University of Zurich[UZH]

INDEPENDENT STUDY – Communication Systems Group, Prof. Dr. Burkhard Stiller

Security Analysis of the Blockchain Agnostic Framework Prototype

Alexander Hofmann
Zurich, Switzerland
Student ID: 11-916-863

Supervisor: Eder John Scheid, Bruno Bastos Rodrigues
Date of Submission: December 9, 2019

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Independent Study
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: https://www.csg.uzh.ch/

Kurzfassung

Die Communication Systems Group (CSG) an der Universität Zürich entwickelte eine Anwendung, um die Interoperabilität der Blockchain mit einer Low-Entry-Barriere für Benutzer und Entwickler ohne Blockchain-Expertise zu ermöglichen. Die Anwendung befindet sich derzeit in der Prototypenphase, und vor der Weiterentwicklung entschied sich die CSG-Gruppe für eine umfassende Sicherheitsanalyse der Anwendung. Diese Arbeit zielt darauf ab, die folgenden Schritte durchzuführen: *(i)* eine Umfrage über gängige Angriffsvektoren für Webanwendungen durchführen, *(ii)* eine Risikoanalyse der Plattform unter Berücksichtigung verschiedener Angreiferprofile durchführen, *(iii)* eine Überprüfung der Architektur durchführen, um etwaige Fehler im Design des Systems zu entdecken, und *(iv)* eine Sicherheitsanalyse des implementierten Systems durchführen. Für jedes Risiko oder jeden Sicherheitsfehler, der entdeckt wird, wird eine Minderungs- oder Gegenmassnahme vorgeschlagen, um die Anwendung zu härten. Basierend auf der Wissensbasis von Open Web Application Security Project (OWASP) stellt der Bericht die häufigsten Angriffsvektoren von Webanwendungen der letzten zehn Jahre vor, gefolgt von einer Reihe von Profilen möglicher Gegner und einer Auflistung aller Vermögenswerte des Blockchain for Cold-Chain (BC4CC)-Systems. Mit diesen Informationen deckt die Sicherheitsüberprüfung die Risiken für das Projekt auf und klassifiziert sie anhand von Wahrscheinlichkeit und Auswirkungen, bewertet das Gesamtrisiko und schlägt mögliche Gegenmassnahmen vor. Die grössten Risiken ergeben sich aus der Verwendung veralteter Abhängigkeiten, die oft veröffentlichte Schwachstellen und Anwendungsfehler mit sich bringen und eine teilweise oder vollständige Umgehung von Sicherheitskontrollen ermöglichen. Das Risiko eines Datenlecks der im System gespeicherten privaten Schlüssel, obwohl sie von grösster Bedeutung sind, kommt an zweiter Stelle, nur aufgrund der geringeren Wahrscheinlichkeit, dass dies geschieht. Für diesen Fall werden sowohl Designvorschläge als auch konkrete Massnahmen angeboten. Schliesslich werden die Architektur und die Anwendung überprüft, um Schwachstellen aufzudecken, von denen die kritischste es einem Benutzer erlaubt, nicht verifizierte Benutzerkonten in der BC4CC-Anwendung zu erstellen und dabei jede Form von Buchhaltung zu umgehen.

Abstract

The Communication Systems Group (CSG) at UZH developed an application to enable blockchain interoperability with a low-entry barrier for users and developers with no blockchain expertise. The application at the moment is in a prototype phase, and before continuing development, the CSG group decided to perform an all-encompassing security analysis of the application. This work aims to perform the following steps: *(i)* conduct a survey of common attack vectors affecting web applications, *(ii)* provide a risk analysis of the platform under consideration of various attacker profiles, *(iii)* conduct a review of the architecture to discover any flaws in the design of the system, and *(iv)* perform a security analysis of the implemented system. For each risk or security flaw discovered a mitigation or countermeasure is proposed to harden the application. Based on the Open Web Application Security Project (OWASP) knowledge-base, the report presents the most common web application attack vectors of the last decade, followed by a series of profiles of possible adversaries and a listing of all assets of the Blockchain for Cold-Chain (BC4CC) system. With this information, the security review exposes and classifies risks to the project based on likelihood and impact and assesses the overall risk while proposing possible mitigations. The highest risks found are related to the use of outdated dependencies, which often carry published vulnerabilities with them and application errors, permitting partial or total bypass of security controls. The risk of a leak of the private keys stored in the system, while being of highest impact, comes second due only to the lower likelihood of this happening. For this eventuality both design suggestions as well as concrete measures are offered. Finally, the architecture and application are reviewed exposing vulnerabilities, the most critical of which allows a user to create unverified user accounts in the BC4CC application, bypassing any form of accounting set in place.

Contents

Chapter 1

Introduction

The transport of medical drugs is usually done using Cold Chains, which is are temperature-controlled supply chains. To ensure regulatory compliance it is of utmost importance that it can be proven that this chain was never broken during transportation. In this sense, blockchains (BC) technology together with Internet-of-Things (IoT) sensors can be used to prove the integrity of the cold-chain [1]. However, these supply chains are often handled and overseen by multiple different stakeholders, each with their own requirements for compliance, which impacts the choice of the blockchain, resulting in different policies [2]. Thus, to keep costs down there is a need for an open standard that each stakeholder can refer to when exchanging information, which can be transparent to the concrete BC technology underneath.

To this end, the Blockchain Based Temperature Monitoring for Cold Chains in Medical Drug Distribution (BC4CC) project [3] set the goal of developing novel methods for supporting cold chain distribution process of medical drugs using blockchain. As part of this project the CSG group of the UZH developed a Policy-based Blockchain Agnostic Framework [2]. This framework is composed of the Bifröst application Programming Interface (API) [4], the Policy-based Blockchain Selection Framework, and a Flask-based [5] front-end and wrapper, which provides user access control and endpoints for the API. The framework is still in development, however given the sensitivity of the assets handled, the CSG wishes to do a security audit to ensure the security and safety of the platform.

The following requirements were set forth for the BC4CC project:

- Perform an in-depth analysis of existing blockchain technologies and develop a technically detailed portfolio. Respective requirements of those blockchains will be defined for major possible applications in the pharmaceutical supply chain as applicable to the BC4CC use cases to derive a generic scenario description.
- Create an open API that can be used by newly onboard customers without specific blockchain knowledge or understanding. The API requirements from potential partners will be collected and based on these, a selection of the most promising requirements will be chosen. These requirements need to align with the modum.io infrastructure.

- Provide a framework that is not dependent on a single BC (*e.g.,* Ethereum), but allow a selection between different blockchains. Blockchain agnosticism will also permit modum.io and their customers various commercial options to "cherry-pick" their optimal blockchain, and provide the potential technical capabilities for migration.
- Implement a BC4CC prototype putting into operation within a pilot, and evaluate, during and after its operation, aspects such as suitability of functionality, performance of transactions and interactions, and operational costs.

1.1 Description of Work

This Independent Study (IS) is related to the BC4CC project [3]. The prototype developed is currently deployed on a centralized server behind an Apache reverse proxy and implement basic security features, *e.g.,* requests limiters and basic HTTP authentication. However, these features alone do not provide the security of the entire framework. Thus, the goals of the independent study are:

- Perform an overview of the BC4CC project: needed to understand the main architectural components of the system
- Survey on Security Vulnerabilities and Attack Vectors which could affect BC4CC
- Security Analysis on the reverse proxy machine: the reverse proxy manages the forwarding of requests to the internal server which hosts the application and is the only part of the system exposed directly to the internet. As such its availability and integrity are most important.
- Security Analysis of the internal machine hosting BC4CC: this is where the sensitive information is stored and transactions are issued to the BCs.
- Documentation of possible attack scenarios, vectors, found vulnerabilities and mitigation techniques should be documented.

1.2 Report Outline

The remainder of this report is structured as follows. Chapter 2 presents an overview of the BC4CC project. Then, Chapter 3 describes a survey on security vulnerabilities and attack vectors, followed by a risk analysis and threat model in Chapter 4. Further, a security audit is conducted in Chapter 5. Finally, Chapter 6 summarizes the report.

Chapter 2

BC4CC Prototype Overview

This section provides a brief overview of the goal of the BC4CC system and the requirements in preparation for the risk analysis presented in Chapter 4 and the audit presented in Chapter 5.

2.1 System Goal and Requirements

The main goal of the prototype is to provide a simple interface to interact with different BCs. Its main user group are developers without or with little BC experience. Thus, the main requirement is to provide an API that allows a developer to transparently store data on various BCs without in depth knowledge of the details of the underlying technology.

2.2 Functional Requirements

This section aims to provide a clear overview of the capabilities of the system. It is important to assess which operations the system must be capable of performing under normal circumstances and which information is visible to which type of users, when performing a security audit.

- **Log in.** Users can authenticate using username and password.
- **Store data.** An authenticated user can see a form into which to enter the data to store and choose the BC. This data is then added to the selected blockchain without needing specialized knowledge of BCs.
- **Retrieve Data.** An authenticated user can retrieve data stored on a BC by giving the corresponding TX hash. The system can automatically determine which BC it refers to and retrieve the data.

- **Migrate Data.** An authenticated user can migrate data from BC X to BC Y by simply specifying the TX hash of the data on BC X and the target BC Y.
- **View transaction list.** An authenticated user can view all of their own transactions with their respective hashes, BCs, timestamps, validity (*"was the data migrated to another BC?"*), and a link to verify the transaction on the various BCs TX viewers. A user with the admin role is able to view and verify all transactions that took place on the system.
- **API.** A developer, has an API at disposable to integrate with other software. All API calls must be authenticated.
- **Access Token.** A developer can generate a token to authenticate API calls.

2.3 Non-Functional Requirements

Aside from the above mentioned functional requirements, the following non-functional requirements are considered:

- **Flexibility**: This requirement reflects the possibility to store arbitrary data in the BC of choice.
- **Modularity**: The system should be extensible by making it possible to add other BC to the system.
- **Ease of Use**: Developers should not need to know the details of how the various BCs work in order to be productive.

Chapter 3

Survey on Security Vulnerabilities and Attack Vectors

This section describes the most critical attack vectors for standard web application. In order to conduct the survey, the following sources were researched for possible attack vectors: OWASP Top 10 Web Application Security Risks [5], NIST Server Hardening guide [6] and Flask Security Guideline [7].

3.1 OWASP Top 10 Web Application Security Risks

The Open Web Application Security Project (OWASP) is an online community and non-profit organization founded in 2001 with the goal of producing freely-available content on the topic of web application security. Since its inception it has become the de-facto standard in the field, with other reputable entities, for example, the National Institute of Standards and Technology (NIST) [8] or PCI Security Standards Council [9] regularly referencing OWASP's work as an integral step to mitigating web application security risks.

The OWASP Top 10 focuses on identifying the most serious web application risks in broad terms, but each organization is unique. As such, it is important to develop a risk analysis to accurately determine the level of risk of a system. Such an analysis can be seen in Chapter 4. In general OWASP presents the most comprehensive survey on attack vectors, in particular when considering the previous versions as well.

The latest version of the Top 10 was released in 2017 and contains the following items [5]:

- **Injection.** An injection occurs when untrusted data is sent to an interpreter as part of a command or query. This data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
- **Broken Authentication.** Authentication mechanism are often very complex. Mistakes in the implementation or configuration, may expose the application to the risk of attackers compromising passwords, key, or session tokens, or impersonating other users.

- **Sensitive Data Exposure.** Insufficient controls may often result in attackers obtaining sensitive data, such as financial, healthcare, and Personally Identifiable Information (PII).

- **XML External Entities (XXE).** XML documents may refer to external entities such as file system locations, external services, etc. A poorly configured XML processor may attempt to evaluate these entities which can then be used to disclose sensitive information, to cause a Denial of Service (DoS), or perform a Remote Code Execution (RCE) attack.

- **Broken Access Control.** Closely related to *Broken Authentication*, implementation and configuration errors might make it possible for attackers to access unauthorized functionality or data. Examples include, viewing and/or changing another user's data and sensitive files, and change access rights.

- **Security Misconfiguration.** The most common risk, the result of insecure configurations might often end up exposing sensitive information, *e.g.,* exposing an unencrypted database service with a weak password. Further, software at all levels should regularly be patched and upgraded.

- **Cross-Site Scripting (XSS).** An XSS flaw is a type of injection attack in which untrusted code is injected in a web page without validation or escaping. This allows attackers to execute scripts in the victim's browser which can result in session hijacking, defacement, or redirecting users to malicious sites.

- **Insecure Deserialization.** In many programming languages and frameworks, incoming request payloads (often JSON or XML) are transformed from byte streams or strings into an object. This process is called Deserialization. Through implementation errors or misconfiguration, it may be possible to abuse this deserialization to run the attackers code or cause a DoS.

- **Using Components with Known Vulnerabilities.** Often exploits for commonly used libraries and frameworks become readily available. Using software with known vulnerabilities may facilitate compromise and reduce the effectiveness of other controls.

- **Insufficient Logging and Monitoring.** Insufficient logging and monitoring make it easier for attackers to attack a system, maintain persistence, pivot to more system, and tamper, extract or destroy data. Most breaches studies show time to detect a breach is over 200 days, typically detected by external parties.

Notable mentions from previous OWASP Top 10 releases:

- **Cross-Site Request Forgery (CSRF) [10].** A CSRF attack forces the victim's browser to send an authenticated request on behalf of the victim without the victim's knowledge to the target application. CSRF attacks leverage the fact that browsers automatically include cookies and authorization headers when sending a request to an application it has previously visited.

- **Unvalidated Redirects and Forwards [10].** Application often redirect users to other locations. Often this happens based on requests parameter (*e.g.,* OAuth2's 'redirect_uri' parameter [11]). When these redirect parameters are not validated, an attacker can trick victims in browsing web pages controlled by the adversary. The fact that these redirect links come from the application itself helps in giving credibility to the malicious pages.

3.2 Flask Security and NIST Hardening Guide

Most recommendations on Flask's *security Considerations [7]* page and NIST's Server Hardening guide [6], refer to secure configuration aspects. These would fall under OWASP's Top 10 *Security Misconfiguration* presented previously.

Chapter 4

Risk Analysis and Threat Model

Threat modelling is a process by which we attempt to draw a profile of probable attackers, the most likely attack vectors, and the assets most desired by an attacker [12]. Risk Analysis is then the process of identifying and estimating risks to the identified assets, given profiles of attackers [13].

Cybersecurity is by nature asymmetric. Considering, as an example an email service in which only legitimate users can access only their own mailboxes: even such a system can be composed of various subsystems, such as a front-end, database, access control components, and email reading and sending components. An adversary has numerous possibilities for attacking the system. Any subcomponent could be compromised independently from each other. An attacker for example might attack the front-end, injecting code, which when executed in the context of a legitimate user's browser, leaks information, or the attacker might exploit a vulnerability in the operating system. In contrast, engineers developing and implementing security measures must consider the security of the entire system. Covering all possible attack scenarios is simply not feasible. Thus, to discuss attack surface and attack vectors, first it is necessary to define, which are the components to protect, and the motivation and skill level of possible attackers, in order to assess the probability and impact of an incident happening.

4.1 System Scope

Firstly, it is necessary to define the scope of the system that is going to be protected, which determines the scope of the following risk analysis. Figure 4.1 depicts the network structure of the system.

- **Reverse Proxy.** The entry point to the system is an Apache [14] server running inside a virtual machine, behind an iptables [15] software firewall. The Apache server acts as a reverse proxy, forwarding requests coming from the internet to the Flask application, located on a separate machine. The reverse proxy exposes aside from HTTP and HTTPS also SSH for administrative access. It is the only machine

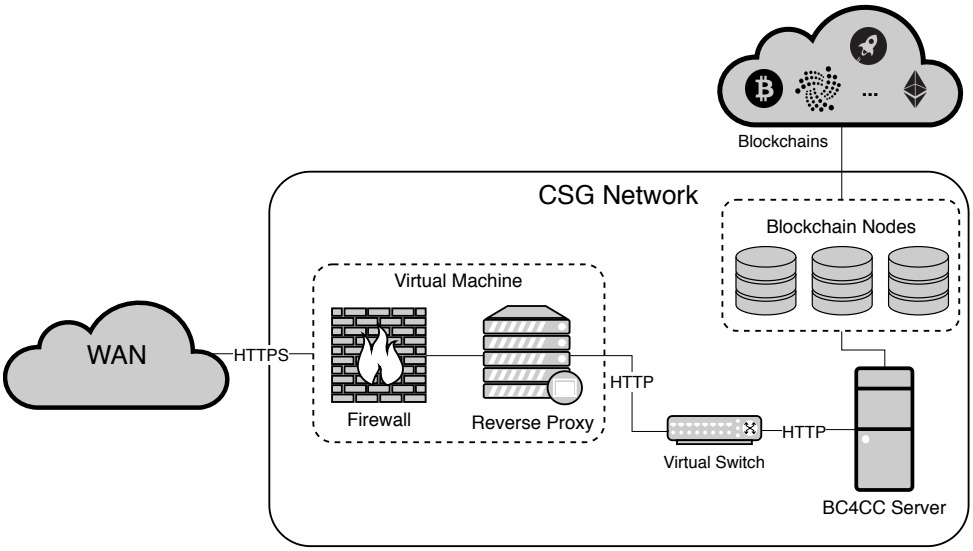

Figure 4.1: System Networking Structure

exposed directly to the internet, and as such its availability is of critical importance. TLS traffic terminates here and is further forwarded as HTTP. The OS is Ubuntu 18.04.3 LTS.

- **Virtual Switch.** Software program which forwards packets from the VM to the Flask application.
- **BC4CC Server.** Physical server hosting the Flask application implementing the business logic of BC4CC and the database. It is a physical machine, a Dell XPS desktop, running Ubuntu 18.04.3 LTS.
- **Blockchain nodes.** RPC servers needed to send and read transactions, necessary for the core functionality of BC4CC.

4.2 Stakeholders

A stakeholder is any individual or group which is influenced by a platform's successes or failures [16]. A stakeholder while being invested in the success of a system, might also be a threat source, as seen in Section 4.4.

- **Normal Users.** A normal user can use the application through the front-end or the API and only see their own transactions or a transaction in case they have the TX id. A normal user can be an employee of a logistics company or authorities auditing the cold chain.
- **Admin User.** In addition to the permissions of a normal user, can create new users and view any transaction on the platform.

- **System administrator.** System administrators of the platform, does not have access to the application directly, but manages the infrastructure.
- **Communication Systems Group (CSG).** The CSG developed the prototype and is interested in its smooth and secure operation.

4.3 Assets

In the following, the *assets* present in the system and their relevance to security are discussed. The ISO 27000 standard defines an *asset* simply as *"any item that has value to an organisation"* [17]. This does not simply include information, as one might be inclined to think at first thought, but can take many forms such as: information, software, physical assets (*e.g.,* computers), services, people, and intangible assets (*e.g.,* reputation).

Usually, assets are subdivided in different categories such as physical, logical, persons, and intangible goods [18]. Inside the same categories we often find the same type of vulnerabilities; thus, these are grouped accordingly. Assets are then, associated to a certain state, depending on whether it complies with security requirements [18].

4.3.1 Logical Assets

Logical assets can be divided into two categories, *(a)* software, and *(b)* information [18].

Software

This category includes operating systems and application software. All software can either be up-to-date; thus, patched with all available patches for known vulnerabilities or outdated.

Information

This category includes all data that is of value to stakeholders. The state of information can be determined by the security property associated with the information. For example, if a required property is availability, its state can be negatively affected if a legitimate user cannot access the information.

- **Username & Passwords for the application or access tokens.** While all transactions are not confidential (assuming they happen on public BCs), access to credentials allows to act on the behalf of the user, and as such should remain confidential.
- **Username & Passwords or access keys to the system components.** Should only be known to the system administrator and should also remain confidential.

- **Transaction information.** Integrity is guaranteed by the BCs. Regarding confidentiality, we should distinguish between transactions happening on public BCs and private ledgers:
  - **Public BC.** Information has to be available to anyone per as per BC definition. The availability is guaranteed by the decentralization of the BC itself, in case the BC4CC application were to be offline.
  - **Private ledger.** Information has to be available only to authorized users. Here the application has to apply access control to make sure, unauthorized users cannot access TX information. The confidentiality guarantees of the private ledger itself are beyond the scope of this audit. Again availability would be guaranteed, however, to legitimate users by checking the ledger itself, in case the application were offline.
- **BCs private keys.** Private keys are even more critical than user credentials, as they allow to impersonate a user beyond the confines of the BC4CC application directly on the BC. As such, their confidentiality is of utmost importance. Additionally, compromised credentials can be changed, compromised private keys cannot.
- **Backups.** In an incident, backups are necessary to avoid losing state and resuming operations as quickly as possible. Backups should be stored encrypted and only be accessible by the system administrator. Considering the system stores essentially private keys to various cryptocurrencies, a loss or leak of backups, would have catastrophic consequences to the platform and its image, with the effect of a loss of customers.

4.3.2 Persons

All the personnel involved in the development and operations of BC4CC.

- **System administrators.** Maintain the servers and VMs. Administrators have access to all critical data on the system. They keep the whole system up to date and running and are responsible for the secure configuration of the platform.
- **Developers.** Tasked with the continued development of the application and are inherently responsible for application security.

4.3.3 Intangible Goods

Intangible assets are typically of qualitative nature and reflect the intended public image of the service provider.

- **Customer confidence.** Since the customer and auditors use BC4CC to ensure that no Service Level Agreement (SLA) violations took place, the integrity of the information provided is crucial to maintain customer confidence, which is a prerequisite for a successful business relationship.

- **Timeliness.** Data needs to be available when the customers require it, as there are often time restrictions in place.

4.4 Threat Sources

This section identifies and describes the most relevant threat sources in the context of the prototype. They are the following:

- **Employees.** System administrators, developers and anyone who has physical access to the machines running BC4CC have to be taken into account (this includes for example also the cleaning and the maintenance team). Employees might have malicious intents (for example stealing the funds managed by the system), but also well-meaning employees may cause unintentional damage.
- **Amateur Hackers.** These adversaries usually only possess basic computer knowledge and use mainly known vulnerabilities for which exploits are publicly available. The reverse proxy is connected directly to the internet and is thus exposed to attacks by even amateur hackers. The web page and API are also indirectly connected. They may be motivated by the private keys to get to the cryptocurrencies stored or simply the challenge.
- **Skilled Hackers.** A skilled hacker has expert knowledge in applications and networking. They will usually carry out more complex attacks and might even use unpublished vulnerabilities (zero-days). They may be hired by a competitor or act on their own. They may also act out of their own desire for glory and challenge.
- **Competitors.** Usually these adversaries do not carry out their own attacks but hire someone else to do them. They are usually interested in espionage, although damaging the image of the competition to sway potential customers might also be a goal.
- **Malware.** Although malware such as viruses or worms may be used by the other categories, there is still the possibility of undirected attacks, which is why it is also listed as its own source. The goal for undirected malware attacks might be to control as many machines as possible to create a botnet, which can later be used to launch distributed denial of service attacks. Or ransomware for monetary gain.

For the given system other possible threat sources might be, *(a) Organized crime*, and *(b) Governmental Agencies*.

The former *(a)*, is typically motivated by monetary gain. However, unless BC4CC will hold millions in cryptocurrencies (or rather the private keys to the funds), it seems unlikely that organized crime would be interested. Assuming organized crime is using BC4CC to transport medical drugs and give legitimacy to their business, another possibility is that, they might be interested in stealing private key to fake their cold-chain data to lower transportation costs in order to undercut competitors. While the latter *(b)*, are usually

motivated by espionage, surveillance and criminal prosecution. BC4CC is a Business-to-Business (B2B) platform and as such it probably would not warrant much interest from the government, unless there is a suspicion of money laundering or tax fraud. Knowing this both actors would unlikely pose a threat to the system, not out of a lack of skills (it is rather the opposite, they would probably be the most skilled) but rather out of a lack of incentive.

4.5 Risk Evaluation

The NIST defines risk as a function of the likelihood of a threat event happening, and the impact, the adverse effect, such an event has on the organization [13]. Thus, measures for both impact and likelihood, and the function by which to compute the resulting *risk* must be defined. Given the difficulty in assigning an absolute value to these measures, it was preferred to use a five-step qualitative scale [13] as presented in Table 4.1 and Table 4.2

In order to estimate the risk associated with an event, first, it must be defined which is the **impact** of this event in case that it occurs. Table 4.1 presents the five steps of the impact severity.

Table 4.1: NIST Impact Definitions [13]

Severity	Description
Very High	The event would have multiple severe or catastrophic adverse effects, in such a way that recovery might not possible.
High	The event would have a severe or catastrophic adverse effect, in such a way *(i)* to cause a severe degradation or loss in mission capability; *(ii)* cause major damage to assets and/or financial loss; or *(iii)* result in human death or injury.
Moderate	The event would have a serious adverse effect, in such a way *(i)* to cause degradation in mission capability but its extent and duration would still allow an organization to perform its primary functions; *(ii)* result in significant damage to assets and/or financial loss; or *(iii)* result in significant human injury.
Low	The event would have a limited adverse effect, in such a way *(i)* to cause degradation in mission capability but its extent and duration would still allow an organization to perform its primary functions *(ii)* result in minor damage to assets and/or financial loss; or *(iii)* result in minor harm to individuals.
Very Low	The event would have negligible adverse effect.

Further, following [13], the **likelihood** of a threat event be initiated from an adversarial standpoint is defined in Table 4.2.

Table 4.2: NIST Likelihood Definitions [13]

Frequency	Description
Very High	The threat source is highly motivated and sufficiently capable and is almost certain to initiate a threat event. The controls put in place are ineffective.
High	The threat source is highly motivated and sufficiently capable and is highly likely to initiate a threat event. The controls put in place are ineffective.
Moderate	The threat source is motivated and capable. The controls put in place might impede the adversary.
Low	The threat source lacks the motivation or is not capable of initiating a threat event. The controls put in place might severely impede the adversary.
Very Low	The threat source is neither motivated nor capable of initiating a threat event. The controls put in place are effective.

Having defined all necessary parameters, a **risk evaluation**, following the examples found in [18], is conducted on the assets defined in Section 4.3 and is presented in the following sections.

4.5.1 Logical Assets: Software

No.	Threat	Countermeasure(s)	Likelihood	Impact	Risk
1	Employees (system administrators): unintentional misconfiguration of software results in loss of availability, confidentiality or in weakened controls.	Introduce regular reviews of configurations. Security design principles such as *"Compartmentalization", "Least Privilege" and "Fail-safe Defaults"* can help mitigate the effects [19].	*Moderate*	*Moderate*	*Moderate*
2	Employees (system administrators): outdated Operating System (OS). Many common security issues can be avoided by keeping the OS up-to-date [6].	Introduce a patch management process [20].	*High*	*Moderate*	*Moderate*
3	Employees (system administrators): no vulnerability management process is in place. Keeping the system up-to-date is the first step, for the remaining vulnerabilities there is a need to analyse, categorize and prioritize vulnerabilities and their countermeasures [20].	Introduce a vulnerability management process [20].	*High*	*Moderate*	*Moderate*

4	Employees (system administrators): system administrators with malicious intent introduce sniffing software leveraging the fact that all internal traffic is routed in clear-text.	Review all changes to the OS, software installed and configurations.	*Low*	*Very High*	*Moderate*
5	Employees (system administrators): No incident recovery process. In case of an incident, the main goal should be to regain operability of the system.	Exercising this event in test runs helps in recovering quickly from an incident and exposing possible problems in backup management.	*Moderate*	*Very High*	*High*
6	Employees (system administrators): insufficient logging and monitoring. Monitoring and logging are vital in detecting incidents, countering them and determining the scope of the damage done.	Consider introducing a logging infrastructure, ideally, shipping logs to a separate remote system to counter tampering by adversaries.	*Moderate*	*Very High*	*High*
7	Employees (developers): development mistake results in loss of availability, confidentiality or in weakened controls.	Regular reviews of all changes and invest in employee training.	*Moderate*	*High*	*Moderate*
8	Employees (developers): Outdated third party dependencies are in use. A common entrypoint for hackers are third party dependencies with known vulnerabilities (see A9, Owasp Top 10 [5]). The older the outdated dependency is, the higher the chance that vulnerabilities have been found and published.	Regular reviews of third party dependencies and introduce a patch management process. Introduce automated dependency checkers.	*High*	*High*	*High*
9	Employees (developers): developers with malicious intent introduce backdoors to the system, compromising confidentiality and integrity of the platform.	Regular reviews of all changes.	*Very Low*	*Very High*	*Moderate*

10	Amateur hackers: a hacker gains low privilege control over the reverse proxy server. From this position it could snoop on traffic to the application in clear-text, attempt privilege escalation, bring the server offline or mount an attack on the BC4CC server.	Keeping software up-to-date. Regular monitoring and introducing an Intrusion Detection System (IDS). Encrypt internal traffic. Strong password policies and *Least Privilege principle* applied to the server to avoid low privilege user from accessing sensitive information.	*Very Low*	*Very High*	*Moderate*
11	Amateur hackers: a hacker gains high privilege control over the reverse proxy server. From this position it could snoop on traffic to the application in clear-text, install a rootkit to maintain persistent control, bring the server offline and mounting an attack on the BC4CC server would be much simpler.	Keeping software up-to-date. Regular monitoring and introducing an IDS. Encrypt internal traffic. Strong password policies and *Least Privilege principle* applied to the server to avoid low privilege user from accessing sensitive information.	*Very Low*	*Very High*	*Moderate*
12	Skilled hackers: a hacker gains low privilege control over the reverse proxy server. From this position it could snoop on traffic to the application in clear-text, attempt privilege escalation and mount an attack on the BC4CC server.	Regular monitoring and introducing an IDS. Encrypt internal traffic. Strong password policies and *Least Privilege principle* applied to the server to mitigate risk of privilege escalation.	*Low*	*Very High*	*Moderate*
13	Skilled hackers: a hacker gains high privilege control over the reverse proxy server. From this position it could snoop on traffic to the application in clear-text, install a rootkit to maintain persistent control, bring the server offline and mounting an attack on the BC4CC server would be much simpler.	Same countermeasures as in Risk 12.	*Low*	*Very High*	*Moderate*
14	Competitors: competitors hire skilled hackers to conduct industrial espionage.	Same countermeasures as in Risk 12.	*Moderate*	*Very High*	*High*
15	Undirected malware spreads through the internet/internal network, causing a loss of availability, corrupting files (integrity).	Patch and vulnerability management, review of firewall rules, Enumerate all networked services and disable any which are unneeded.	*Moderate*	*Very High*	*High*

16	Network Denial of Service.	Make use of load balancing, rate limiters, or third party services for DoS mitigation.	*Moderate*	*Moderate*	*Moderate*
17	Software Denial of Service: bad input or computationally intensive operations, for example regular expressions or public cryptography computations, that can be triggered by a user can cause accidental or intentional DoS.	Conduct a review of the source code to expose any performance bottleneck or problematic points where a lot of computational power is needed. These places should have very strict rate limits. Possible examples in the context of BC4CC could be public cryptography computation.	*Low*	*Moderate*	*Low*

4.5.2 Logical Assets: Information

No.	**Threat**	**Countermeasure(s)**	**Likelihood**	**Impact**	**Risk**
18	Phishing campaign to steal application credentials.	Monitoring to detect unusual traffic origins and patterns.	*High*	*Very Low*	*Low*
19	Application user credentials are leaked due to an adversary sniffing traffic and intercepting credentials.	Ensure all external, as well as internal, traffic is encrypted.	*High*	*Low*	*Moderate*
20	Application user credentials are leaked due to an adversary bruteforcing credentials online.	Enforce strong password policies and use of multi-factor authentication (MFA). Use rate limiters which block *(i)* IP addresses, *(ii)* accounts, and in extreme cases *(iii)* geographical regions and network prefixes.	*Moderate*	*Low*	*Low*
21	Application user credentials are leaked due to an SQL injection.	Sanitize all parameters used in database queries. Ensure credentials are salted and hashed before saving to impede offline brute-forcing. Enforce strong password policies and MFA.	*Low*	*High*	*Moderate*
22	Application user credentials (cookies) are leaked due to a code injection in the front-end or cross-site request forgery.	Sanitize all parameters used in HTML templates. Use cookie flags to harden security: http-only, secure, same-site.	*High*	*Low*	*Moderate*

23	System credentials are leaked or brute-forced.	Ensure credentials and system accounts are only shared on a need-to-know basis. Allow only strong authentication mechanisms (*e.g.,* public key authentication for SSH) and brute-force prevention software (*e.g.,* fail2ban).	*Low*	*Moderate*	*Low*
24	Transactions information is tampered with on the BC of origin	Integrity is provided by the BCs.	*Very Low*	*High*	*Low*
25	Transactions information is tampered with when being retrieved from the BC nodes or when transmitting to the user's clients, falsifying the TX information shown.	Ensure the RPC connections to the BC nodes happens over an authenticated channel which supports integrity [21] and all data is transmitted over HTTPS to users.	*Low*	*Low*	*Low*
26	Transaction is tampered with when sending to a node to cause a denial of service or when the users submits the transaction.	Use of network analysis tools to detect attempts to man-in-the-middle the connection. Physical connection to the node(s) to avoid tampering. Use of HTTPS.	*Low*	*Low*	*Low*
27	BC private keys are leaked through SQL injection	Sanitize all parameters used in database queries.	*Low*	*Very High*	*Moderate*
28	BC private keys are leaked by accessing database backups	Encrypt database backups. Store encryption keys on a separate machines. Restrict access to backups and encryption keys.	*Moderate*	*Very High*	*High*
29	BC private keys are compromised due to an adversary accessing the BC4CC server and reading the clear-text keys.	Consider using a hot-wallet, cold-wallet approach [22][23]. Move funds as needed, keeping most of them on the accounts linked to the keys stored in the cold-wallets. Additionally, consider moving all signing operations outside of the BC4CC application using a tamper-proof Trusted Platform Module (TPM).	*Moderate*	*Very High*	*High*

30	Backups are leaked.	Encrypt backups. Restrict access to backups and encryption keys as much as possible. In case of leak of encrypted backups, the data stored has to be considered as compromised and the assets linked to the private keys must be moved at once.	*Moderate*	*Very High*	*High*
31	Backups are deleted unintentionally (system administrator mistake or hardware/software malfunction) or maliciously.	Keep multiple redundant copies of the backups off-site. Restrict access to backups and encryption keys to a minimum. Credentials to offsite backups should be kept outside of the system.	*Moderate*	*Very High*	*High*

4.5.3 Persons

No.	Threat	Countermeasure(s)	Likelihood	Impact	Risk
32	Developers or system administrators: sudden illness or accident results in interruption or termination of employment.	Contractual framework for documentation, configuration and knowledge transfer.	*Low*	*Low*	*Low*
33	Developers or system administrators: bribery to leak confidential information.	Non-disclosure agreements or contractual policies.	*Low*	*High*	*Moderate*

4.5.4 Intangible Goods: Customer Confidence

No.	Threat	Countermeasure(s)	Likelihood	Impact	Risk
34	Undisclosed security incident is made public: *e.g.,* an undisclosed leak is made public by the press would ruin the image of BC4CC.	A transparency policy on how to handle security incidents should be prepared in advance.	*Low*	*Moderate*	*Low*
35	A software problem or DoS causes the application to not be available when needed.	An agreement with the customers regarding what is considered an acceptable and not acceptable delay as well as what constitutes a delay which is beyond control (*e.g.,* in case of a denial of service).	*Moderate*	*Moderate*	*Moderate*

4.6 Risk Matrix

Having defined the **impact**, **likelihood**, and the **risk evaluation**, each threat event is assigned to a defined risk level, resulting in the risk matrix presented in Table 4.7.

Table 4.7: Risk Matrix [13]

Likelihood/Impact	Very Low	Low	Moderate	High	Very High
Very High					
High	[18]	[19, 22]	[2, 3]	[8]	
Moderate		[20]	[1, 16, 35]	[7]	[5, 6, 14, 15, 28, 29, 30, 31]
Low		[25, 26, 32]	[17, 23, 34]	[21, 33]	[4, 12, 13, 27]
Very Low				[24]	[9, 10, 11]

Table 4.8: Risk matrix legend

Color	Legend
	Very High - Not Acceptable. Risk reduction required
	High - Risk reduction necessary or risk acceptance after further analysis by stakeholders.
	Moderate - Acceptable using ALARP. Consider further risk reduction.
	Low - Acceptable
	Very Low - Acceptable.

The matrix shows, that most risks are in the moderate-high region. These need to be addressed, either by accepting the risks or by introducing measures, before release. Considering the sensitivity of the data stored on the application, a high-level risk is to be expected.

Chapter 5

Security Audit

This sections presents the security audit of the prototype deployed at `https://bc4cc.ddns.net`, which was carried out from September 2019 to December 2019. At the moment of writing this review, the system is still under development.

5.1 Review of the system architecture

The general architecture of the system is well suited for the requirements. The design is simple and usable and only the firewall being connected to the internet is a good way to minimize exposure. Simplicity is an important security property, often overlooked [19]. Simple systems are easier to analyse and are less likely to contain flaws than complex ones. The choice of having only a reverse proxy exposed to the internet would also simplify DoS protection. All traffic goes through the reverse proxy, thus the proxy could be extended to rate limit requests which do not contain a valid, session cookie, and freely let through the rest of the requests. If a logged-in user is misbehaving, this can be handled on the application layer by rate limiting user requests as well. If the load is such that even with the countermeasures in place, the reverse proxy cannot handle the traffic, additional proxies can be added for load balancing.

There are some improvement points possible. Communication inside the perimeter happens over HTTP, a channel which does not provide confidentiality: an adversary which sits behind the firewall, would be able to listen in on any traffic to the application, including user credentials and TX information for private ledgers which should not be made public. Given that the university is a public institution, any number of people could enter the premise and depending on the internal UZH network configuration, access the CSG network from behind the firewall. Aside from passive traffic sniffing, also active MITM is possible, since HTTP is not an authenticated channel. A defence-in-depth approach would be to encrypt traffic even behind the perimeter to mitigate the risk.

Eventually, when the application is deployed to production, special care should be taken to protect the connection between the BC4CC server and the RPC nodes. As active interference of this connection would render the application unusable due to not being

able to fulfill its functional requirements anymore. In this sense, a single connection to a BC node is a single point of failure in the system. A possibility would be to introduce multiple connection to various remote nodes, to avoid a possible separation attack between BC4CC and the various BCs. At the moment the nodes are running in docker containers with a direct interface to the BC4CC application, thus the connection is secure, in the sense that it cannot be interrupted or interfered with.

The storage of private keys could be further improved. At the moment all keys are stored as plain-text in the database. This is necessary in order to compute a transaction signature before sending the TX to the BC. This does have the disadvantage that if an adversary were able to access the BC4CC server, all private keys would be compromised. A simple, yet effective, improvement is to delegate the signing operation to a further system which exposes an API to compute signatures and/or to use a tamper-proof hardware module to store the private keys that handles transaction signing. This approach can be seen in Figure 5.1. This brings two advantages: *(i)* a compromise of the BC4CC server does not compromise the private keys, and *(ii)* the business logic becomes simpler, which means that it is easier to reason about, which in turn increases the security of both the BC4CC and Private Keys Server. Additionally, BC4CC could employ a hot-wallet, cold-wallet approach [22][23]. Two distinct sets of private keys are needed. The first set would be further stored as plain-text and directly accessible to BC4CC. The second set would be stored offline, in a hardware wallet. The majority of the cryptocurrencies would be stored in the cold-wallet. As the funds in the hot-wallet are used, more can be transferred from the cold-wallet in a controlled manner. Both approaches can be used in parallel in order to further reduce risk of compromise.

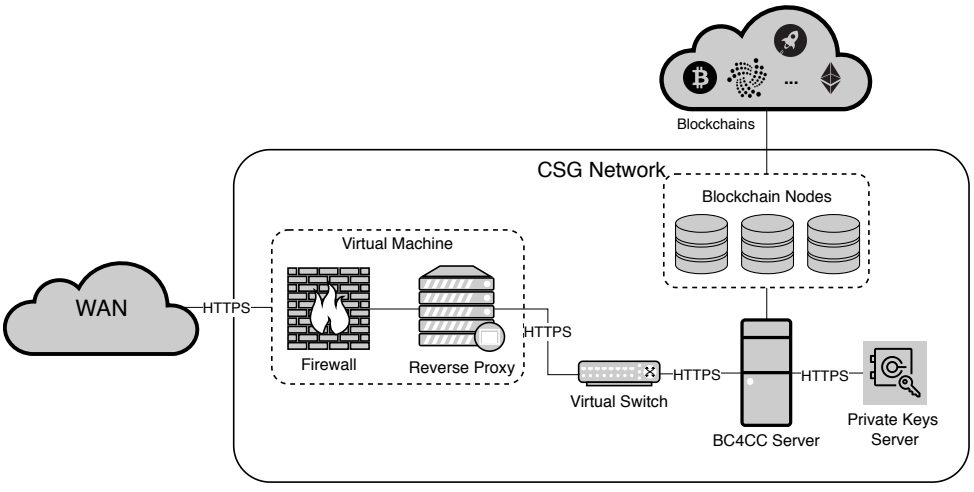

Figure 5.1: Separation of the signing component from the BC4CC server.

5.2 Security Analysis

This section presents the analysis of vulnerabilities found on the reverse proxy and the BC4CC server. Each finding is assigned a score, which represents a subjective opinion on

the severity of the finding as well as the cost-effectiveness of implementing the measure. The range of scores is presented in Table 5.1.

Priority	Description
1	Great benefit with minor effort.
2	Implementation recommended, but effort or usability should be taken into account.
3	Large effort leading to a small benefit.

In addition, each finding is assigned a category based on the finding classification found in the Open Source Security Testing Methodology Manual (OSSTMM) [24]. Each category is described in Table 5.2.

Category	Description
Vulnerability	Flaw or error which allows for privileged access to a certain infrastructure.
Weakness	Flaw or errror which reduces, or nullifies the effectiveness of security controls.
Concern	Not a direct threat, but not compliant with best-practices.
Exposure	Disclosure of sensitive information.
Anomaly	An unknown in the system which could not be identified.

5.2.1 Security analysis on the Reverse Proxy

Since the job of the reverse proxy is to forward requests, the audit of this server only includes a review of any possible mechanism which would allows an adversary to bypass any security implemented on the proxy to access BC4CC. In particular, we are interested in which services and OS are running on this machine.

The reverse proxy is running the latest Ubuntu LTS version (18.04.3), with all recent security patches installed. An nmap scan [25] show the following open ports:

```
$ nmap -P0 -sV --reason -vvv bc4cc.ddns.net

PORT     STATE  SERVICE      REASON          VERSION
22/tcp   open   ssh          syn-ack ttl 56 OpenSSH 7.6p1 Ubuntu
80/tcp   open   http         syn-ack ttl 56 Apache httpd 2.4.29 ((Ubuntu))
113/tcp  closed ident        reset ttl 62
443/tcp  open   ssl/ssl      syn-ack ttl 56 Apache httpd (SSL-only mode)
2000/tcp open   cisco-sccp?  syn-ack ttl 62
5060/tcp open   sip?         syn-ack ttl 62
8080/tcp closed http-proxy   reset ttl 56
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Listing 5.1: Nmap scan of the reverse proxy

Findings

This section discloses all security related findings exposed during the audit of the reverse proxy VM.

No.	Category	Flaw	Measure	Priority
1	Vulnerability	**Outdated OpenSSH**. According to nmap's banner grabbing seen in Listing 5.1, the version in use of OpenSSH is outdated (v7.6p1, released in 2017). Overtime, security patches are released for software. According to NIST National Vulnerability Database (NVD), there are multiple vulnerabilities affecting this version.	Update OpennSSH to the latest version. Note: not all vulnerabilities necessarily affect the system even if the version matches the one mentioned in the NVD.	1
2	Concern	**Exposed *phpPgAdmin* page**. The reverse proxy exposes the administrative interface *phpPgAdmin* to the internet. An attacker may gain full control over the database of the affected system after a successful authentication or authorization bypass.	If not strictly required, disable access over internet. Otherwise restrict access by IP address or force use of a VPN.	1
3	Concern	**Exposed SSH service**. From the internet, SSH is reachable on port 22 and the version banner is visible. If a vulnerability is published, there is nothing to stop any adversary from exploiting it to gain administrative access to the VM.	Traffic should be filtered to reduce attack surface (*e.g.,* allow only traffic from the internal network to port 22).	2

4	Anomaly	**VOIP ports exposed**. From the internet, ports 2000 and 5060 are visible. Both are VOIP related procotols. However these services do not appear to be running on the reverse proxy VM. The TTL (seen in the nmap scan) shows a higher value than the one shown by the Apache server, which seems to be pointing to another machine further upstream exposing these ports.	No action is required.	3

5.2.2 Security Analysis on the Flask-based Wrapper

The BC4CC server is where the application business logic is implemented and where all sensitive data is stored. As such this audit includes both a black-box, as well as, a white-box testing approach.

The BC4CC server is running the latest Ubuntu LTS version (18.04.3), with all recent security patches installed. Since the BC4CC server is not reachable from the internet, port scanning was done from the Reverse Proxy server. The nmap scan [25] show the following open ports:

```
$ nmap -P0 -sV --reason -vvv bc4cc-server

PORT     STATE SERVICE        REASON         VERSION
22/tcp   open  ssh            syn-ack        OpenSSH 7.6p1
80/tcp   open  http           syn-ack        Apache httpd 2.4.29
3306/tcp open  mysql?         syn-ack
4004/tcp open  zmtp           syn-ack        ZeroMQ ZMTP 2.0
5000/tcp open  http           syn-ack        Gunicorn 19.9.0
5222/tcp open  http           syn-ack        Gunicorn 19.9.0
8000/tcp open  bitcoin-jsonrpc syn-ack       Bitcoin JSON-RPC v0.10.0.0-unk
8008/tcp open  http           syn-ack        aiohttp 2.3.2 (Python 3.5)
```

Listing 5.2: Nmap scan of the BC4CC application server

Findings

This section discloses all security related findings exposed during the audit of the BC4CC server.

No.	Category	Flaw	Measure	Priority
5	Vulnerability	**Outdated OpenSSH**. According to nmap's banner grabbing in Listing 5.2, the version in use of OpenSSH is outdated (v7.6p1, released in 2017). Overtime, security patches are released for software. According to NIST National Vulnerability Database (NVD), there are multiple vulnerabilities affecting this version.	Update OpennSSH to the latest version. Note: not all vulnerabilities necessarily affect the system even if the version matches the one mentioned in the NVD.	1
6	Weakness	**Cookie without *SameSite* flag.** The session cookie is set without the *SameSite* flag. The browser will always send the cookie with every request, even if it was initiated by another domain (cross-origin requests). This simplifies Cross-Site Request Forgery (CSRF). The absence of the flag could be exploited to perform actions on the user's behalf.	The cookie should be flagged with *SameSite=strict* [26] [27].	1

7	Vulnerability	**No CSRF protection.** None of the actions in the application is protected against Cross-Site Request Forgey (CSRF). This means that an attacker may execute arbitrary actions in the name of another user by making the user visit a specially crated rogue web page. Concretely, it was possible execute transactions by tricking a user into visiting a specially crafted web page. If the victim is an admin, it is possible to create new users.	All state-changing operations require CSRF protection [28] [29]. More information is provided in Section 5.2.2.	1
8	Concern	**Cookie without *Secure* flag**. Without the flag, web browsers send the cookie along with unencrypted HTTP requests, which could be read by an adversary.	If possible, all cookies should be set with the secure flag [30] [31].	1
9	Weakness	**Session cookie is not invalidated on logout & the session never times out.** When logging out the session token is only marked as *logged out* on the client, but it remains active on the server. Coupled with the fact that session tokens do not appear to expire, if an attacker is able to hijack the session of a user, he will be able to use this session for a long time even if the user logs out.	The session should time out after a certain period of time (preferably short), and the session should be terminated on the server on user logout [32].	1

10	Weakness	**Cleartext Storage of Sensitive Information in a Cookie.** The application stores sensitive information in the cookie, which includes username and password. Adversaries who manage to get hold of a token, can easily read the information [33].	Remove sensitive information, as long as it not needed by the application, or replace coded session cookies with opaque cookies which are mapped on the backend to the information.	2
11	Concern	**User passwords are hashed without a salt.** The application stores passwords hashed but without the use of a salt. This makes it easy for an adversary to pre-compute hashes using rainbow tables [34] [35] [36].	Passwords must be hashed together with a salt [37].	1
12	Concern	**User passwords are hashed using an algorithm with insufficient computational effort.** The application stores passwords that were hashed using an algorithm which does not have a sufficiently high computational effort [38]. This means that a brute force attack is feasible.	Prefer one of the algorithms listed by OWASP [37] *e.g.,* Argon2. More details are provided in Section 5.2.2.	1
13	Concern	**No password change mechanism.** In case of user credential compromise, it is not possible to change the password. An attacker would be able to freely use the victim's account until a developer/system administrator changes the password for the user.	Allow users to update their own password.	2

14	Weakness	**User enumeration**. The clients page is visible to all logged-in users with name and email address. On this page, all users of the application are listed. This greatly simplifies a possible brute-force attack or phishing attempt [39].	Enforce that only administrators are allowed to view all users.	3
15	Concern	**Weak password policy.** There is no password policy in place, which increases the risk of users choosing weak passwords [40].	Introduce a password policy following the guideline of OWASP ASVS [41].	2
16	Concern	**Use of hard-coded credentials**. The flask code contains a hard-coded secret it uses for cookie signature. In case of a leak of this secret (for example by checking it into a code repository, sharing the code, etc.) an adversary would be able to create arbitrary cookies and thus bypass authentication [42].	Consider passing the secret to the application at runtime as an environment variable.	3

17	Concern	**Clickjacking/UI Redress Attack**. The server does not set the HTTP header *X-Frame-Options*. As such the application is not sufficiently protected against *Clickjacking*. Clickjackings is a significant risk for users. An adversary could embed the vulnerable web application into his own and use it as a transparent overlay, only showing the content of his own website. This content could induce a user to interact and trigger actions in the invisible embedded website.	If not strictly necessary, disable embedding of web pages setting the *"X-Frame-Options"* header to *"deny"*	1
18	Weakness	**Missing rate limiter on certain endpoints.** Certain API endpoints which require basic authentication are missing rate limiters. This makes it easy to brute-force user credentials, especially in combination with Finding 14.	Add rate limiters to all API endpoints.	1

Cross-Site Request Forgery

A Cross-Site Request Forgery (CSRF) is an attack in which an adversary tries to carry out a legal operation in the name of another user [5]. An example is effecting a payment or changing a password. These are operations a user is authorized to do. If an adversary is able in changing the password of a user, he has effectively taken over the account.

For the attack to be successful the following conditions need to be fulfilled:

- The user needs to have a valid session/be logged in at the time of the attack.
- The attacked needs to lure the user to a specially crafted website.

As soon as the user lands on the attacker's website, the *"unintended"* request is sent to the server. As the user already has a valid session, the browser sends the session token, stored in the cookie, to the server as well and the request is authorized.

```
fetch('https://bc4cc.ddns.net/api/createclient', {
    credentials: 'include',
    method: 'POST',
    body: 'username=csrf&password=test' +
          '&name=test&email=test%40test.com&phone=test',
    headers: {
        'Accept': 'text/html',
        'Content-Type': 'application/x-www-form-urlencoded'
    }
});
```

Listing 5.3: CSRF exploit code

In the case of BC4CC if an authenticated admin user goes to a website containing the Javascript code contained in Listing 5.3, a new user account will be created without the knowledge of the user.

CSRF protection, thus, has to make sure that an operation is carried by the user on purpose. This may be achieved if for every request of the browser in the response of the server, an additional token is returned. This token must not be stored as cookie, otherwise it will always be included by default as the session cookie. This token needs to be reused with the next request. The server verifies whether the token in the new request matches with the token in the previous response. This procedure makes sure that the user triggers the operation on purpose.

Password Storage

A key concern when using passwords for authentication, is how to verify passwords. The naïve solution, is to simply store passwords in clear text and check if they match on login requests. This has the disadvantage that if the user database is ever leaked, all credentials are immediately compromised. A better approach is to compute a cryptographic hash function [43] of a password and store only the hash in the database [37]. Knowledge of the hashing algorithm and clear text password, provided by the user on login (which is immediately discarded after verification), is enough to authenticate the user. Additionally, a *"salt"*, a per-password fixed-length random value, should be appended to the password when computing the hash. This has two goals:

- two identical passwords, do not appear the same once hashed.
- increase entropy with the aim of making pre-computed lookup attacks not feasible.

The hashing function used should not be too fast to compute, thus enabling a brute-force attack, while still being quick enough to be usable for legitimate purposes: if a hashing functions is too fast, an adversary can pre-compute lookup tables using word-lists and then use these to derive the user credentials; if it is too slow the application can become unusable and in extreme case enable DoS attacks. Special purpose-built adaptive algorithms like BCrypt and Argon2 are designed to be memory- and time-hard in order to make pre-computation attacks intractable [36].

In BC4CC passwords are stored hashed using SHA-512 without a salt. SHA-512 while being a cryptographic hash, was designed to be fast for signature verification, which makes it a poor candidate for password storage, indeed, SHA-512 is 6 orders of magnitude faster than BCrypt [44].

Chapter 6

Summary and Conclusions

The security audit has produced medium results within the scope of the project. The architecture is well-thought but there is margin for improvement. Handling of the private keys that give access to the cryptocurrencies needs to be hardened. A leak of the keys, has the potential for financial damage and Denial of Service, which would have severe consequences on the customer's confidence in the platform, directly influencing the possibility of success of the project. This document presents possible improvements on the existing architecture which would greatly impede an adversary that attempts to access the keys or limit the damage an incident could cause. Hardening measures aside, there are no fundamental errors in the design of the BC4CC application.

The implemented application however does present some issues. The most critical issue is the possibility of executing Cross-Site Request Forgery (CSRF). Through CSRF it is possible to create unverified user accounts, as well as performing transactions on behalf of the user. Creating unverified user accounts, removes any traceability of the operations occurred, and makes it impossible to recover any funds the user might have consumed. By generating enough CSRF requests the adversary has the possibility of locking a user out of the application by triggering the rate limiter. The CSRF vulnerability hurts the availability and non-repudiation security properties, makes any form of accounting impossible, as well as causing severe financial damage.

All findings as well as the architecture review can be found in Chapter 5.

This report has presented an all-encompassing audit of the BC4CC application. Starting with a risk analysis and through a penetration test, the application was verified in search of vulnerabilities and weaknesses both in the design as well as in the implementation. A comprehensive list of mitigation for risks and measures for findings were listed in Chapter 4 and Chapter 5. All mitigations and measures are not difficult to fix but they are critical enough, that they should be implemented before considering deploying BC4CC.

Bibliography

[1] T. Bocek et al. "Blockchains Everywhere - a Use-Case of Blockchains in the Pharma Supply-Chain". In: *IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2017)*. Lisbon, Portugal, May 2017, pp. 772–777.

[2] E. J. Scheid, B. Rodrigues, and B. Stiller. "Toward a Policy-based Blockchain Agnostic Framework". In: *IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2019)*. Washington - DC, USA, Apr. 2019, pp. 609–613.

[3] Communication Systems Group (CSG). *Blockchain Based Temperature Monitoring for Cold Chains in Medical Drug Distribution (BC4CC)*. [Online] `https://www.csg.uzh.ch/csg/en/research/BC4CC.html` last visit November 20, 2019. 2019.

[4] E. J. Scheid et al. "Bifröst: a Modular Blockchain Interoperability API". In: *IEEE Conference on Local Computer Networks (LCN 2019)*. Accepted. To be published. Osnabrück, Germany, Oct. 2019, pp. 1–9.

[5] OWASP Foundation. *OWASP Top 10 Web Application Security Risks*. [Online], `https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project#tab=Main`. Accessed: 2019-12-01. 2017.

[6] Karen Scarfone, Wayne Jansen, and Miles Tracy. "Guide to general server security". In: *NIST Special Publication* 800.s 123 (2008).

[7] *Flask - Security Considerations*. [Online], `https://flask.palletsprojects.com/en/1.1.x/security/`. Accessed: 2019-12-01.

[8] Kaitlin Boeckl (NIST) Joshua Franklin (NIST) Gema Howell (NIST). "Mobile Device Security: Corporate-Owned Personally-Enabled (COPE)". In: *NIST Special Publication* 1800.21 (2019).

[9] PCI Security Standards Council Emerging Technologies. *PCI Mobile Payment Acceptance Security Guidelines for Developers*. Tech. rep. Sept. 2017.

[10] OWASP Foundation. *OWASP Top 10 Web Application Security Risks*. [Online], `https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project#OWASP_Top_10_for_2013`. Accessed: 2019-12-01. 2013.

[11] Internet Engineering Task Force (IETF). *The OAuth 2.0 Authorization Framework*. Tech. rep. 2012.

[12] Suvda Myagmar, Adam J Lee, and William Yurcik. "Threat modeling as a basis for security requirements". In: *Symposium on requirements engineering for information security (SREIS)*. Vol. 2005. Citeseer. 2005, pp. 1–8.

[13] Joint Task Force Transformation Initiative et al. *Guide for conducting risk assessments*. Tech. rep. National Institute of Standards and Technology, 2012.

[14] *Apache - HTTP Server Project*. [Online], `https://httpd.apache.org/`. Accessed: 2019-12-01.

[15] *iptables - administration tool for IPv4 packet filtering and NAT*. [Online], https://linux.die.net/man/8/iptables. Accessed: 2019-12-01.

[16] H. Sharp, A. Finkelstein, and G. Galal. "Stakeholder identification in the requirements engineering process". In: *Proceedings. Tenth International Workshop on Database and Expert Systems Applications. DEXA 99*. Sept. 1999, pp. 387–391.

[17] *Information technology - IT asset management - Part 1: IT asset management systems*. Standard. Geneva, CH: International Organization for Standardization, 2017.

[18] David Basin, Patrick Schaller, and Michael Schläpfer. *Applied information security: a hands-on approach*. Springer Science & Business Media, 2011.

[19] Jerome H Saltzer and Michael D Schroeder. "The protection of information in computer systems". In: *Proceedings of the IEEE* 63.9 (1975), pp. 1278–1308.

[20] Peter Mell, Tiffany Bergeron, and David Henning. "Creating a patch and vulnerability management program". In: *NIST Special Publication* 800 (2005), p. 40.

[21] R Thurlow. *RPC: Remote Procedure Call Protocol Specification Version 2", RFC 5531*. Tech. rep. 2009.

[22] *Cold Storage*. [Online], https://en.bitcoin.it/wiki/Cold_storage. Accessed: 2019-12-01.

[23] *Hot Wallet*. [Online], https://en.bitcoin.it/wiki/Hot_wallet. Accessed: 2019-12-01.

[24] Pete Herzog. "OSSTMM 3-The Open Source Security Testing Methodology Manual: Contemporary Security Testing and Analysis". In: *ISECOM-Institute for Security and Open Methodologies* (2010).

[25] *Nmap: the Network Mapper*. [Online], https://nmap.org. Accessed: 2019-12-01.

[26] *SameSite*. [Online], https://www.owasp.org/index.php/SameSite. Accessed: 2019-12-01.

[27] *Configuration Handling*. [Online], https://flask.palletsprojects.com/en/master/config/#SESSION_COOKIE_SAMESITE. Accessed: 2019-12-01.

[28] *CWE-352: Cross-Site Request Forgery (CSRF)*. [Online], https://cwe.mitre.org/data/definitions/352.html. Accessed: 2019-12-01.

[29] *CSRF Protection*. [Online], https://flask-wtf.readthedocs.io/en/stable/csrf.html. Accessed: 2019-12-01.

[30] *CWE-614: Sensitive Cookie in HTTPS Session Without 'Secure' Attribute*. [Online], https://cwe.mitre.org/data/definitions/614.html. Accessed: 2019-12-01.

[31] *Configuration Handling*. [Online], https://flask.palletsprojects.com/en/master/config/#SESSION_COOKIE_SECURE. Accessed: 2019-12-01.

[32] *CWE-613: Insufficient Session Expiration*. [Online], https://cwe.mitre.org/data/definitions/613.html. Accessed: 2019-12-01.

[33] *CWE-315: Cleartext Storage of Sensitive Information in a Cookie*. [Online], https://cwe.mitre.org/data/definitions/315.html). Accessed: 2019-12-01.

[34] *CWE-759: Use of a One-Way Hash without a Salt*. [Online], https://cwe.mitre.org/data/definitions/759.html). Accessed: 2019-12-01.

[35] *OWASP Top 10 2010-A7-Insecure Cryptographic Storage*. [Online], https://www.owasp.org/index.php/Top_10_2010-A7-Insecure_Cryptographic_Storage). Accessed: 2019-12-01.

[36] Paul A Grassi, Michael E Garcia, and James L Fenton. "Digital identity guidelines". In: *NIST special publication* 800 (2017), pp. 63–3.

[37] *Use a cryptographically strong credential-specific salt.* [Online], https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#use-a-cryptographically-strong-credential-specific-salt). Accessed: 2019-12-01.

[38] *CWE-916: Use of Password Hash With Insufficient Computational Effort.* [Online], https://cwe.mitre.org/data/definitions/916.html. Accessed: 2019-12-01.

[39] *Testing for Account Enumeration and Guessable User Account.* [Online], https://www.owasp.org/index.php/Testing_for_Account_Enumeration_and_Guessable_User_Account_(OTG-IDENT-004). Accessed: 2019-12-01.

[40] *CWE-521: Weak Password Requirements.* [Online], https://cwe.mitre.org/data/definitions/521.html. Accessed: 2019-12-01.

[41] *OWASP Application Security Verification Standard.* [Online], https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project. Accessed: 2019-12-01.

[42] *CWE-798: Use of Hard-coded Credentials.* [Online], https://cwe.mitre.org/data/definitions/798.html. Accessed: 2019-12-01.

[43] William Stallings. *Cryptography and network security: principles and practice.* Pearson Upper Saddle River, 2017.

[44] Atishay Aggarwal, Pranav Chaphekar, and Rohit Mandrekar. "Cryptanalysis of bcrypt and SHA-512 using Distributed Processing over the Cloud". In: *International Journal of Computer Applications* 128.16 (2015).

List of Figures

List of Tables