



École Polytechnique Fédérale de Lausanne

D-voting - Security audit

Master Semester Project Report

Document owner Chen Chang Lew (sciper: 321016)
Supervisor Noémien Kocher, Pierluca Borsò-Tan, Simone
Responsible Prof. Bryan Ford

Table of content

| | |
|-------------------------------|-----------|
| Introduction | 2 |
| Motivation | 2 |
| Background | 2 |
| External Dependencies | 3 |
| Analysis Steps | 3 |
| Result | 8 |
| Verifiability Proposal | 8 |
| Future work | 9 |
| Conclusion | 9 |
| Reflection | 10 |

Introduction

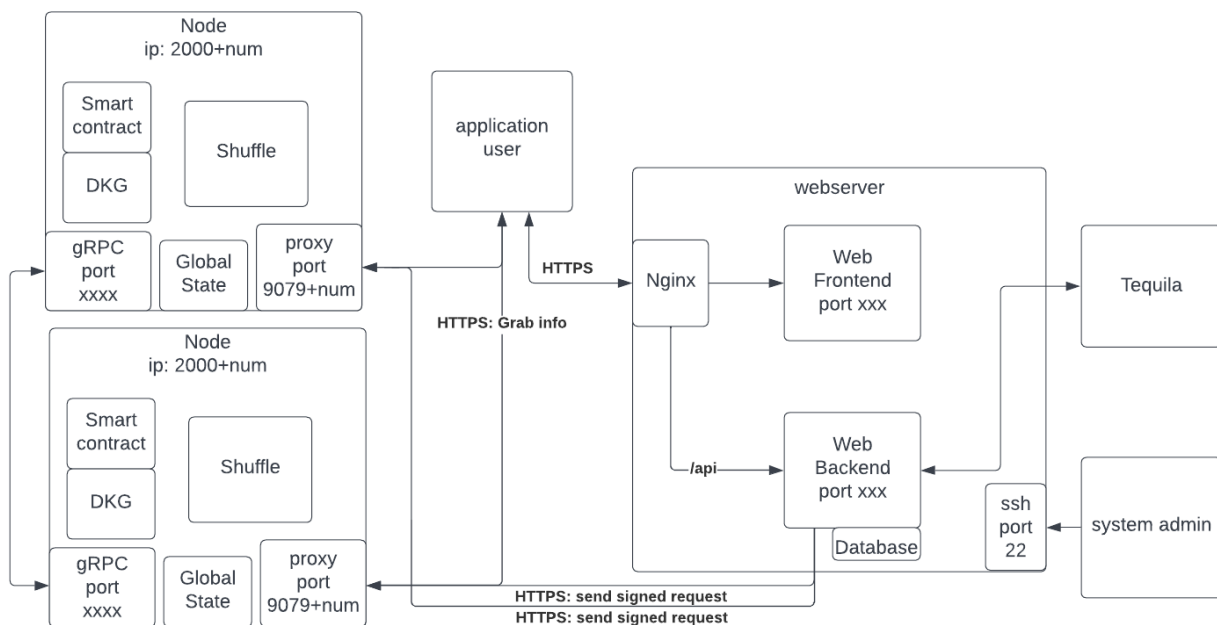
D-Voting is an e-voting platform based on the [Dela](#) blockchain. It uses state-of-the-art protocols that guarantee the privacy of votes and a fully decentralized process. This project was born in early 2021 and has been iteratively implemented by EPFL students under the supervision of DEDIS members.

Motivation

D-voting project aims to launch and provide service to EPFL users in 2023. My main job is to conduct a security analysis for this open-source project. Conducting a security analysis in this project is important for several reasons. First and foremost, it helps protect the users of the project by identifying and addressing potential vulnerabilities that could be exploited by malicious actors. It also helps protect the reputation and integrity of the project by demonstrating a commitment to security and responsible development practices. Additionally, it can improve the overall quality and stability of the project by identifying and addressing potential issues before they become widespread. Overall, conducting a security analysis on this project is a critical step in ensuring that the project is safe, reliable, and trustworthy for all users before we launch this service.

Background

D-voting basic architecture is shown as follows:



The d-voting system itself is divided (roughly) into two “planes”, or groupings of components, the following table describes each plane, and groups the aforementioned components.

| Plane | Description | Components |
|------------------|---|--|
| Web Plane | Web Plane is focused on the end-user journey. It controls the state of the election and the user authority. | Web Frontend, Web Backend, Database |
| Blockchain Plane | The blockchain plane guarantees the privacy of voters and a fully decentralized process to store the encrypted vote and the transactions. The nodes are public thus anyone can view and verify the states on the nodes. | Proxy, Blockchain node, Smart Contract, DPKG |

External Dependencies

In software development, an external dependency is a library or module that is required for a piece of code to run, but which is not part of the code itself. External dependencies are typically included in a project through a package manager or by downloading the required files from a remote repository.

| Dependencies | Usage | Dependencies version |
|-----------------|--|---|
| Dela blockchain | is the structure of the d-voting blockchain node. | d977167 |
| Crypto library | The crypto library utility from Dela | v0.0.0-20221214133440-d977167551e6 |
| golang | Use to write d-voting smart contract main logic | 1.19 + |
| node | source server environment | V16.17.1 + |
| typescript | Use to write d-voting web app main logic (both frontend and backend) | V4.5.5 + |
| TLS | Communications transfer over TLS | Organization: Let's Encrypt TLS 1.3, X25519, AES_256_GCM |

Analysis Steps

1. Investigate the user flow of the system.


I created several user flow graphs to help visualize the steps that users would take to perform several actions. I followed a process of defining the task or goal, identifying the necessary steps, creating a visual representation using a flowchart tool, and reviewing and revising the graph to ensure accuracy and effectiveness. These user flow graphs were a valuable resource

for understanding how users navigate and interact with our product, and they helped inform the design and development process. All the user flow graphs are in the appendix pages on the [security report page](#)

2. Apply SNYK to the d-voting (check for third-party libraries).

Snyk is a software tool that helps developers find and fix known vulnerabilities in the open-source libraries and dependencies they use. Snyk can be used to scan projects for vulnerabilities, and it can also be integrated into the development process to help prevent the inclusion of vulnerable dependencies in the first place. Snyk is designed to work with a variety of programming languages and package managers, including npm, Maven, and pip.

I apply SNYK to the d-voting repository. It sends weekly reports about the known threats of third-party library dependencies, and it can even create PR automatically for us.

 **snyk**

nkcr's weekly report

28th of December – 4th of January 2023

Status of all 4 active projects

| | |
|------------------------------------|-----------------------------------|
| 26 known vulnerabilities | 1911 total dependencies |
| 0 C 12 H 14 M 0 L | |

Review the status of your projects on your dashboard. [View on Snyk](#)

[Snyk] Security upgrade golang from 1.17.7-alpine to 1.18.7-alpine #207

Merged Flamewind97 merged 1 commit into main from snyk-fix-61d8742ad190ca3d83e9a5aeab446ad1 3 weeks ago

Conversation 5 Commits 1 Checks 9 Files changed 1

snyk-bot commented on Nov 9, 2022

Keeping your Docker base image up-to-date means you'll benefit from security fixes in the latest version of your chosen image.

Changes included in this PR

- Dockerfile

We recommend upgrading to `golang:1.18.7-alpine`, as this image has only 0 known vulnerabilities. To do this, merge this pull request, then verify your application still works as expected.

Some of the most important vulnerabilities in your base image include:

Reviewers

- Flamewind97
- nkr
- pierluca

Assignees

No one—assign you

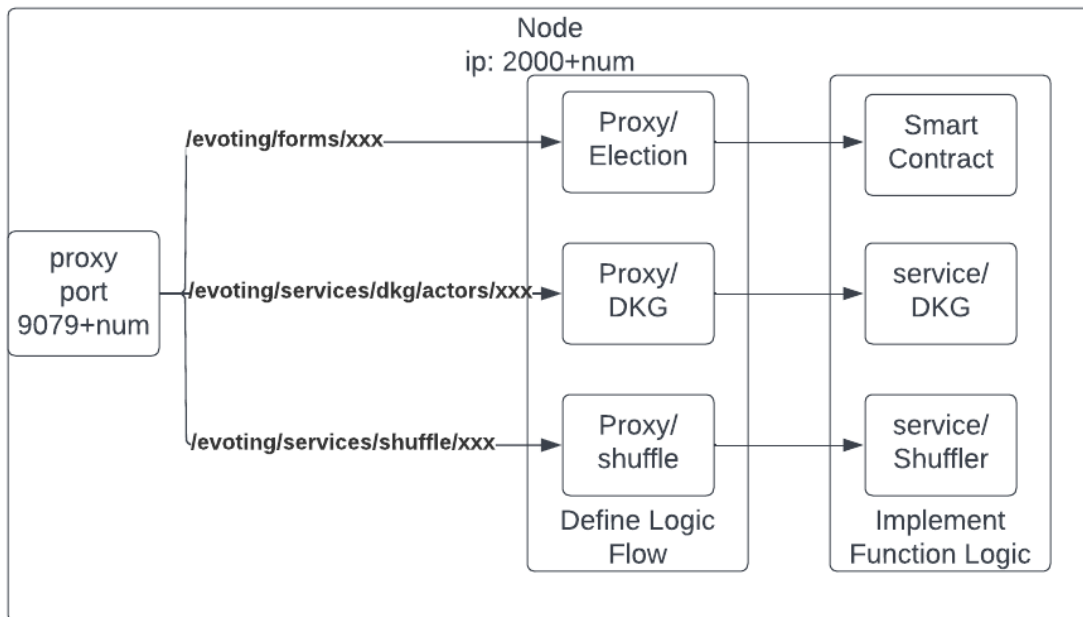
Labels

3. Add “[security.txt](#)” to the [d-voting codebase](#).

“security.txt” is a file that can be placed in the root directory of a website to provide a way for security researchers to report security vulnerabilities they discover to the website's owner. The file should contain contact information, such as an email address or other means of communication, that researchers can use to report vulnerabilities. The purpose of security.txt is to provide a standard way for researchers to report vulnerabilities in a responsible manner, rather than publicly disclosing them or exploiting them for malicious purposes. It is a way for website owners to encourage responsible disclosure of security issues and to make it easier for researchers to report vulnerabilities to them.

4. Inspect the [proxy's code](#).

After Defining all the user flow, I start looking at the proxy codebase. A proxy offers the mean for an external actor such as a website to interact with a blockchain node. It is a component of the blockchain node that exposes HTTP endpoints for external entities to send commands to the node. The proxy is notably used by web clients to use the voting system.



I am able to find some technical debts that can be fixed to provide better readability to the code.

- [Technical Debt - verify signature before execute request #210](#)
- [Technical Debt - unclear/wrong comment #213](#)
- [Technical Debt - variable name "buff, formIDBuf, formIDBuff, formID" not consistent #214](#)

5. Inspect the DKG's code.

The DKG service allows the creation of a distributed key pair among multiple participants. Data encrypted with the key pair can only be decrypted with the contribution of a threshold of participants. This makes it convenient to distribute trust in encrypted data. In the D-Voting project, we use the Pedersen version of DKG.

The DKG service needs to be set up at the beginning of each new election because we want each election to have its own key pair. Doing the setup requires two steps: 1) Initialization and 2) Setup. The initialization creates new RPC endpoints on each node, which they can use to communicate with each other. The second step, the setup, must be executed on one of the nodes. The setup step starts the DKG protocol and generates the key pair. Once done, the D-Voting smart contract can be called to open the election, which will retrieve the DKG public key and save it on the smart contract.

I am able to find some technical debts and Threats for DKG:

- [Technical Debt - check lenAddrs before sending getPeerKey #216](#)
- [THREAT - Denial of Service. Dkg public key will always return false if an adversary compromise one device. #217](#)
- [Technical Debt - Encrypt function in DKG && Marshall ballot function in smart contract is not used anymore #218](#)
- [Technical Debt - Duplicate function getForm\(\) #219](#)
- [Technical Debt - Need refactor in dkg & shuffler #220](#)

6. Inspect the shuffler's code.

The shuffling service ensures that encrypted votes can not be linked to the user who cast them. Once the service is set up, each node can perform what we call a "shuffling step". A shuffling step re-orders an array of elements such that the integrity of the elements is guaranteed (i.e no elements have been modified, added, or removed), but one can't trace how elements have been re-ordered.

I am able to find some technical debts and Threats for shuffler:

- [Technical Debt - unclear/wrong comment #213](#)
- [Technical Debt - Duplicate function getForm\(\) #219](#)
- [Technical Debt - Need refactor in dkg & shuffler #220](#)
- [Technical Debt - change loop and sleep to channel + ctx timeout to increase readability of code. #221](#)
- [Technical Debt - shouldn't use fingerprint function for pseudorandomness because it is not efficient. #244](#)

7. Inspect the smart contract's code.

In the D-Voting system, a single D-Voting smart contract handles the elections. The smart contract is written in Golang. The smart contract ensures that elections follow a correct workflow to guarantee desirable properties such as privacy. For example, the smart contract won't allow ballots to be decrypted if they haven't been previously shuffled by a threshold of nodes.

I am able to find some technical debts and threats for smart contract:

- [Technical Debt - change legacy structure "createForm" #212](#)
- [Technical Debt - variable name "buff, formIDBuf, formIDBuff, formID" not consistent #214](#)
- [Technical Debt - Encrypt function in DKG && Marshall ballot function in smart contract is not used anymore #218](#)
- [Technical Debt - shouldn't use fingerprint function for pseudorandomness because it is not efficient. #244](#)
- [THREAT - Frontend create form didn't check for the maximum length of the form #249](#)
- [THREAT - Election will not be able to reveal the result if anyone submits a fake vote. #250](#)
- [THREAT - ShuffleThreshold shouldn't be used as the nbrSubmissions threshold #251](#)

8. Inspect the web backend's code.

The web backend is built with Typescript and runs with NodeJs. The web backend handles authentication and authorization. Some requests that need specific authorization are relayed from the web frontend to the web backend. The web backend checks the requests and signs messages before relaying them to the blockchain node, which trusts the web backend. The web backend has a local database to store configuration data such as authorizations. Admins use the web frontend to perform updates. The blockchain node only trusts the backend as a source of data

I am able to find some technical debts and threats for web backend:

- [Technical Debt - Remove point & public key related in web backend since we didn't use it. #245](#)

- [THREAT - All the election stages can be ignored by malicious node #247](#)
- [THREAT - A user can vote multiple times \(count as multiple votes\) in an election. #248](#)
- [THREAT - A user who is not an admin or operator cannot vote. #253](#)

9. Inspect the web frontend's code and pages.

The web frontend is a web app built with React. It offers a view for end-users to use the D-Voting system. The app is meant to be used by voters and admins. Admins can perform administrative tasks such as creating an election, closing it, or revealing the results. Depending on the task, the web frontend will directly send HTTP requests to the proxy of a blockchain node, or to the web backend. The end user needs a web-frontend to perform every action related to the voting services

I am able to find some technical debts and threats for the web frontend:

- [THREAT - The public/private key of the election is changed by the Adversary #246](#)
- [THREAT - Frontend create form didn't check for the maximum length of the form #249](#)
- [THREAT - Logout will not clear all the sessions in the browser #252](#)

Result

I have created a [security report](#) that includes a detailed threat model and identifies potential security risks, as well as strategies for mitigating them. To address these risks, I have taken necessary steps and will continue to monitor the system to ensure its ongoing protection.

During my audit, I identified a total of 9 security issues and 11 technical debt issues. To track the resolution of these issues and ensure that they are properly addressed, I have created corresponding [issues in GitHub](#).

Verifiability Proposal

Upon initial review, I assumed that our d-voting system had a mechanism in place for users to verify their cast ballots. However, upon further analysis, I discovered that this feature had not yet been implemented. As a result, I have added [documentation](#) to GitHub outlining our plan for adding verifiability to the system and detailing the steps required to do so.

| step | Component involves | Descriptions |
|------|--------------------|--|
| 1 | Web Frontend | Edit the submit vote function - hash the encrypted ballot and shows it to the user. |
| 2 | Blockchain Node | Add users' hash of encrypted ballot to each election |

| | | |
|---|--------------|--|
| | | <ul style="list-style-type: none"> - edit API <code>"/evoting/forms/{formID}"</code>, add the hash of the ballot to the form structure.. |
| 3 | Web Frontend | <p>Edit the form details page to show the hash of the ballot.</p> <ul style="list-style-type: none"> - A user can select an election to see its details. - In the detail page, it shows the voter and the hash of their ballot. - Users can check if the hash they received is the same as the hash on the details. |

Future work

The following items represent potential areas for further security analysis and improvement:

1. Conduct a review on Dela and Kyber crypto package to identify any additional security risks because we assume them to be safe in the security report.
2. Review and add more security measure tools for d-voting to ensure that it meets current security best practices.
3. Conduct regular security assessments to identify and address any new or emerging security threats.

Conclusion

My main job for conducting security analysis for d-voting can be summarize follows:

1. Apply security scanning tools SNYK to help us to identify third party libraries vulnerabilities and how to mitigate them.
2. Add securitytxt in codebase to provide a standard way for researchers to report vulnerabilities in a responsible manner, rather than publicly disclosing them or exploiting them for malicious purposes.
3. Inspect the whole codebase and identified a total of 9 security issues and 11 technical debt issues. Some of these risks have been mitigated by our colleagues, while the remaining risks have been documented and will be addressed in future updates.
 - a. Security Issues:
 - [T1: The public/private key of the election is changed by the Adversary](#)
 - [T2: All the election stages can be ignored by malicious node](#)
 - [T3: A user can vote multiple times \(count as multiple votes\) in an election.](#)
 - [T4: DKG public key will always failed if an adversary compromise one device](#)
 - [T5: Frontend create form didn't check for the maximum length of the form](#)
 - [T6: Election will not be able to reveal the result if anyone submits a fake vote](#)
 - [T7: ShuffleThreshold shouldn't be used as the nbrSubmissions threshold](#)
 - [T8: Logout will not clear all the sessions in the browser](#)
 - [T9: A user who is not an admin or operator cannot vote.](#)
 - b. Technical Debts:
 - [TD1: verify signature before execute request \(proxy\)](#)

[TD2: change legacy structure "createForm" \(smart contract\)](#)

[TD3: unclear/wrong comment \(proxy, shuffling\)](#)

[TD4: variable name "buff, formIDBuf, formID" not consistent](#)

[TD5: check lenAddrs before sending getPeerKey \(dkg\)](#)

[TD6: Encrytp function in DKG && Marshall ballot function in smart contract is not used \(dkg, smart contract\)](#)

[TD7: Duplicate function getForm\(\) \(dkg, shuffling\)](#)

[TD8: Need refactor in dkg & shuffler \(dkg, shuffling\)](#)

[TD9: change loop and sleep to channel + ctx timeout to increase readability of code \(shuffling\)](#)

[TD10: shouldn't use the fingerprint function for pseudorandomness because it is not efficient. \(shuffling, smart contract\)](#)

[TD11: Remove point & public key in the backend since it is not used. \(web backend\)](#)

4. Adding vote verifiability design and planning.

[V1: show vote hash to user after casting a vote. \(web frontend\)](#)

[V2: add users' hash of the encrypted ballot for each election. \(smart contract, proxy\)](#)

[V3: show hash of encrypted vote in election \(web frontend\)](#)

Reflection

The semester project was a challenging but ultimately rewarding experience. I was honored to have the opportunity to review the entire codebase of d-voting and gain a deeper understanding of how each component works.

One of the most memorable aspects of the project was the discussion around the verifiability feature in d-voting and how we arrived at the chosen solution. I also encountered some challenges while working with the smart contract code, as I was not yet proficient in Dela. However, with the help of Pierluca, Simone and Noémien, I was able to overcome these challenges.

In the future, I hope to improve my technical documentation reading skills in order to more quickly grasp the concepts underlying the code. Overall, I am grateful for the opportunity to work on the semester project and feel that it has helped me grow both personally and professionally.