Overview of the Application-Level Security of the Swiss E-voting System

Author: Mazin Ahmed <mazin@mazinahmed.net>

April 2019

1. Introduction

This year, 2019, the Swiss government is planning to fully establish E-voting within the next elections. During 25 February to 24 March 2019, the Swiss Post has hosted a public program where researchers, hackers, and professionals can register for a program where they can attack the Swiss E-voting system in a controlled environment. Any individual who identifies a security vulnerability or a potential attack that affects the system can report it to the Swiss Post program, where the Swiss Post will analyze and fix the finding as well as financially rewarding the reporter. The Swiss Post has also released the source code of the system as a separate program, but rewards were only paid if the individual were only able to exploit the weakness on the public program.

I wanted to write this research to show an overview of the Swiss Post E-voting system security from an application level. I am a security consultant with years of experience in auditing applications security. You can read more about my work at mazinahmed.net.

The conducted analysis mainly focuses on the security overview of the application. It does not cover cryptographic implementations within the system.

2. Findings

2.1 CSP Misconfigurations

Content Security Policy (CSP) is a standard for protecting against XSS vulnerabilities on the browser level. Reviewing the Content Security Policy used by The Swiss E-voting system, I found a bypass for the policy:

The CSP for Swiss E-voting System (as of February 26th, 2019):

```
Content-Security-Policy: default-src 'none'; script-src 'self' 'unsafe-eval'; style-src 'self' 'unsafe-inline'; img-src 'self'; connect-src 'self'; child-src 'self'; font-src 'self'
```

A well-written CSP should prevent the exploitation of XSS attacks. This above policy allows `unsafe-inline` and `unsafe-eval`. This opens a door for typical JavaScript vectors to be executed. For example,

```
<script>alert("inline JS");</script>
```

Can execute normally as inline JavaScript is permitted by the policy.

It's a clear finding that can be marked as a "bypass of an implemented protection layer". However, I can't say it's a severe issue, as this needs to be combined with an XSS (Cross-Site Scripting) vulnerability in order to be effective. Without the presence of an XSS vulnerability, I can not make use of this issue or exploit the CSP misconfiguration.

2.2 Authentication Issue

There is a severe issue in the authentication flow that the Swiss E-voting system is using to authenticate voters. Although the exploitation of this issue is relatively impossible to go with detection, it's a scenario that can pose a threat to the authentication system.

Before demonstrating the attack, I'm going to briefly discuss the authentication flow.

1. The voter receives the voting card by mail.
2. The voter enters their identification code. This code is uniquely assigned for each voter.
3. An authentication token is generated according to the voter identification code. This token is returned in the response, and used in future steps in the voting process.
4. The encryption/decryption keys are then requested using the authentication token.

The steps go from this point until voting completes for the voter.

From the above demonstration, we can see that the authentication token generated per each voter is the key to compromise any vote. If an attacker can obtain an authentication token, they can vote on voters behalf. This clearly affects the system directly.

The structure of the authentication token is the following:

```
[0-9A-F]{32}
```

Which is the same pattern as MD5 hash scheme.

In all cases, attempting to try all possibilities the authentication tokens requires 3.4 ^ 38 requests. There is no way that this is done or tested correctly without being noticed. The possibility of having a correct brute-forcing attack is quite low, and the execution is difficult and requires a lot of resources. An additional note is that the web-server does not support HTTP/2 or HTTP Pipelining, so reusing the same TCP connection in order to do multiple HTTP requests is not possible.

After reporting the attack to the Swiss Post, I found that an attacker would require to obtain a verification code that is 8-digits numeric code. Although this attack was already almost impossible to achieve, it's much more difficult to achieve with the verification (finalisation) code check.

2.3 Using components with known vulnerabilities

When the source-code was released, I analyzed the components and the dependencies of the Swiss E-voting system. There are a number of disclosed security vulnerabilities that are known by NVD. I'm highlighting the CVE IDs below:

- CVE-2018-7489
- CVE-2018-3745
- CVE-2018-3737
- CVE-2018-3728
- CVE-2018-3721
- CVE-2018-19362
- CVE-2018-19361
- CVE-2018-19360
- CVE-2018-16472

- CVE-2018-14721
- CVE-2018-14720
- CVE-2018-14719
- CVE-2018-14718
- CVE-2018-1275
- CVE-2018-1272
- CVE-2018-1271
- CVE-2018-1270
- CVE-2018-1257
- CVE-2018-12022
- CVE-2018-1199
- CVE-2018-11040
- CVE-2018-11039
- CVE-2018-1000850
- CVE-2018-1000844
- CVE-2018-1000643
- CVE-2018-1000620
- CVE-2017-7525
- CVE-2017-18214
- CVE-2017-17485
- CVE-2017-17461
- CVE-2017-16138
- CVE-2017-16137
- CVE-2017-16114
- CVE-2017-16113
- CVE-2017-16028
- CVE-2017-15095
- CVE-2017-15010
- CVE-2017-1000427

If an application is using a vulnerable dependency or component, that doesn't 100% confirms that the application is vulnerable. I haven't explored all the vulnerabilities due to a shortage of time. A fix is essentially simple: update to the latest patches while ensuring the newest patches do not break the functionalities of the system.

2.4 Rate-Limiting Issues

The authentication process on the checking the is heavily protected using rate-limiting and similar mechanisms during the initial authentication step. The aggressive rate-limiting measures create a possibility of mass-locking voters from elections if an unauthorized party is capable of identifying the voter's identification code. However, this does not seem to be an issue since the identification code length is relatively long, and a similar attack can be easily seen and identified.

3. Public Intrusion Test Findings

After the PIT is done, the Swiss Post has publicly disclosed the valid findings reported in the PIT. The following issues were identified:

- [BEST PRACTICES] Crafted X-Forwarded-For HTTP header injection
- [BEST PRACTICES] Missing HTTP to HTTPS redirection on 'pit-admin.evoting-test.ch'
- [BEST PRACTICES] Outdated version of Bootstrap Web Framework
- [BEST PRACTICES] Vulnerable TLS cipher-suites (LUCKY13)
- [BEST PRACTICES] Missing 'Expect-CT' HTTP header
- [BEST PRACTICES] Missing 'base-uri' in Content Security Policy
- [BEST PRACTICES] Incorrect 'HTTP-Strict-Transport-Security' header on 'pit-admin.evoting-test.ch'
- [BEST PRACTICES] Use of 'unsafe-eval' and 'unsafe-inline' in Content Security Policy
- [BEST PRACTICES] Multiple occurrences of 'X-XSS-Protection' HTTP header
- [BEST PRACTICES] Use of outdated version of AngularJS
- [BEST PRACTICES] Strict Transport Security Mis-configuration
- [BEST PRACTICES] Use of cipher suites without forward secrecy support
- [BEST PRACTICES] Missing charset declaration in some response's Content-Type header
- [BEST PRACTICES] Missing CSP header in redirect responses
- [BEST PRACTICES] Cross Origin Request possible on specific endpoint
- [BEST PRACTICES] Missing CSP header on http://pit-admin.evoting-test.ch/

The details of the PIT findings are published online at:
https://www.onlinevote-pit.ch/stats/

I wanted to note that every single finding mentioned here poses a low impact and the risk is relatively low. Consultants do not consider the majority of these issues an actual threat to the application unless it's combined with another vulnerability that can cause damage. In fact, I encountered the vast majority of these 16 findings on the Swiss E-Voting site during the research.

3. Conclusion

I was able to identify security vulnerabilities on the Swiss E-voting system, but I was not able to break the system.

I haven't reported the findings to the PIT because of the reasons mentioned in this report, but I have shared the report with the Swiss Post privately.

It's also worth-noting to state that the Swiss E-voting system that was included on the Public Intrusion Test covers the voter's interface. The backend/admin panel for the voting system was not covered in the Public Intrusion Test. It was formally in the scope but were not accessible for researchers. The goal of this part is to simulate an actual election system in real-world, where only authorized parties are capable of accessing the administration panel. Attempting to access the administration panel was in the scope.

Several bugs were also discovered in the source code of the Swiss E-voting system by third-party researchers. However, researchers were not able to remotely exploit the findings and attacks on the live Swiss E-voting system during the Public Intrusion Test due to the extensive protection measures done by the Swiss Post side.

The Swiss Post did a great work on mitigating a number of attacks on the E-voting system. The server configuration, the infrastructure design, and the web-application-firewall setup made the attacks much more difficult to conduct. The results, and the fact of not having proven a full remote attack against the system reflects the maturity level of the web application security.

I’m very glad to have the opportunity to test the Swiss E-voting system. It’s a big move that I’m hoping it to become a reality this year. The disclosed vulnerabilities can be fixed. I have also been informed that a number of the vulnerabilities were already patched before publishing the report.

Mazin Ahmed
Email: Mazin@MazinAhmed.net
Website: MazinAhmed.net
CV: MazinAhmed.net/cv
LinkedIn: https://Linkedin.com/in/infosecmazinahmed