Security claims*

There is no document formalizing the threat model or security goals considered by the SEC during the development of the electronic voting system. This complicates security analysis, since the adversary to protect against becomes a moving target, conveniently changing depending on the attack under discussion. Fortunately, the SEC published a Q&A document defending the design of some of the security mechanisms (Superior Electoral Court, 2014), in response to the results obtained in the hacking challenges. This document is useful to understand the rationale behind some of the design decisions. As with any paperless DRE system, all security properties ultimately depend on integrity of the voting software and hardware, and their resistance against tampering. Our results contradict several of the claims in that document, as we elaborate below. The original writing is in Portuguese, but we attempt to provide translations as close as possible.

The second question in page 10 states the security goal concerning software integrity:

> *It is not possible to execute unauthorized applications in the voting machine. Along the same way, it is also not possible to modify an application in the machine.*

We were able to include additional files and modify two shared libraries in the install card, with the software installation process being completed without hassle. The software installed in the machines preserved the modifications and its execution later violated the integrity of running software.

Pages 12–14 give details about the adversarial model:

> *The voting machine is not vulnerable against external attackers. (...) This is guaranteed by several security mechanisms, based on digital signatures and encryption, which create a chain of trust between hardware and software and prevent any violation of the voting machine.*

DRE machines are notably insecure against malicious insiders with control over the voting software. Our successful attack also demonstrates how an external attacker in control of install cards can manipulate voting software before it is installed in the machines. Because each card installs software in 50 voting machines, the impact is amplified, reducing the logistic obstacles and cost of a large-scale attack.

Page 22 establishes security goals for the LOG file:

> *The log file is another transparency and auditing mechanism made available by the SEC.*

The fact that the shared library handling log events lacked digital signatures completely removes the possibility of using the LOG as an auditing mechanism, because the generated events may be under adversarial control. An attacker would then be able to erase specific events or manipulate security-sensitive metrics, such as the false positive rate reported for the fingerprint identification system. This is naturally true of any electronic record.

The last question clarifies the expected security properties of the file system encryption, here transcribed in more detail:

The objective of the file system encryption is to impose an additional barrier to an external attacker with little or no knowledge about the software organization in the voting machine. This way, an attacker would find obstacles to start analyzing the memory card contents.

There is a single secret key used for encrypting file systems in all memory cards. If this key were not unique, it would be impossible to replace a malfunctioning machine with another one, and any auditing in the voting machines would be compromised. However, stating that possessing the encryption key makes possible to generate cards "with different content" is incorrect.

It is worthy noting that the file system encryption is not the sole software security mechanism in the voting machine. All files which require integrity and authenticity are digitally signed. This is the case, for example, of the voting machine applications and election metadata, and the poll tapes, DRV, among others. Files requiring secrecy are also encrypted. In all these cases, other keys are employed. These signature and encryption mechanisms prevent the memory contents from being manipulated.

The file system encryption is thus claimed to be one of many security barriers against external attackers. It is designed as a first obstacle to attackers without much information about the system, having other cryptographic mechanisms as stronger defenses against more sophisticated attackers. While the former is technically correct, since the file system encryption is actually just obfuscation, we observed that capturing the encryption keys provided a disproportionate power to the attacker, who becomes able to choose or reveal more important cryptographic keys. This happened because decryption allowed us to fully inspect the contents of the install card and detect serious vulnerabilities in the integrity checking mechanism, violating the software integrity claims and the main security properties of the system as direct consequence.

6.4. *Countermeasures*

Beyond the technical improvements already discussed in this section, the Brazilian voting system fundamentally lacks a mechanism to allow independent verification of election results that does not depend on the security or integrity of hardware and software. In the academic literature and practice of elections, this concept is commonly referred as *software independence* (Rivest, 2008) and implemented through either a voter-verifiable paper audit trail (VVPAT) that is kept within the machine; or end-to-end (E2E) auditing measures. In the case of VVPAT, disputes about the election outcome are typically settled by checking increasingly larger samples of the voter-verified physical record with the electronic counts until confidence is high enough. This procedure is called a *risk-limiting audit* (Lindeman and Stark, 2012) and requires integrity of the physical record to be effective. E2E systems usually provide the voter with some encrypted record that can be used to verify that his or her ballot is correctly included in the outcome (*individual verifiability*), but without giving the same power to an adversary; and a global mechanism to verify that all votes were correctly counted (*universal verifiability*) (Benaloh et al., 2013; Chaum et al., 2008). A limitation in usability of E2E systems is the addition of user-facing cryptographic mechanisms that may be difficult to understand for layman voters.

In Brazil, a major obstacle to implement a software independence mechanism is the understanding by the Supreme Court that attaching a physical record to the voting system violates ballot privacy, the latter a constitutional requirement. Despite VVPATs having been successfully used in a significant fraction of the polling places in 1996 and 2002, multiple attempts to later introduce the transparency mechanism by law have been deemed unconstitutional. In 2009, the Brazilian Congress approved the introduction of a paper audit trail for the 2014 general election in which individual voter-verified ballots would be printed and digitally signed by the voting machine to prove their authenticity. In a follow-up decision, the Supreme Court interpreted the law as if the voting machines would sign the voter's identity together with their choices, trivially violating ballot secrecy (S.C. of Brazil, 2014), and declared the law unconstitutional. In the latest attempt, a revised law without the digital signature requirement was approved with bipartisan support in 2015, but deployment was suspended a few months before the 2018 general election, after a prototype was already built by the electoral authority. Again the new law faced claims of unconstitutionality and ended up suspended. In both decisions, the Supreme Court has repeatedly claimed that VVPATs would introduce a high risk of physical records being used for voter coercion, notwithstanding the fact that physical records would be printed behind the voting booth and stay in the polling place (Superior Electoral Court, 2017b). Despite technical evidence to the contrary, another recurrent claim put forward both by the SEC and the Supreme Court is that the current system provides sufficient security, thus adding a physical record would only serve to increase the system's complexity and attack surface. It is not clear how the debate will proceed further, but introducing a physical record for auditing and recounting would be the cheapest short-term option to increase transparency of the voting system without requiring a major overhaul of its design and operation (Aranha and van de Graaf, 2018).

7. Conclusion and perspectives

We thank the SEC for the opportunity to contribute with improving the security of the Brazilian voting machines and give brief suggestions about how to improve effectiveness of the hacking challenges: minimize the bureaucracy and staff intervention during the event; improve agility of internal processes to authorize entry of documents and software packages in the testing environment; enlarge the scope by including the fingerprint identification system and parts of transmission/tabulation infrastructure; increase the duration of the event and reduce the dependence on secret source code by making it widely available. In particular, we kindly ask readers to not extrapolate the time consumed by our team during the challenges as an estimate of the time required to mount an attack against real elections, as the number and impact of artificial restrictions was substantial.

We conclude by stating that the Brazilian voting machine software still does not satisfy minimal security and transparency requirements, and is very far from the maturity expected from a 20-year mission-critical system. We recommend the SEC to carefully revise their development practices and consider adopting voter-verified paper trails or end-to-end verification measures in the system to provide stronger guarantees of its correct functioning on election day. We hope that our findings contribute to the ongoing debate in Brazil concerning the adopting of paper records as a way to improve security and transparency of the voting system. The code written during the hacking challenge and additional files can be found in the repository available at https://github.com/epicleet/tps2017.

REFERENCES

Amato F, Oro IAB, Chaparro E, Lerner SD, Ortega A, Rizzo J, Russ F, Smaldone J, Waisman N. Vot.Ar: una mala elección. https://ivan.barreraoro.com.ar/vot-ar-una-mala-eleccion/; 2015.

Aranha DF, van de Graaf J. The good, the bad, and the ugly: two decades of e-voting in Brazil. IEEE Secur Privacy 2018;16(6):22–30.

Aranha DF, Karam MM, Miranda A, Scarel F. Software vulnerabilities in the Brazilian voting machine. In: Zissis D, Lekkas D, editors. In: Design, development, and use of secure electronic voting systems. IGI Global; 2014. p. 149–75.

Aranha DF, Ribeiro H, Paraense ALO. Crowdsourced integrity verification of election results – an experience from Brazilian elections. Ann Télécommun 2016;71(7–8):287–97.

Aumasson J, Bernstein DJ. SipHash: a fast short-input PRF. In: Lecture Notes in Computer Science, 7668. Springer; 2012. p. 489–508.

Barreto PSLM, Naehrig M. Pairing-friendly elliptic curves of prime order. In: Proceedings of the 12th international conference on selected areas in cryptography. Kingston, ON, Canada; 2005. p. 319–31. SAC'05

Basili VR. In: Technical Report. Software modeling and measurement: the goal/question/metric paradigm. MD, USA: University of Maryland at College Park; 1992.

Benaloh J, Byrne MD, Eakin B, Kortum PT, McBurnett N, Pereira O, Stark PB, Wallach DS, Fisher G, Montoya J, Parker M, Winn M. In: Proceedings of EVT/WOTE. Star-vote: a secure, transparent, auditable, and reliable voting system. USENIX Association; 2013.

Bernstein DJ, Duif N, Lange T, Schwabe P, Yang B. High-speed high-security signatures. In: Lecture Notes in Computer Science, 6917. Springer; 2011. p. 124–42.

Boneh D, Boyen X. Short signatures without random oracles and the SDH assumption in bilinear groups. J Cryptol 2008;21(2):149–77.

S.C. of Brazil. Claim against unconstitutionality No. 4543, (In Portuguese). http://portal.stf.jus.br/processos/downloadPeca.asp?id=267161872&ext=.pdf; 2014.

Calandrino JA, Feldman AJ, Halderman DWJA, Yu WPZH. Source code review of the diebold voting system. Available at https://jhalderm.com/pub/papers/diebold-ttbr07.pdf; 2007.

Chaum D, Essex A, Carback R, Clark J, Popoveniuc S, Sherman AT, Vora PL. Scantegrity: end-to-end voter-verifiable optical-scan voting. IEEE Secur Privacy 2008;6(3):40–6.

Committee TO. Report of the overseeing committee of the public security tests, 2017 edition (In Portuguese). http://www.justicaeleitoral.jus.br/arquivos/relatorio-da-comissao-avaliadora-do-teste-publico-de-seguranca-2017; 2017.

Feldman AJ, Halderman JA, Felten EW. In: Proceedings of electronic voting technology workshop (EVT). Security analysis of the diebold accuvote-ts voting machine. USENIX Association; 2007.

Gonggrijp R, Hengeveld W. In: Proceedings of electronic voting technology workshop (EVT). Studying the nedap/groenendaal ES3B voting computer: a computer security perspective. USENIX Association; 2007.

van de Graaf RFCJ. Electoral technology and the voting machine – report of the Brazilian computer society, (in Portuguese). Available at http://www.sbc.org.br/index.php?option=com_jdownloads&Itemid=195&task=view.download&catid=77&cid=107; 2002.

Levin IO. Sapparot-2: fast pseudo-random number generator. http://www.literatecode.com/get/sapparot2.pdf; 2005.

Lindeman M, Stark PB. A gentle introduction to risk-limiting audits. IEEE Secur Privacy 2012;10(5):42–9.

Marsaglia G. Xorshift RNGs. J Stat Softw 2003;8(14):1–6. doi:10.18637/jss.v008.i14.

Rivest RL. On the notion of 'ssoftware independence' in voting systems. Philos Trans R Soc A: Math Phys Eng Sci 2008;366(1881):3759–67.

Stallings W. Cryptography and network security: principles and practice. 7th ed. Pearson; 2016.

Superior Electoral Court. Execution of testing plan, (In Portuguese). https://web.archive.org/web/20160107163923/, http://www.tse.gov.br/internet/eleicoes/arquivos/Teste_Sergio_Freitas.pdf; 2009.

Superior Electoral Court. Frequently asked questions (FAQ) about the Brazilian voting system, 2nd ed(In Portuguese), http://www.justicaeleitoral.jus.br/arquivos/tse-perguntas-mais-frequentes-sistema-eletronico-de-votacao; 2014.

Superior Electoral Court. Resolution number 23,444, (in Portuguese). Available at http://www.tse.jus.br/legislacao/codigo-eleitoral/normas-editadas-pelo-tse/resolucao-no-23-444-de-30-de-abril-de-2015-2013-brasilia-2013-df; 2015.

Superior Electoral Court. Public security tests of the Brazilian voting system: compendium, (In Portuguese). http://www.tse.jus.br/hotsites/catalogo-publicacoes/pdf/teste-publico-de-seguranca-2016-compendio.pdf; 2016.

Superior Electoral Court. Call for participation in the public security tests, (In Portuguese). http://www.tse.jus.br/hotsites/teste-publico-seguranca-2017/arquivos/TPS-testes-publicos-seguranca-edital.pdf; 2017a.

Superior Electoral Court. Draft of resolution concerning the electoral procedures for implementing a paper trail in 2018, (In Portuguese). http://www.justicaeleitoral.jus.br/arquivos/tse-audiencias-publicas-voto-impresso; 2017b.

Superior Electoral Court. Responses to the vulnerabilities and suggestions for improvement observed in the public security tests, 2017 edition (In Portuguese). http://www.justicaeleitoral.jus.br/arquivos/relatorio-tecnico-tps-2017-1527192798117; 2018.

Teuwen P. python-cryptoplus, a python implementation of AES-XTS. Available at https://github.com/doegox/python-cryptoplus; 2017.

Wolchok S, Wustrow E, Halderman JA, Prasad HK, Kankipati A, Sakhamuri SK, Yagati V, Gonggrijp R. Security analysis of India's electronic voting machines. In: Proceedings of ACM conference on computer and communications security. ACM; 2010. p. 1–14.

Diego F. Aranha is an Assistant Professor in the Department of Engineering at Aarhus University. He holds a Ph.D. degree in Computer Science from the University of Campinas. His professional experience is in Cryptography and Computer Security, with special interest in cryptographic engineering and security analysis of real-world systems. He coordinated the first independent team who explored software vulnerabilities in the Brazilian voting machine. He received the Google Research Award for Latin America twice for research in privacy, and the MIT TechReview's Innovators Under 35 Brazil Award for his work in electronic voting.

Pedro Y.S. Barbosa is a security engineer at a Big Four multinational technology company. He has a Ph.D., M.Sc., and B.Sc. in Computer Science from the Federal University of Campina Grande, and a B.Tech. in Telematics from the Federal Institute of Paraíba. His areas of interest include security and privacy enhancing technologies. In his free time, he enjoys participating in Capture The Flag competitions (hacking and security), playing for the Epic Leet Team.

Thiago N.C. Cardoso has a M.Sc. and a B.Sc. in Computer Science from the Federal University of Minas Gerais. His areas of interest include machine learning, data mining and privacy preserving data analysis. Currently he is the Chief Technology Officer at Hekima, startup that builds end-to-end AI systems. He is also a Capture The Flag player for the Epic Leet Team.

Caio L. Araújo is currently pursuing his B.Sc. degree in Computer Engineering at the Federal University of Pernambuco. He has studied Computer Security over the past seven years. His interests include reverse engineering, generative art and weird machines. He is also a Capture The Flag player for the Epic Leet Team.

Paulo Matias is an Assistant Professor in the Department of Computing at the Federal University of São Carlos. He holds a Ph.D. degree in Computational Physics from the São Carlos Institute of Physics at the University of São Paulo. His professional experience is in embedded systems and custom hardware architectures, with a special interest in their security properties. He is also a Capture The Flag player for the Epic Leet Team.