The Return of Software Vulnerabilities in the Brazilian Voting Machine

Preprint · March 2018

DOI: 10.13140/RG.2.2.16240.97287

5 authors, including:

Diego F. Aranha
Aarhus University
129 PUBLICATIONS **1,198** CITATIONS
SEE PROFILE

Pedro Yóssis Silva Barbosa
Google Inc.
18 PUBLICATIONS **121** CITATIONS
SEE PROFILE

Thiago Nunes Coelho Cardoso
Federal University of Minas Gerais
14 PUBLICATIONS **112** CITATIONS
SEE PROFILE

Caio Lüders
Federal University of Pernambuco
6 PUBLICATIONS **11** CITATIONS
SEE PROFILE

The Return of Software Vulnerabilities in the Brazilian Voting Machine

Diego F. Aranha
University of Campinas

Pedro Yóssis Silva Barbosa
Federal University of Campina Grande

Thiago Nunes Coelho Cardoso
Hekima

Caio Lüders de Araújo
Federal University of Pernambuco

Paulo Matias
Federal University of São Carlos

Abstract

This paper presents a detailed and up-to-date security analysis of the voting software used in Brazilian elections. It is based on results obtained by the authors in a recent hacking challenge organized by the Superior Electoral Court (SEC), 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 those libraries opened the possibility of executing arbitrary code in the equipment, violating the integrity of the running software. Our progress is described chronologically, 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 5 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 an expected 144 million voters in the upcoming 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).

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 this year, with an ongoing debate about how the law should be interpreted and what is the mandated coverage in number of polling places [15].

As a response to the frequent calls for increased transparency, the Superior Electoral Court (SEC) started organizing hacking challenges in 2009, officially called *Public Security Tests of the Electronic Voting System*. 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 conducted in 2009 and 2012, the event became mandatory and now happens from 6 months to one year before the elections [24].

Under a turbulent and polarized political environment, frequently contaminated by corruption scandals, 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.* [2]. 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 with the system. As the main concerns, the authors

[1] Based on 2016 statistics (in Portuguese): `http://www.tse.jus.br/eleitor-e-eleicoes/estatisticas/eleicoes/eleicoes-anteriores/estatisticas-eleitorais-2016`

detected insecure storage and massive sharing of cryptographic keys, insufficient integrity verification, and a faulty software development process conducted under an inadequate adversarial model.

Our contributions. We continue and update the efforts regarding the analysis of Brazilian voting machine software and its security, 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 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 all requirements to do so were satisfied by the vulnerabilities detected at that point. 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 a supposedly correct 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 and some of its the security mechanisms. In Section 3, we summarize the history and discuss the format, limitations and results in previous hacking challenges of the Brazilian voting machines. Section 4 continues by reporting about our day-to-day progress when participating in the 2017 hacking challenge. Section 5 reviews related work and analyzes our findings in context, tracing a historical perspective with relation to other security analysis of the same system. Section 6 finishes the paper.

2 Background

A surprising characteristic of Brazilian elections is that the entire 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 electoral authority to consider the deployment of electronic devices for collecting votes. In 1982, the SEC started the move to digital 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 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 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 voting population.

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 electors by their registration number or fingerprint, and a voter terminal where votes are cast via a keyboard reminiscent of a modern telephone. The full machine is shown in Figure 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 memory card, called *voting card* (or "*flash de votação*"), is inserted during elections in the same external slot for providing voter and candidate registration numbers. 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-Disclose Agreement (NDA) in 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 through install cards: voting software is installed in the machines through the flash memory 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 then be initialized from internal memory, and a hardware test is performed on 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 [7], instantiated with a 160-bit Barreto-Naehrig curve [5], and the resulting signature is used as witness, called *correspondence code* (or "*resumo da correspondência"*). A database containing the assignments is published afterwards.

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 7AM and 8AM. This is an official public document supposedly attesting that no votes were computed for any candidates before the start of the elections.

2. The election official opens the voting session at 8AM by typing a command in the election official terminal.

3. The voters provide identification information and are authorized to cast votes in the machines.

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

Figure 1: The two terminals of the Brazilian 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 connecting the two allows software executed in the election official terminal to authenticate voter data (registration number or fingerprint) stored in the voter terminal.

4. The election official closes the voting session at 5PM, 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), an electronic 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 containing a GNU/Linux distribution. This system establishes a secure connection with the election authority 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.

2.3 The software ecosystem

The whole software codebase has a complexity in the order of tens of millions of lines of code, including all of its 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 voting machine software source code alone has more than ten million lines [8] 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 Apuração*"), 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 there is the inclusion of device drivers and modification of some standard security mechanisms. The codebase is currently 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 in 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 an encrypted repository of cryptographic keys in AES-256, 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 [4]. Figure 2 presents how the encryption layers are organized, with modules in the top containing the keys to decrypt modules immediately below.

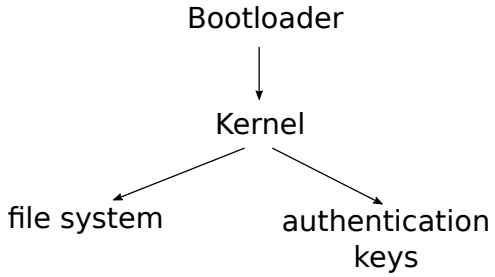

Figure 2: Chained layers of encryption in the install card. Arrows denote possession of encryption keys to decrypt the next layer.

Authentication. In terms of authentication, the BIOS, bootloader and kernel images are digitally signed using ECDSA, instantiated with the NIST P-521 curve. Figure 3 presents the voting machine 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 public key for verification is embedded in the kernel and used by the loader 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 Brasileira de Inteligência*"). The poll tapes have QR codes encoding partial results in a machine-readable format, to facilitate automated comparisons with published results [3], and these are digitally signed using Ed25519 [6].

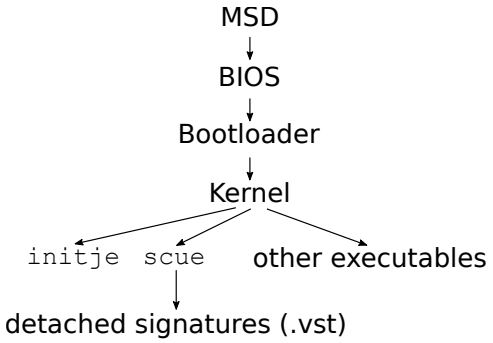

Figure 3: Voting machine chain of trust, starting with the MSD. Arrows denote signature verification. Detached signatures for installation files are verified by the SCUE module.

Random numbers. Many of the deployed cryptographic algorithms for encryption and authentication require random numbers, and there are multiple algorithms across the codebase. The Elgamal signatures computed by the MSD rely on the `xorshift` family [21] of pseudo-random number generators (PRNG). The mechanism for shuffling votes inside the DRV file is implemented through a combination of two other generators: reading directly from `/dev/urandom` or from a customized userland PRNG based on a 32-bit variant of the 64-bit version of an obscure algorithm called Sapparot-2 [20].

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 machines running GEDAI execute an operating-system-level suite of security software in an attempt to prevent tampering dur-

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

ing 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 Segurança*" 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 edition of the TPS, organized in 2009, researchers did not have access to the source code of the voting machine 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 [11]. 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 monetary 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 election. The attack was based only on public data published in the DRV file and superficial knowledge about how the votes were stored in a hash table. The main attack vector was a vulnerability in the vote shuffling code involving a call to `srand(time(0))` 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 authority), 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 include massive sharing of cryptographic keys, insecure storage of secret key material and inadequate choice of algorithms. Aranha *et al.* [2] provide a detailed first-hand account of the event, vulnerabilities and aftermath.

The challenges resumed only in 2016, when the NDA introduced in that edition ended up alienating a large portion of the local technical community. This edition saw the first successful attack against the integrity of results, presented again by Sergio Silva [13]. 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 scope was also extended to include software components from the tabulation system, the MSD firmware and the GEDAI system for generating install cards. Although it has been progressively deployed in the past ten years, the fingerprint identification system is still considered out of scope.

3.2 The 2017 edition

This work reports our findings collected during last year's 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 [14]. Multiple committees were involved in the organization, the main ones were the organizing committee composed of SEC staff, and an independent overseeing committee who monitored 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.

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 connected computers were available in a different section of the room for Internet access. Figure 4 presents a layout of the room and its main sections. The rules explicitly stated that researchers would not have access to cryptographic keys [14].

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 must 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. There were tie-breaking criteria for when the capacity would be reached, such as prioritizing teams who did not ask for travel expenses to be covered. The authors did not ask for a stipend to maintain financial independence and increase likelihood of selection. 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 Group 1 – the authors, 3 and 4).

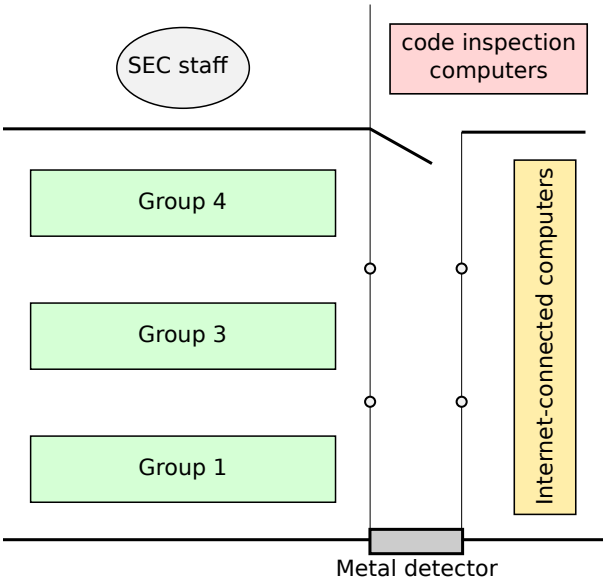

Figure 4: Layout for the room where the TPS was conducted, highlighting the areas for Internet access, code review and the stands where groups worked in the trials. Group 4 was formed by merging Group 2 with individual participants from the Brazilian Federal Police.

During the trials, teams had five days to execute the previously submitted plans, between November 27 and December 1, from 9AM to 6PM. 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 code portions from being exfiltrated through paper. Additionally, leaving the code inspection area to the stands or Internet-connected computers 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 machines; 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 before submission.

After the public security tests were finished, two reports were published. The first one, written by the overseeing committee, contains the results obtained by individual teams and suggestions for improving the public test as a whole [10]. The second report was written by SEC staff and discusses the vulnerabilities found by researchers and the measures taken by the SEC for mitigation [16].

4 Chronology

This section presents our day-to-day progress when participating in the TPS, from before the event (code inspection and preparation of the testing plans) to the actual trials (the five days of testing).

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. We were not able to install other software in these machines, like more powerful text editors such as `vim`[4], a C compiler, neither any of our preferred tools.

The main tools we used for searching vulnerabilities were the `grep`[5] command and the `nano`[6] 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 Figure 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 $get\_byte(n)$ 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 one 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.

[4]Vim – the ubiquitous text editor: `http://www.vim.org`
[5]GNU Grep: `https://www.gnu.org/software/grep`
[6]GNU Nano text editor: `https://www.nano-editor.org`

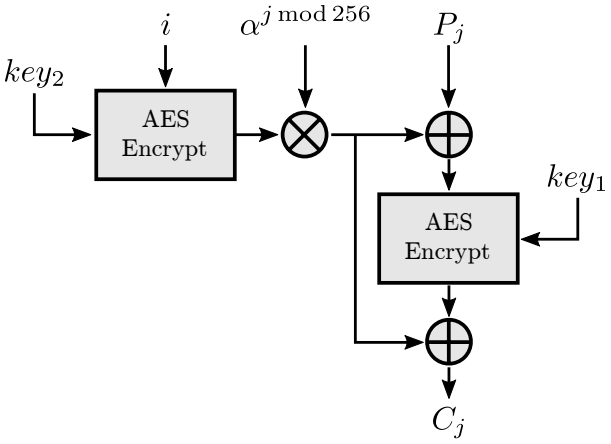

Figure 5: Install card file system encryption scheme based on AES-XTS (adapted from [22]).

Listing 1 Obtaining key_2 used to encrypt/decrypt the install cards using AES-XTS.

```
function GET_KEY2
    key2 ← {}
    offset1 ← 512 + 7
    offset2 ← 512 + 128
    b ← get_byte(offset1)

    for n ← 0 to 32 do
        key2[n] ← get_byte(offset2 + n) ⊕ b

    return key2
```

The decryption uses the same block scheme presented in Figure 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 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 submitted and had approved four test plans ranging from retrieving cryptographic keys to using extraneous USB devices for manipulating the voting system. Each one is described in detail in the next subsections, and they illustrate how large is the attack surface of the system.

Extraction of cryptographic keys. During the code inspection phase, multiple references to cryptographic keys were found in the source code, although most of the particular files containing the definitions were removed

from the available source. Because cryptographic keys are shared among all DRE machines, obtaining the key from a single card means that all DREs could be compromised.

This test plan consisted of, through reverse engineering, trying to extract the cryptographic keys using only the install card used for distributing and installing the voting software. The keys could later be used to decrypt sensitive files or authenticate files containing fake results.

Breach of ballot secrecy. In a previous challenge, Aranha *et al.* found a vulnerability that allowed to violate ballot secrecy for entire polling places or selected voters [2]. This was possible because the pseudo-random number generator used for shuffling votes in the DRV was seeded with a timestamp, a predictable value. This value, easily obtained from the zero tapes, allowed the reversal of the shuffling algorithm and the obtaining of the votes in the same order that they were cast. After the 2012 edition of the hacking challenges, the SEC reportedly replaced the shuffling mechanism with a new one.

This testing plan consisted of checking if this vulnerability was indeed fixed and that the DRV pseudo-random number generator was improved to protect ballot secrecy.

Insertion of malicious USB devices. The DRE has two USB ports in the main voter terminal and another one in the election official terminal. All of those are connected to the same bus shared by internal USB devices, including the MSD. The MSD is actually an ARM7 microcontroller exposed to the system as an USB Human Interface Device (HID). Both the firmware and the device drivers for the MSD are custom-made for the DRE. Although the device drivers implement a challenge-response protocol aiming to authenticate the presence of the HSM, we found during code inspection that the parsing code in the driver apparently lacked bounds checking and did not appear to be carefully designed to avoid buffer overflows.

This testing plan consisted of using programmable USB devices, like Raspberry Pi Zero and FaceDancer, to impersonate the MSD. By fuzzing its communication protocol, potential code execution vulnerabilities [23] could be found.

Remote code execution on the web platform. The web platform of the tabulation system was part of the scope of the tests for the first time. The platform is comprised of two main components: *RecArquivos*, the application which receives and processes the digital version of the poll tapes produced by the voting machines, and *InfoArquivos*, the application which monitors the tabulation status. Access to the web platform is allowed only through a VPN, however the VPN credentials could probably be extracted from the JE Connect LiveUSB designed by the SEC to set up the VPN connection from inside insecure networks.

Both web applications are hosted in JBoss 6, an application server for which multiple vulnerabilities have been disclosed [17]. The SEC did not reveal which release of JBoss 6 was running on their servers, so we planned to scan for these vulnerabilities in order to check whether the servers had the latest patches applied. We also intended to check the web applications source code for other potential Remote Code Execution vulnerabilities. Compromising those servers could allow for tampering with tabulation results. Although results could probably be corrected afterwards by checking the paper version of poll tapes, the attack would potentially slow down and undermine trust in the electoral process.

4.3 Day 1: Assembling the work environment

In the beginning of the first day, November 28, 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 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 running Ubuntu. 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 machines to Kali Linux[7] 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 was correct, we quickly wrote a small Python decryption script in the code inspection computers invoking command-line OpenSSL[8] 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.

[7] Kali Linux – Penetration Testing and Ethical Hacking Linux Distribution: `https://www.kali.org`

[8] OpenSSL Cryptography and SSL/TLS Toolkit: `https://www.openssl.org`

4.4 Day 2: Decrypting install cards

With new computers, we started the day by installing Kali Linux in all machines, a very time-consuming process which took the whole day. In parallel, we continued to explore the testing environment. We also received a somewhat distracting visit of international observers invited by the SEC to follow the challenge.

Our first attempt was to plug a regular keyboard from the regular computers in one of the USB ports of the voting machine. With this, we could observe that the keys pressed were echoed in the screen during the boot process, and therefore, we concluded that the machine had hardware and driver components that enabled the usage of regular USB keyboards. However, using sequence keys like `<Ctrl> + <Alt> + <F1>` and others, we were not able to drop in a terminal. We then started following our testing plans.

For the first one, named "*Extraction of cryptographic keys*", the idea was to try to decrypt the real install cards with the key_1 value found during the code inspection stage. If that went wrong, we would try to get the correct key_1 value through reverse engineering of the bootloader and kernel image. We generated an install card using the provided GEDAI machine and reimplemented the AES-XTS decryption program, memorizing and typing down the decryption key a few bytes at a time. We ended up rewriting the whole program by adapting code from another AES-XTS implementation [25] to increase performance.

In our first decryption attempt, we 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 them looking for additional vulnerabilities to escalate access.

4.5 Day 3: Executing our code

During the third day, we better analyzed the chain of trust established by the DRE voting machine. Basically, subsystems verify signatures of subsystems running next, as shown in Figure 3. The process starts with the verification of the BIOS signature by the MSD and ends with the validation of single files using a detached signature file with extension VST.

VST files contain file paths along with their signatures in a custom binary format, described in ASN.1 syntax. During the installation and initialization of the voting machine, the files with entries in the VST are checked and the signatures validated. We observed that extraneous files could be added to the install card without triggering any security alert.

By inspecting the VST files we noticed that two shared libraries used by the voting system did not have corresponding detached signatures. These were used for logging (`libapilog.so`) and HMAC-based Extract-and-Expand Key Derivation Functions (HKDF) (`libhkdf.so`). 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. This possibility of arbitrary code execution was the main attack vector used throughout the rest of the challenge.

During the rest of the day, other tests were made so the SEC staff could validate and register our findings. We observed which infected function was the first one to be called, and in one of the attacks, a custom text was printed in the terminal presented during the system initialization. Using the `write` syscall outputting to `stdout`, we printed the text "`FRAUD!`" in the terminal. Extending the attack, a simple read-echo loop was created in order to show the possibility of using a regular USB keyboard, writing into the `stdout` messages read from `stdin`.

At the end of the second day, our team decided to abandon the other testing plans to focus on the recently acquired capabilities obtained after the decryption of install cards. Because the first testing plan was already considered successful, we submitted two more, called *Violation of ballot secrecy for selected votes* and *Arbitrary code execution in the DRE voting machine*, which were an escalation from the success in the first one.

4.6 Day 4: Manipulating logging, violating ballot secrecy for selected voters

After exploring the possibility of running arbitrary code and using a regular USB keyboard in the DRE voting machine, the next attack was to include a full-featured shell like BusyBox[9] in order to make it easier to debug the running system and prototype new attacks. The first attempt was adding the binary into the install card, which allowed storing non-signed files, and executing it through the `execve` syscall. This attack was unsuccessful because, before executing a binary, the kernel checks for a signature appended in the end of the file. Since the BusyBox binary was not signed, the kernel disallowed the execution. The second attempt was to embed the shell inside one of the non-signed shared libraries. This was a promising idea, but since our attack vector was already relocated in memory, we needed to either implement an

[9] BusyBox: `https://busybox.net`

ELF loader or modify the whole binary to be position-independent code. Because of the time constraints and limited tooling, we did not pursue this idea further.

To illustrate other possibilities of exploiting our code injection, we performed more attacks to the DRE machine. 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 simulating an entire election and observing the modifications in the corresponding file stored in the MR.

In another attack, we infected the HKDF library to force the derived cryptographic keys to be 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. Since 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 only the `count` argument is 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 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 figures of authority), 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 the differences in chronological order inside the voting card.

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[10] and could not be unpacked by the standard UPX command line tool. Thus, we decided to bring the UPX source code on the next day to facilitate debugging of the unpacking issues.

[10]UPX: the Ultimate Packer for eXecutables – `https://upx.github.io`

4.7 Day 5: Tampering with screen contents and votes

In the morning, we received a formal visit of the president of the SEC and were asked to give a live demonstration to the overseeing committee of progress so far. The increased media coverage was distracting and greatly impaired our team's ability to concentrate on the work. Despite that, we proceeded to debug the binary unpacking issue found in the previous day, and 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 work for exploitation because targeted contents were 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 exists both in SCUE and VOTA but contains 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. The original string was "The Hour of the Star", but we modified 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).

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 DRE screen after this attack is shown in Figure 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 proceeded to improve the HDKF library infection technique. Until this moment, we simply overwrote the `hkdf()` library function which actu-

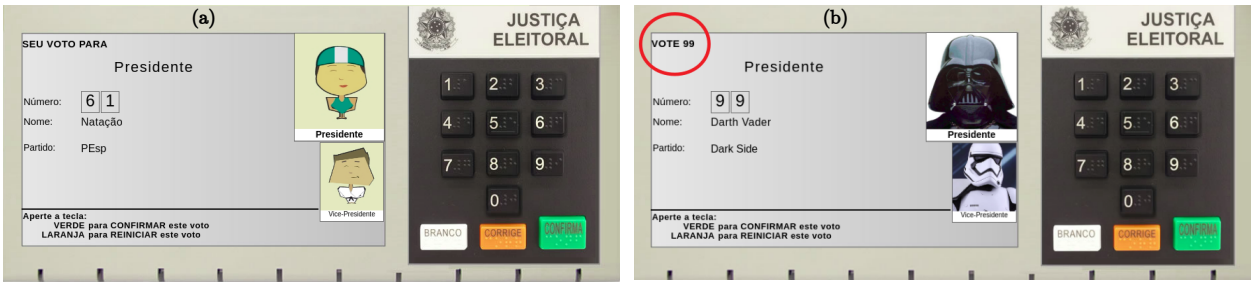

Figure 6: Reproduction of the in-memory modification of one of the strings contained in the VOTA application. **(a)** Original DRE. **(b)** Compromised DRE – the voting software now advertises the choice of a hypothetical candidate.

ally computes a key. Because of this, our payloads always caused the DRV key to be zeroed, similar to the attack against ballot secrecy described in Section 4.6. Although an apparently corrupt DRV file would hardly raise any suspicions, since the DRV file is only inspected when an audit is performed, we implemented a method to avoid this drawback by preserving HKDF functionality.

As illustrated in Figure 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, which 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.

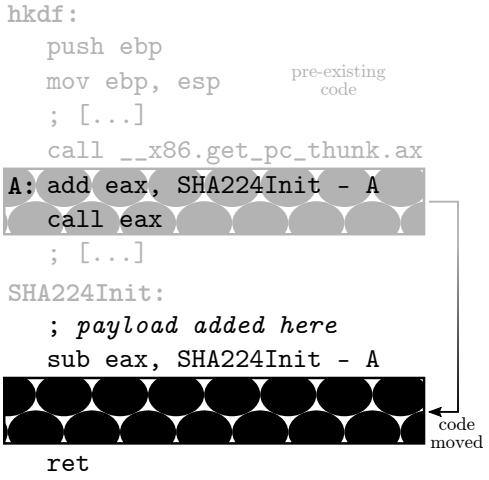

Figure 7: A simple library code infection technique we employed to furtively compromise `libhkdf.so`.

Continuing to analyze the VOTA binary with a disassembler, 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 presented in Listing 2. 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 memory addresses (starting at `edi`).

Listing 2 Code injected in the `AddVote` method to manipulate votes in favor of candidate number "99".

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

Nevertheless, when we tested that payload in the DRE, it caused a segmentation fault just before the VOTA application was expected to appear in screen. Since that meant the `AddVote()` method was never actually called, we quickly found the mistake. To overwrite `AddVote()`, our Python script was issuing `lodsb/w/d` instructions instead of `stosb/w/d`. As `esi` contained zero (we correctly loaded the address to `edi`), this caused a null pointer dereference.

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 no more typos were present 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 the offices, however just after pressing the confirm button for the last office, a consistency check triggered an alert message in the DRE screen stating that the 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.

5 Discussion

DRE voting machines are largely criticized in academic literature, mainly due to its 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 [9]. 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 [18], and hardware attacks against the Indian EVMs discussed by Halderman *et al.* [26]. 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 academics [1].

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 [19]. The report suggests many improvements to the election workflow, but no detailed 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 had the clearance to examine the voting software 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 rich literature about the security features of Brazilian DRE machines.

The report published by Aranha *et al.* after the 2012 edition of the TPS [2] 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 point out the insecure storage of cryptographic keys and inherent limitations of the software integrity checking mechanism. Our work should help filling this gap and to accurately update the state of Brazilian voting technology to the international technical community. We split the discussion in the software integrity and ballot secrecy properties and finish arguing how our results invalidate official security claims published by the SEC.

5.1 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, Group 4, 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.

We thus conclude that the integrity of software, and consequently the results, depends ultimately on the secrecy of a symmetric key embedded into the bootloader. This key is trivially accessible by the whole voting software development team, and is 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.* [2] 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.

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 bug in the verification code (and its unit test) prevented the manipulation to be detected [16]. This suggests a development process in need of urgent revision of its critical procedures.

The staff also states that cryptographic keys for encrypting the file system will not be hard-coded anymore in future versions of the software, but computed on-the-fly with help of the BIOS [16], increasing the level of obfuscation. This design choice was justified by the SEC with observing that not all voting machines have the MSD device available, which limits the possibility of using its idle space for storing cryptographic keys. Because of the imposed requirements that any voting machine must be able to replace any other voting machine on election day, it is considered risky to have different versions of the software in operation. This suggests further that overall security of the system is dictated by the oldest model in operation, in this case the 2007 one without 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 one of these keys leak. In the longer term, a key distribution architecture should be implemented and all cryptographic keys should be moved inside the MSD security perimeter.

5.2 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 failure of implementing paper records in 2002. As claimed by the SEC, it supposedly permits independent verification of election results [12], 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.* [2] concludes that the DRV file does not serve any security purpose besides violating ballot secrecy if designed or implemented insecurely. For this reason, preventing attacks against the DRV relies on the implementation of a secure random number generation algorithm.

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 [21] 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 [20]. 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 [11].

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 looks 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 [2]. 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 and

[11] Sapparot-2: `http://www.literatecode.com/sapparot2`

again recommend the DRV file to be eliminated, and the law to be changed.

5.3 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 becomes a moving target, conveniently changing depending on the attack under discussion. Fortunately, the SEC published a Q&A document defending some of the security mechanisms [12], in response to the results obtained by Aranha *et al.* [2]. This document is a very useful resource 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 in this paper 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.*

Our attacks were able to include additional files and modify two shared libraries in the install card, with the software installation process being completed without any hassle. The software installed in the machines preserved the intended 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 voting 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 up to 50 voting machines, this approach has an amplification effect, 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 performance metrics, such as the false positive rate reported for the fingerprint identification system. This is naturally true of any electronic record, and explains why purely electronic voting systems are inherently insecure and opaque.

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, a possible 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 important to note that the file system encryption is not the sole mechanism supporting software security 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 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 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 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.

Availability

The code written during the hacking challenge and additional files can be found in the repository available at `https://github.com/epicleet/tps2017`

References

[1] AMATO, F., ORO, I. A. B., CHAPARRO, E., LERNER, S. D., ORTEGA, A., RIZZO, J., RUSS, F., SMALDONE, J., AND WAISMAN, N. Vot.Ar: una mala elección. `https://ivan.barreraoro.com.ar/vot-ar-una-mala-eleccion/`, 2015.

[2] ARANHA, D. F., KARAM, M. M., MIRANDA, A., AND SCAREL, F. *Software vulnerabilities in the Brazilian voting machine*. IGI Global, 2014, pp. 149–175.

[3] ARANHA, D. F., RIBEIRO, H., AND PARAENSE, A. L. O. Crowdsourced integrity verification of election results - An experience from Brazilian elections. *Annales des Télécommunications 71*, 7-8 (2016), 287–297.

[4] AUMASSON, J., AND BERNSTEIN, D. J. Siphash: A fast short-input PRF. In *INDOCRYPT* (2012), vol. 7668 of *Lecture Notes in Computer Science*, Springer, pp. 489–508.

[5] BARRETO, P. S. L. M., AND NAEHRIG, M. Pairing-friendly elliptic curves of prime order. In *Proceedings of the 12th International Conference on Selected Areas in Cryptography* (Kingston, ON, Canada, Aug. 2005), SAC'05, pp. 319–331.

[6] BERNSTEIN, D. J., DUIF, N., LANGE, T., SCHWABE, P., AND YANG, B. High-speed high-security signatures. In *CHES* (2011), vol. 6917 of *Lecture Notes in Computer Science*, Springer, pp. 124–142.

[7] BONEH, D., AND BOYEN, X. Short Signatures Without Random Oracles and the SDH Assumption in Bilinear Groups. *Journal of Cryptology 21*, 2 (Feb. 2008), 149–177.

[8] BRAZILIAN SOCIAL DEMOCRACY PARTY – PSDB. Report on the Special Audit in the 2014 Voting System. Available at `http://www.brunazo.eng.br/voto-e/arquivos/RelatorioAuditoriaEleicao2014-PSDB.pdf`, 2014.

[9] CALANDRINO, J. A., FELDMAN, A. J., J. A. HALDERMAN, D. W., AND H. YU, W. P. Z. Source Code Review of the Diebold Voting System. Available at `https://jhalderm.com/pub/papers/diebold-ttbr07.pdf`, 2007.

[10] COMMITTEE, T. O. Report of the Overseeing Committee of the Public Security Tests, 2017 edition (In Portuguese). `http://www.tse.jus.br/hotsites/teste-publico-seguranca-2017/arquivos/tps2017-relatorio-comissao-avaliadora.pdf`, 2017.

[11] COURT, S. E. 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.

[12] COURT, S. E. Frequently Asked Questions (FAQ) about the Brazilian voting system, 2nd edition (In Portuguese). `http://www.justicaeleitoral.jus.br/arquivos/tse-perguntas-mais-frequentes-sistema-eletronico-de-votacao`, 2014.

[13] COURT, S. E. 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.

[14] COURT, S. E. Call for participation in the Public Security Tests. `http://www.tse.jus.br/hotsites/teste-publico-seguranca-2017/arquivos/TPS-testes-publicos-seguranca-edital.pdf`, 2017.

[15] COURT, S. E. Draft of resolution concerning the electoral procedures for implementing a paper trail in the 2018. `http://www.justicaeleitoral.jus.br/arquivos/tse-audiencias-publicas-voto-impresso`, 2017.

[16] COURT, S. E. Vulnerabilities and suggestions for improvement to the voting machine ecosystem observed in the Public Security Tests, 2017 edition (In Portuguese). `http://www.tse.jus.br/hotsites/teste-publico-seguranca-2017/arquivos/tps2017-relatorio-tecnico.pdf`, 2017.

[17] DATABASE, N. V. JBoss Vulnerabilities. Available at `https://nvd.nist.gov/products/cpe/search/results?keyword=jboss&status=FINAL&orderBy=CPEURI&namingFormat=2.3`, 2018.

[18] GONGGRIJP, R., AND HENGEVELD, W. Studying the nedap/groenendaal ES3B voting computer: A computer security perspective. In *EVT* (2007), USENIX Association.

[19] J. VAN DE GRAAF, R. F. C. 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.

[20] LEVIN, I. O. Sapparot-2: Fast Pseudo-Random Number Generator. `http://www.literatecode.com/get/sapparot2.pdf`, 2005.

[21] MARSAGLIA, G. Xorshift RNGs. *Journal of Statistical Software, Articles 8*, 14 (2003), 1–6.

[22] STALLINGS, W. *Cryptography and Network Security: Principles and Practice 7th edition*. Pearson, 2016.

[23] STEINMETZ, F. USB – an attack surface of emerging importance. Bachelor's thesis, Technische Universität Hamburg, 2015. Available at `https://tubdok.tub.tuhh.de/handle/11420/1286`.

[24] SUPERIOR ELECTORAL COURT OF BRAZIL. 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.

[25] TEUWEN, P. python-cryptoplus, AES-XTS python implementation. Available at `https://github.com/doegox/python-cryptoplus`, 2017.

[26] WOLCHOK, S., WUSTROW, E., HALDERMAN, J. A., PRASAD, H. K., KANKIPATI, A., SAKHAMURI, S. K., YAGATI, V., AND GONGGRIJP, R. Security analysis of India's electronic voting machines. In *ACM Conference on Computer and Communications Security* (2010), ACM, pp. 1–14.