2019

Exploring applicability of blockchain to enhance Single Sign-On (SSO) systems

Samuel Matloob
University of North Florida, bmatloob@gmail.com

EXPLORING APPLICABILITY OF BLOCKCHAIN TO ENHANCE SINGLE SIGN-ON (SSO) SYSTEMS

by

Samuel Matloob

A thesis submitted to the
School of Computing
in partial fulfillment of the requirements for the degree of

Master of Science in Computer and Information Sciences

UNIVERSITY OF NORTH FLORIDA
SCHOOL OF COMPUTING

December, 2019

ACKNOWLEGEMENTS

There are many people around me without whom I would not have been able to reach this milestone in my life. First and foremost, I would like to give special thanks to my wife and my sons who were very supportive and patient with me. I also would like to give special thanks to my thesis advisor Dr. Swapnoneel Roy and to my advising committee: Dr. Asai Asaithambi and Dr. Peyman Faizian, who supported me in my academic journey. Their advice, encouragement, patience, and great feedback helped my advancement at the personal level as well as the professional level, which in turn helped improve the research work significantly. Also, I would like to thank Cyber Florida as this research work was supported in part by their grant (#220384). Lastly, I would like to thank all professors at the University of North Florida who taught me during my master's program, as I have learned so much from each of them.

CONTENTS

List of Figures vi

List of Tables vii

Abstract viii

Chapter 1 INTRODUCTION 1

1.1 Background 3

1.2 Objectives of the SSO 5

1.3 The Proposed Solution to Single-Point Failure in SSO 6

Chapter 2 Blockchain Technology 7

2.1 Introduction to Blockchain 7

2.2 Blockchain Applications 8

2.3 Hyperledger Fabric Componenets 10

2.4 Initial Setup for this Project 11

2.4.1 Blockchain Setup Details 12

2.5 Implementation of Final Model 14

Chapter 3 Implementation Details 17

3.1 Requirements for the Model 17

3.2 Concept Implementation 18

3.3 Test Model 19

3.4 Hardware Used 21

3.5 Operating System and Software Used 22

3.6 Putting All Pieces Together 23

Chapter 4 Results and Analysis 26
4.1 Results. 26
4.2 Analysis 26
Chapter 5 Conclusion and future work 31
5.1 Conclusion. 31
5.2 Future Work 33
REFERENCES 34
Vita. 40

FIGURES

Figure 1.1	Single Sign On Concept	5
Figure 2.1	Implemented Blockchain Model	13
Figure 3.1	Single Sign On Login Page	20
Figure 3.2	Implementation of the Test Page	21
Figure 3.3	How the hardware are connected	24

TABLES

Table 4.1 Using Blockchain with 100 iterations and Round Robin decision making policy 27

Table 4.2 Using Database with 100 iterations and Round Robin decision making policy - traditional connection 27

Table 4.3 Using Database with 100 iterations and Round Robin decision making policy - Non-traditional connection 27

Table 4.4 Using Blockchain with 100 iterations and Random decision making policy 28

Table 4.5 Using Database with 100 iterations and Random decision making policy - traditional connection 28

Table 4.6 Using Database with 100 iterations and Random decision making policy - Non-traditional connection 28

ABSTRACT

Single-Sign-On (SSO) systems usage has been on the rise exponentially. One of the major benefits of having an SSO system is to have a central authentication service, which other applications can use. However, SSO services are also prone to failure. If an SSO service becomes unavailable due to failure, every application that uses the SSO service become simultaneously inaccessible to users. The goal of this research is to explore a technique to mitigate the availability issue of the SSO by customizing its functionality, and distributing its data using blockchain technology over the network. The Blockchain data structure possesses inherent properties that can be useful to improve an SSO service's availability and hence, its overall functionality and reliability.

CHAPTER 1

INTRODUCTION

Applications are programs that help users to store, retrieve, and analyze data. As organizations grow bigger and expand their scope of work, they need more applications to drive their daily operations [35]. These applications give users access to information (private and public), such as financial data, customers' data, etc. Therefore, access to different applications has to be limited respectively to only those authorized to access them [33]. Securing applications require an authentication mechanism that verifies the user's identity before letting the user access these applications. Authentication can take different forms, such as basic authentication, biometric authentication, etc. The most common method of authentication is the basic one, where developers usually add a login page that challenges users to provide a valid username and password [4, 10, 37]. This login page can be attached to a developed resource to protect it. But, as the number of applications increases, securing these applications will become a daunting task, not to mention maintaining the authentication code in each of them. To alleviate this problem, developers started to think of building a centralized solution that all applications could use [5], which led to the development of the widely used SSO system [9, 13, 26, 42].

SSO brings many benefits to users. One of the major benefits is that the user will only need to login once. After a successful sign-on in the SSO's login page, the user will have access to all applications connected to that SSO service and without the need to log in again. In the same manner, when the user logs out from any resource, he/she will also generally have the option to log out from all other

applications via the SSO system.

Many organizations already have adopted the SSO system in their network. SSO makes it easy for developers to secure many applications, and allows users to memorize and use a single pair of username and password to access multiple applications instead of overwhelming them with several complex username/password pairs. Besides, maintaining a single authentication application is more manageable than multiple ones. The benefits SSO systems encourage organizations not yet using them to move towards using SSO systems. A major drawback in SSO systems is they have a one point of failure [11]. This vulnerability can have major adverse effects on organizations if the SSO stops working. If the SSO stops working, users will not be able to access any of the secured applications that the SSO manages. To the best of the author's knowledge, little or no research in the literature has reported or proposed a solution to this issue, which was one of the primary motivations of this research.

Blockchain is a data structure designed and implemented in 2008 by an anonymous person or a group under the name Satoshi Nakamoto [40]. The purpose of blockchain is to decentralize data and replicate them across different servers on the network. As will be discussed later in detail, there are many implementations of blockchain with the same goal. This research used a blockchain application called Hyperledger - Fabric [2, 7]. Hyperledger - Fabric stores the data in a special data stores called ledgers that are spread out on the network [39]. Blockchain stores data in blocks, and each block contains various pieces of information (data) depending on the specific application. Also, each block stores a cryptographic hash of that block and a cryptographic hash of the previous block, forming a chain of blocks, hence the name blockchain. Because of this strong link between the blockchain blocks, blockchain's datastores are resistant to modification, a feature that made blockchain very popular and attractive. The techniques blockchain uses

to decentralize data and store data, might help address the SSO one-point failure vulnerability identified previously. The contributions of the research in this thesis are: (1) To explore blockchain based techniques to improve availability of SSO systems, (2) Implement the proposed solution over two protocols, namely round-robin and random, to balance the load of the authentication requests between multiple servers.

The thesis is organized as follows. This first chapter (Chapter 1) provides an introduction to the Singe-Sign-On system and Blockchain technology, the benefits of using them and the proposed solution. Chapter 2 provides additional details about blockchain components, applications, and configuration that the experiment model used. Chapter 3 describes the implementation of the proposed solution and also the test functions. Chapter 4 presents the results that have been collected from the experiment and provides an analysis of the results. Finally, Chapter 5 states the conclusion from this work and presents observations that could expand the scope of this work for future research.

1.1 Background

Securing data and applications that access these data have become a challenging task. Attackers are becoming smarter every day, not to mention the advanced tools they use that allow them to scan and attack systems. Unfortunately, applications that are meant to help administrators to test the security of their servers and networks can also also be used by attackers to exploit systems. These applications are usually available on the Internet and any anyone can download and use them. SSO can help improve the security of the system in two aspects. First, it will provide a single authentication point that makes the maintainability of the data and the code easier on administrators and developers. Second, it prevents storing passwords in more than one place, or places external to the organization's

network [41].

Previous studies discussed the concept and the design of the SSO service. For example, [5] and [21] talked in detail about the SSO protocol and the types of applications it can support, such as Web SSO, Legacy SSO, and Federated SSO. It also discussed the encryption techniques that could be implemented over an SSO to protect data in communication against eavesdroppers who might be listening to the communication. Besides, the paper [5] also talks about protocols that the SSO could use, such as LDAP (Lightweight Directory Access Protocol), Kerberos, RADIUS protocol, etc. for user authentication. With the growing popularity and convenience of using cloud computing as a service in organizations, the paper [27] suggests making the SSO a cloud service, and implementing the resulting centralized user authentication virtually. Only a few papers (e.g. [5, 21]) talk about some of the issues with the SSO. As mentioned before, being a centralized service, SSO has a single point of failure vulnerability. Therefore, if the server goes down, no user can log in and use any of the resources on the network.

This research investigates the availability issue of the SSO and explores a solution that can minimize the adverse impacts of this issue. Throughout this research work, a new model will be built that implements the SSO and the suggested solution, which then it will be tested and compared with the current used model. Some of the questions that this research tries to answer are: (1) What are the possible solutions to single-point failures in SSO? (2) Can blockchains be used as part of the solution? (3) If yes, then how? (4) Would using blockchains add complexity and/or latency to the model compared to other solutions? (5) If yes, then by how much?

1.2 Objectives of the SSO

The invention of the Single-Sign-On (SSO) service brought a lot of benefits that helped developers resolve different issues that they experienced before. The SSO got rid of code replication, since it provided a central authentication service that all applications could use. Figure 1.1 shows the concept of the SSO in a network.

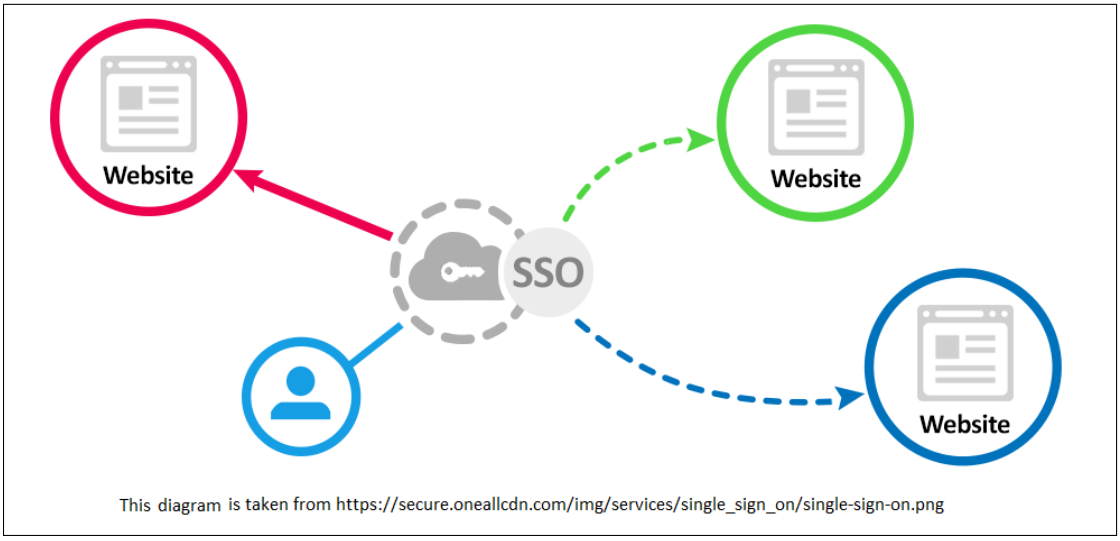

Figure 1.1: Single Sign On Concept

The benefits of using SSO as described in [41] are:

1. No need to store credentials and manage them externally when using a 3rd party application.

2. SSO will liberate users from memorizing a complex password for each application they access.

3. Users will have access to other applications once they authenticate successfully.

4. SSO will minimize the time spent on maintaining the authentication code and data.

1.3 The Proposed Solution to Single-Point Failure in SSO

SSO is a useful authentication system, yet and as explained in the previous section, it has the vulnerability of a one-point failure. Therefore, if the SSO system stopped working because of any reason, users will not be able to log in and/or access any of the secured applications accessible via the SSO system. An intuitive way to address this issue is to run multiple SSO systems any of which can authenticate users, and let them access the secured applications on the network. Although this might work, it might not be the optimal solution to address the problem. That is because, with this solution, all SSO systems share the same database containing user information. Therefore, the failure of the database engine also collapses the whole system. Additionally, this solution might introduce latency if the database and the SSO are located physically in different geographical locations. A better solution will be to distribute and replicate the database across the network too. In this research the usage of blockchain to replicate the database has been explored. Blockchain aims to distribute data across multiple servers in the same or different locations. The proposed solution to the one-point failure in SSO links SSO with blockchain. Although usage of blockchain to replicate databases in SSOs might add complexity to the overall system, it is still worthwhile finding out the amount of complexity in terms of different parameters like latency, memory usage, energy consumption, etc. This research implements the SSO system using blockchain for data distribution and replication and compares it with the current state-of-art SSO systems that use traditional Relation Based databases (RDBMS) to store data and Structured Query Language (SQL) to retrieve data in terms of scalability, maintainability, and vulnerability.

CHAPTER 2

Blockchain Technology

2.1 Introduction to Blockchain

A blockchain is a linked list of records constructed in blocks in which blocks are linked together cryptographically via a hash function [39]. Blockchain stores data using key-value pairs. In blockchains, the record's value does not change. Instead, a new record (block) is appended to the blockchain whenever existing data has to be modified. This behavior can be useful for checking the history of updates on a piece of data contained in the blockchain. Each block in the blockchain contains a cryptographic hash value of the data in that block that depends on the cryptographic hash of the previous block, making data in blocks almost impossible to be altered. One has to modify the hash of all the blocks following a particular block in a blockchain if data of this block has to be modified. These blocks are stored in multiple locations (called ledgers in blockchain terminology) to form a distributed ledgers network. Before adding any block to the ledger, the request goes through different processes, depending on the application. The processes help to make sure that the request is valid, and the blockchain nodes are in consensus with that request before processing it. Blockchain nodes check the identity of the requester by checking the requester's digital signature attached to the request. Once blockchain's initial validation completes, the consensus process starts. The consensus protocol may again differ, depending on the blockchain application [45]. Bitcoin, for example, uses Proof of Work (PoW) protocol [18], while Ethereum uses

Proof of Stake (PoS) [14] consensus protocol. More sophisticated consensus algorithms such as Practical Byzantine Fault Tolerance (PBFT) [1] are also available in certain blockchain applications. Some blockchain applications offer more than one consensus protocol. Each network environment may need different blockchain applications depending on the intended use and the consensus protocol desired. Therefore, the decision-maker will have to examine the network and study the advantages and disadvantages of each consensus protocol to decide on blockchain application that will fit the environment the best.

Blockchain can be categorized into two categories: permissionless and permissioned applications [8, 22, 43, 44]. Permissionless blockchain networks are open for anyone to participate in using the blockchain's functionalities by making the participant's server(s) be a part of the blockchain network. Common examples of permisionless blockchains are Bitcoin and Ethereum [38]. On the other hand, only agreed upon participants can be a part of the blockchain network in permissioned blockchain applications. This agreement has to be decided on before setting up the blockchain network. An example of permissioned blockchain application is Hyperledger Fabric [2], which is what this research uses.

2.2 Blockchain Applications

As mentioned in the previous section, in the permissionless blockchain, anyone can participate in the blockchain network. The participation happens at the consensus stage and will differ by the consensus algorithm adopted by the blockchain application. For example, in PoW (Proof of Work) protocol, the blockchain network asks the participants to solve a cryptographic puzzle. Participant servers will race each other to solve this puzzle. The incentive for these participants on the race is that the first participant who solves the problem will win a reward from the blockchain system. This process is called mining in blockchain, and the miners are

nodes/servers that solve the puzzle [23]. PoS (Proof of Stake), on the other hand, realizes the power consumption issue in solving a complex mathematical problem when using PoW. Instead, PoS motivates participants to buy cryptocurrency to become a more competitive stakeholder and validator.

In permissioned blockchain system, participants are already known by the blockchain network and agreed upon at the early stages of building the network. Communications in permissioned blockchain networks are strict. Therefore, participants that send a message will have to stamp it with their digital signature. Also, any participant that receives a message will verify the digital stamp of the sender before processing the message. The receiving participant will ignore the message if it fails to verify the signature. In other words, the message will fail the receiving participant's verification if it is coming from a non-participant server, or if its cryptographic signature does not match the sender's signature. All participant servers in this scenario know each other and share the same goal. Because the identities of all participants are known, the risk of having a malicious participant that tries to disrupt the blockchain system is less. Also it is easier to identify the a malicious or compromised node/server in this type of blockchain networks.

The properties of the Hyperledger Fabric, such as its robustness, fit most of the requirements of an SSO system. Also, Hyperledger being a permissioned blockchain application makes it more desirable for storing private information such as users' account information that an SSO system uses. Additionally, Hyperledger Fabric is an open source, permissioned distributed ledger blockchain that addresses certain limitations of other permissioned blockchain systems, e.g. usage of a fixed consensus protocol, and not having flexibility in the programming language used for implementation [3]. It was created under the Linux Foundation, and is intended to be used in enterprise systems. Also, Hyperledger Fabric was designed to be modular and configurable, a feature that fits in nicely for SSO systems.

2.3 Hyperledger Fabric Componenets

The main components of Hyperledger Fabric (or Fabric) include ledger(s), peer(s), smart contracts, and orderer(s). Fabric modularizes the network by using containers, and therefore, Fabric has Orderer(s) and Peer(s) in separate docker containers. Ledgers are Fabric's immutable storage units that do not only store the current record's value but also store the transaction history for each record. Each record is stored in the form of blocks, and the data in the blocks are in key-value pairs. Because each block references the previous block's cryptographical hash, it is almost impossible to alter any record. Ledgers live in peers, and each peer can have one or more ledgers. Peers are vital and fundamental components in the Fabric network because they host ledgers and smart contracts, in addition to their role in endorsing and validating new requests.

Smart contracts (also known as chaincodes) are applications that define the policies among the participating entities on the blockchain network [34]. Fabric supports different languages that can be used to develop the chaincode, such as Golang, Java, and Nodejs. Chaincode applications set the behaviors of the blockchain network, such as creating a new record, updating records, deleting records, etc. in addition to maintaining the requirements to initialize any request. Orderers, on the other hand, plays a critical role in collecting and ordering new blocks before distributing them to the peer nodes. Indeed, in Fabric, no single commit to the ledger can be done without orderer's involvement in the process. Generally, if the network receives a request to add a record to the ledger, this request will go through a process. This process comprises of three phases: proposal phase, ordering and packaging transactions into blocks phase, and validation and commits stage. In the proposal phase, the application (which is not part of the blockchain network, but it interfaces with it) will issue a transaction request to

each of the required peers for endorsements. The endorsing peers will check the received request and digitally sign it, yet they do not update their ledgers at this phase. Once the peer approves the request, it returns to the application, and that will mark the completion of the first phase. In the second phase, the system will package and send the signed transactions it received from the endorsed peers to the orderer, which is the pivotal component in the update process. The third phase of the process involves the orderer to distribute the signed request to all peers that will validate the claim and ensure the appropriate organization has endorsed the request before committing the update to the ledger. Because Fabric is a permissioned blockchain, each participating peer must have a cryptographic key that it can use to sign requests. Other peers will check the signature of these requests to verify who worked on these requests. If the verification of any message points to a foreign sender, the peer will ignore these requests. The Fabric also implements different permissions within one network by introducing channels to the system. For peers to communicate with each other, they need to be on the same channel. Otherwise, they are not authorized to talk. Peers can join organizations. Depending on the endorsement policy, peers from different organizations might have to endorse a transaction to make sure all organizations are in agreement with any transaction that occurs in the blockchain network.

2.4 Initial Setup for this Project

The initial research was to build a working Fabric network following the tutorials that [3] provided in Building Your First Network and in the Fabcar tutorial. In the examples illustrated, two organizations have been configured with two peers in each of them. A channel that connects all peers and one orderer has been set up in a single machine. A sample chaincode has also been provided to demonstrate the functionality of the blockchain. Another Fabric componenet (cli) has been installed

to be used to communicate with peers and the orderer. Each blockchain component (Peers, orderer, and the cli) resided in a docker container. These components could communicate with each other, although each resided in separate docker containers.

Major non-trivial modification on the above setup has been carried out to fit it in the current research model of this thesis. In the above setup, the entire process, including initiating the ledger, adding, updating, or querying it, was carried out manually. This requires executing numerous commands in the cli bash terminal, for example to pull data from the ledger or write to the ledger. A further issue with the setup model is that it uses a single machine to set up all blockchain components, which does not reflect a real-world scenario. Thus, an important modification of the existing setup carried out in the research was the separation of each Fabric component into its own server achieved without disrupting the communication between them in the network.

2.4.1 Blockchain Setup Details

The blockchain network in the new model that was created for experiments contained an orderer on a machine and two peers, each residing on a Linux server. The chaincode provided in [6] has been modified and extended in functionality to add update and delete methods. Also, the method's signatures of the chaincode have been modified to better fit the experiments' requirement. The experiments required an application to act as middleware between the blockchain network and applications external to the blockchain network. The implemented application exposes APIs, enabling other applications to interact with the blockchain through these APIs. Fabric's SDK library has been used to build this application. The Fabric SDK supports two programming languages: Java and NodeJS, and future versions are likely to support more programming languages. The NodeJS SDK has been used to develop the APIs for this research. Figure 2.1 summarizes the

blockchain network used in the experiment.

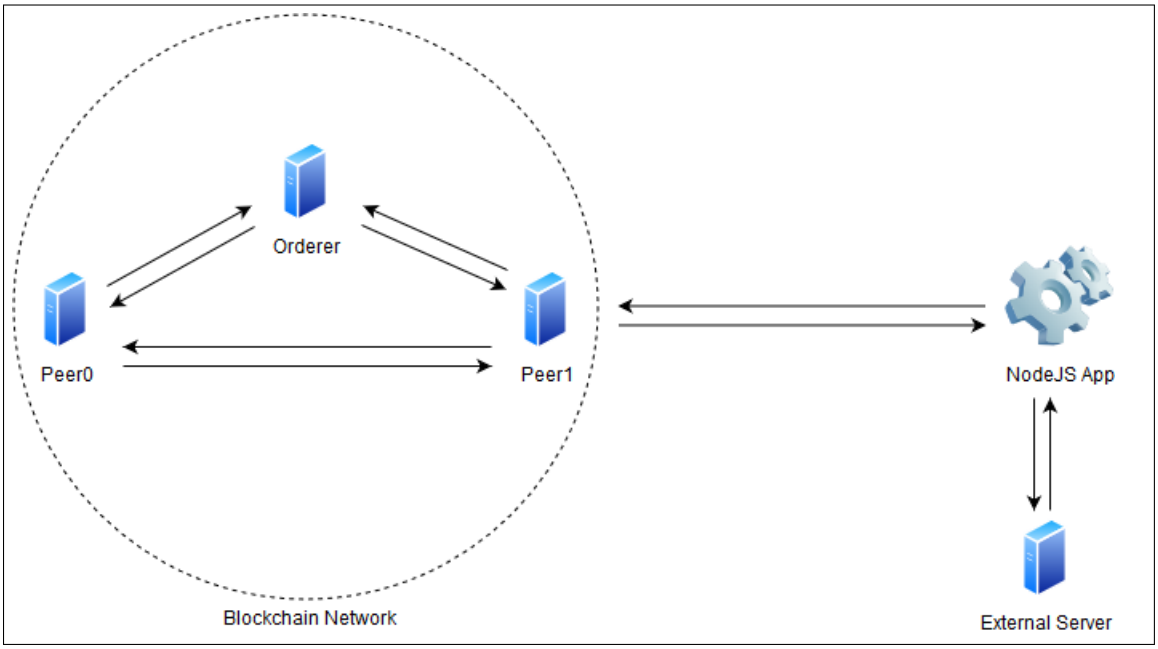

Figure 2.1: Implemented Blockchain Model

In this setup, an organization called org1 has been created having two peers: Peer0 and Peer1. Also, a single channel called mychannel was created and joined to Peer0 and Peer1. One orderer that used the Solo consensus protocol has been installed for simplicity. To set up the blockchain system, we used a Fabric configuration file called configtx.yaml. We also used configtxgen tool, which came with Hyperledger Fabric, to generate the cryptographic keys for each Fabric's component. This tool receives the necessary configuration information from a file called crypt-config.yaml to create necessary cryptographic keys.

The new blockchain network model described above allowed isolating each blockchain component from the rest by hosting it in a separate server. This isolation helped place blockchain nodes in different geolocation maintaining functionality at the same time. Moving nodes to different locations will improve the distrubitiveness and hence availability of the service. A query can potentially be

directed to any peer in the network based on efficiency, improving overall network performance. Furthermore, having blockchain APIs on the network, enables other applications to interface with the blockchain network in an automated fashion, without the need for any manual labor to query or update the ledgers.

2.5 Implementation of Final Model

To be able to isolate each blockchain component and install it in a separate server, network connectivity should be maintained among the hosting servers. All servers were put on the same networking subnet in order to achieve that. The servers had following IP addresses:

- The server that hosts the Orderer: 192.168.1.185
- The server that hosts Peer0: 192.168.1.106
- The server that hosts Peer1: 192.168.1.126

As mentioned before, Fabric was built using docker containers. Each container has a Fabric component installed in it, and each docker container uses a network of its own but communicates with the hosting network. For the experiment, Fabric components the following domains with the respective IP addresses have been assigned. The docker configuration file docker-compose.yml is where the container's network can be set. The following settings for domains were set the file (/etc/hosts) of the hosting server:

- Orderer: 192.168.16.2 orderer.example.com
- Peer0: 192.168.17.3 peer0.org1.example.com
- Peer1: 192.168.18.4 peer1.org1.example.com

To enable the communication between Fabric components, we also added routes on each server, such that, the hosting operating system will know where to route the network packets to. These new routes were as follow:

On Orderer hosting server:

ip route add 192.168.17.0/24 via 192.168.1.106 dev enp0s3

ip route add 192.168.18.0/24 via 192.168.1.126 dev enp0s3

On Peer0 hosting server:

ip route add 192.168.18.0/24 via 192.168.1.126 dev enp0s3

ip route add 192.168.16.0/24 via 192.168.1.185 dev enp0s3

on Peer1 hosting server:

ip route add 192.168.17.0/24 via 192.168.1.106 dev enp0s3

ip route add 192.168.16.0/24 via 192.168.1.185 dev enp0s3

We used the Orderer hosting server to create the initial blockchain setup. That is, defining blockchain's component(s), organization(s), domain(s) and the channel(s) using "configtx.yaml" file that Fabric tool "configtxgen" will consume when we run that tool. Running "configtxgen" tool will create two files: channel.tx and genesis.block. These two files play an important role as we will use them when creating the channel and when joining the peers to the channel in the network. We also used the Orderer hosting server to generate the necessary cryptographic keys using "crypt-config.yaml" config file and "configtxgen" tool. For this experiment, the blockchain is set up not to encrypt any data transferred through the line between blockchain components. We also did not use any encryption on the application that uses the Fabric SDK to expose the blockchain APIs. We programmed the APIs to either read from blockchain, write to it, update it, or delete from it. Of course, updating or deleting any record in blockchain means to add another record that reflects the new value to the key in the ledger. The URLs for these

APIs are:

peer ip address:8080/write?req=update&name=VALUE1&value=VALUE2 - for writing/update to blockchain

peer ip address:8080/write/?req=create&name=VALUE1&value=VALUE2 - for writing/create to blockchain

peer ip address:8080/write/?req=delete&name=VALUE1 - for writing/delete from blockchain

peer ip address:8080//get?req=VALUE1 - for reading from blockchain,

where VALUE1/VALUE2 is the key/value pair for each record.

Now that we have installed and configured the blockchain application and set up its network, we are ready to move on to the next step and start working on the SSO application. Hence, in the next chapter, we will be talking about the implementation of the SSO concept and how it communicates and use the blockchain network. Furthermore, we will discuss the mechanism that we implemented and investigate the performance of the different protocol options that we have developed.

CHAPTER 3

Implementation Details

3.1 Requirements for the Model

Two options were considered to build the SSO model. The first one was to use an opensource application and customize it to make it fit the research needs. The second option was to develop the concept from scratch. While both options were feasible, since building the SSO model from scratch makes customizing it easier, this option was selected. Also, implementing the SSO made it easier to plug it into the blockchain network in addition to the traditional database storage. When developing the SSO, several testing functions have been added to measure different properties of the system. Appending these functions to a previously developed SSO application, would have been a challenge.

The developed SSO model and system provides following features that any traditional SSO application offers:

1. Single login web application.
2. A user can access any resource that uses the SSO service upon login.
3. A user is logged out from all resources upon logout of the SSO service.

Additionally, since this model supports multiple SSO servers, the following requirements were added:

1. Almost identical login web page.

2. A user logged in from either servers will have access to any resource that uses the SSO service.

3. Before redirecting the user to an SSO server, the model checks the availability of the server first. If none of the servers are available, it shows an error message, asking the user to try again later.

Additionally, a few APIs were published for use on the new model. For example an API used for logging in and out a test user. The experiment used the automated authentication process to examine the new model's performance when using blockchain and when using the traditional (SQL) database.

3.2 Concept Implementation

The new SSO model comprises of two main servers: 1) Identity Provider (Idp) server and 2) the Decision-Making (DM) server. Usually, with traditional SSO applications, there are two main servers: Idp server and the Service Provider (SP) server [12]. Idp server is similar in functionality on both models: the current model and the traditional one. Idp's main goal is to provide the user with a login page, authenticate the user then redirect them back to the resource that the user was trying to access. On the other hand, the DM server is similar in functionality to the SP server. However, more functions have been added to the DM server on top of an SP server, to fit it with the experiment needs. The DM server will be responsible for receiving the initial request from the resource, and checks Idp's availability and redirect the user accordingly. The DM's role is vital to the whole system as it plays as the mediator between the resource and the Idp. Therefore, all messages that originate from the applications will run through the DM to the designated Idp. The applications which are using the SSO will send their url to the DM server, which in turn send the url to the Idp. The Idp will use the received url

to redirect the user back to the application that the user was trying to access. The application will attach its url to other servers using a query string parameter via `GET` method. However, checking the availability of the Idps is handled by querying the Idp status using the `POST` method. Checking the availability of the Idp will also check the availability of the blockchain network too.

The Idp, on the other hand, exposes other APIs that serve other purposes. These APIs are built to implement the test environment. To make the experiment simple, no security concept has been applied when developing these APIs. Therefore, servers traverse messages in plain text, with no message integrity validation nor encryption. Depending on what the user chooses, the Idp will either interact with the blockchain network or the database. The DM server will check which Idp is available first, and then it will check the availability of the storage method that the user has selected (blockchain or database). Depending on the outcome of the check, the application will redirect the user to an Idp and at the same time ask the Idp to use the available storage method. A checkbox has been added to the login page to make the user be able to enforce the Idp to use either storage method. The communication to the database or blockchain is seamless to the log in process and the user. If the login page has a ticked checkbox, then the Idp uses the database. Otherwise, The Idp will use blockchain instead. Fig. 3.1) provides a screenshot of the login interface.

3.3 Test Model

After implementing the main concept of the SSO and connecting it to the blockchain and the database, the need was to develop another function that tests the performance when using either storage method (blockchain vs. traditional database). The requirement for this function is that it must be able to calculate (as accurately as possible) the time taken to log a user in. Also, it must be able to receive

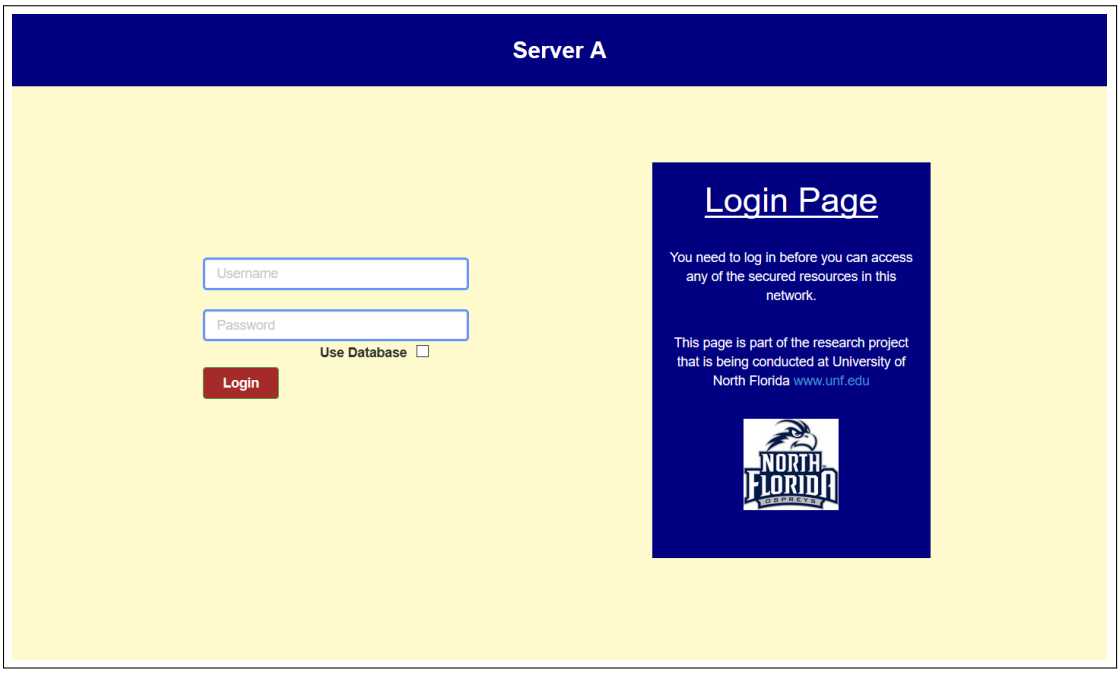

Figure 3.1: Single Sign On Login Page

and information about the Idp's CPU usage and memory usage in each login cycle. Round Robin and Random distribution protocols have been used to distribute the load between the two Idps in the test model. In Round Robin, the requests alternate between the first Idp and the second one, while Random forwards requests to an Idp randomly. As shown in Fig. 3.2, storage method selection and number of iterations that the application will log the test user in and out, have been added. Each time the application logs a user in, it collects information about the time taken by the application to visit the necessary pages back and forth. It also checks which server was used, the CPU and the memory usage respectively for that server. The test application then stores gathered information in a database for later processing. Once the application completes all requested iterations, it shows results at the bottom. The results displayed for time taken, CPU and memory usage are the average of all collected data from all iterations. While the numbers below Server A and Server B fields indicate how many times each server has served the request for each test cycle. Fig. 3.2 summarizes this mechanism.

The applications that pertain to the SSO implementation and testing, were developed using ASP .NET, using .NET framework libraries version 4.6.1. On the other hand, all web sites were developed using Microsoft Internet Information Services (IIS) version 10. The database engine used was Microsoft SQL Server 2017. The experiment used two machines to implement the blockchain and the SSO concept. Each of these machines has IIS, and MS SQL servers installed. The next section will talk about the details of the hardware of each machine and what each machine hosts.

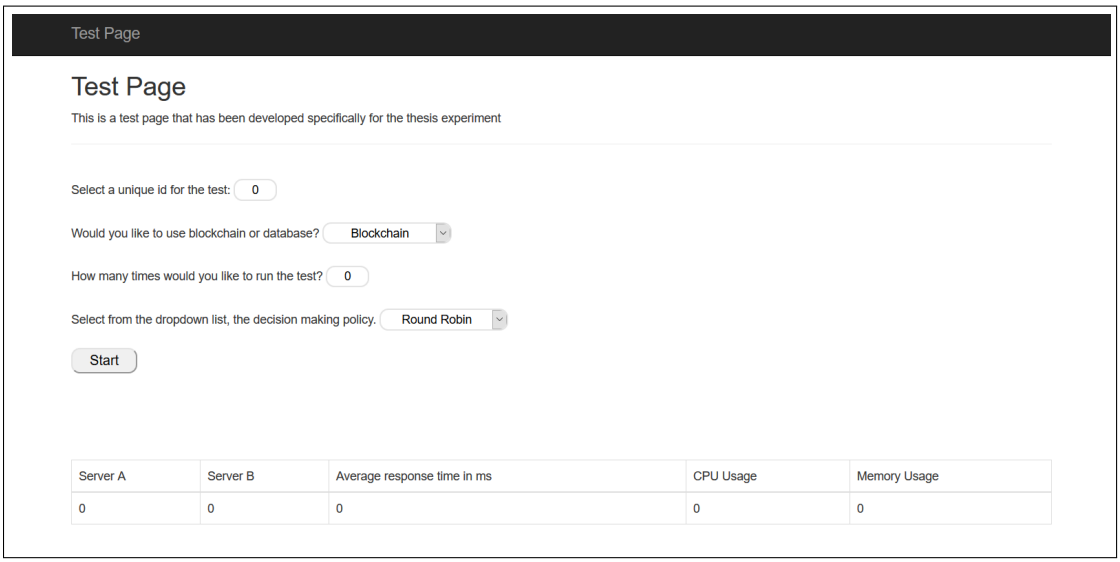

Figure 3.2: Implementation of the Test Page

3.4 Hardware Used

Two machines were used to implement and run the experiment. The first machine has the following specification:

- Type: Desktop
- Model: Dell Optiplex 760
- CPU: Intel (R) Core (TM)2 Duo CPU E8400 @ 3.00 GHz

- Memory: 4GB
- Network card: Intel(R) 82567LM-3 Gigabit Network Connection

The other machine's specification is as follow:

- Type: Laptop
- Model: Dell G3 15
- CPU: Intel(R) Core(TM) i5-8300H CPU @ 2.30 GHz
- Memory: 8 GB
- Network card: Realtek PCIe GbE Family Controller

Three virtual machines were used, which were installed in the desktop machine (Dell Optiplex 760). All virtual machines have the same the specification, and they are as follow:

- Type: Virtual Machine
- Model: n/a
- Virtual CPU: 1 CPU with an execution cap of 50% of the host's CPU
- Virtual Memory: 500 MB
- Virtual Network card: Intel (R) 82567LM-3 Gigabit Network Connection

3.5 Operating System and Software Used

The laptop machine has Microsoft Windows 10 version 1803. The Desktop machine has Microsoft Windows 10 operating system. The desktop computer has a virtual machine application (virtualbox) installed on it, that runs three Linux

server - Debian distribution, that host the blockchain components. The first virtual machine is the Orderer, the second is the first peer (Peer0), and the third is the second peer (Peer1).
Each Linux server has the following software:

- Nodejs version 8.9.4
- Docker and Docker CE version 18.09.3
- Blockchain application version 1.4
- npm version 5.6.0
- Go version 1.11.5

Each Windows operating system has the following software:

- Internet Information System
- Microsoft SQL Server 2017
- Web Browser
- .Net Framework library

3.6 Putting All Pieces Together

Multiple applications were needed to run the experiment. The first two applications are the resources that the user will try to access. The following domain names were assigned to them in the IIS server:

- `a.resource.local`
- `b.resource.local`

The other two applications were the login applications. The following domain names were assigned:

- `a.idp.local`
- `b.idp.local`

Another two applications have been developed to control the redirection of the authentication requests. The domain names for theses are:

- `a.dm.local`
- `b.dm.local`

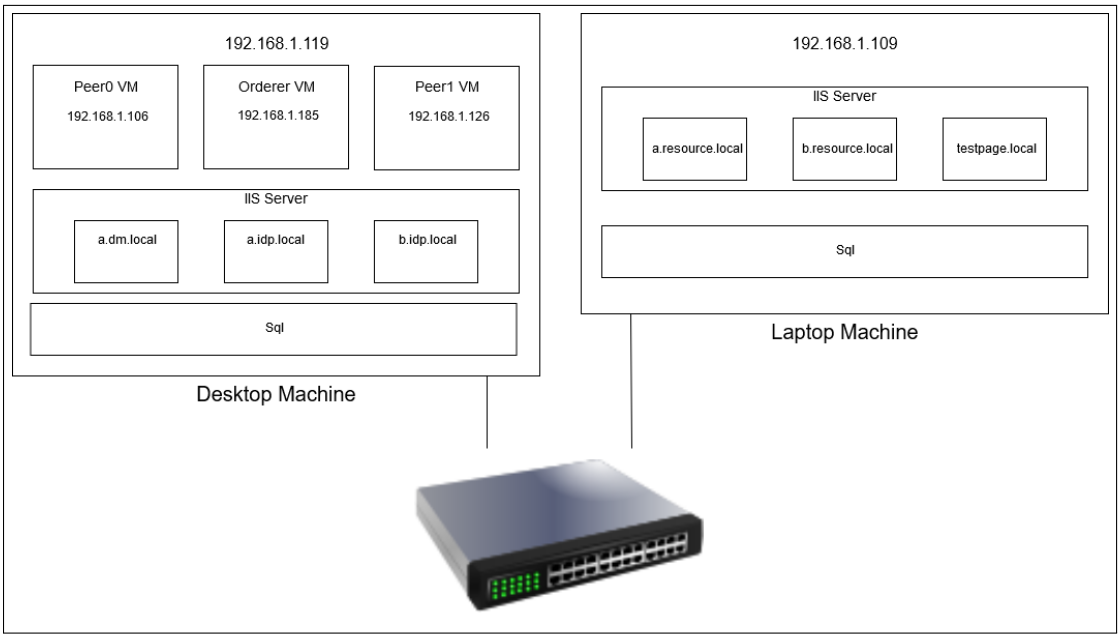

Figure 3.3: How the hardware are connected

Lastly, and to be able to test the whole login cycle and check the system performance, another application has been developed and the following domain was given to it:

- `testpage.local`

The IIS that hosts the resource applications, as well as the test page is installed on the laptop computer and has the IP address of `192.168.1.109`. The rest of the applications are hosted on the Desktop machine's IIS with an IP address of `192.168.1.119`. Fig. 3.3 shows how servers are interconnected with each other. This connection setup serves simplicity and enhances availability. In production networks, each vital component of this network could be installed in different hardware that could be placed in different locations for better availability.

CHAPTER 4

Results and Analysis

4.1 Results

Two systems have been compared in the experiments. An SSO whose database has been built over a blockchain, and an SSO with a traditional relational (SQL) database. Additionally the latter case has been further subdivided into having a single SSO controlling (traditional) the SQL database versus two SSO's controlling the SQL database (non-traditional). The test model has been made to record two seconds of response time to reflect failure when the SSO service is unreachable.

To summarize the results before detailed analysis in the next section, Tables 4.1, 4.2, and 4.3 can be compared. Higher response time of the blockchain based system (Table 4.1) over the SQL database based system with two SSO's (non-traditional, Table 4.3) is noted. However, when using only one SSO, the SQL database based system records considerable higher response times (traditional, Table 4.2) than the blockchain based system. A similar analysis can be done with Tables 4.4, 4.5 and 4.6. This signifies, blockchain add latency (overhead), but make the SSO much resistant against failures and thus mitigates the one-point failure problem.

4.2 Analysis

From the three result sets, one can right away say that the best-gotten result set was the non-traditional database network. The average performance, CPU usage, and memory usage were: 157.575ms, 2.37%, and 83.54%, respectively, when using the

Server A	Server B	Avg Response Time (ms)	CPU Usage %	Memory Usage %
50	50	306.57	2.030681506	83.5248911
50	50	291.69	1.830287921	83.5652267
50	50	292.15	2.014770745	83.6936236
50	50	282.92	2.243154846	83.8098388
50	50	332.2	2.025063038	84.0343197
50	50	340.51	1.917625142	83.6562479
Average		307.6733333	2.010263866	83.71402463

Table 4.1: Using Blockchain with 100 iterations and Round Robin decision making policy

Server A	Server B	Avg Response Time (ms)	CPU Usage %	Memory Usage %
50	50	1087.21	61.9286201	92.6003759
50	50	1091.54	62.3675979	92.7597692
50	50	1085.31	63.3199594	92.7486049
50	50	1086.87	60.69951788	92.4283692
50	50	1083.07	52.48020791	91.283737
50	50	1088.87	51.29524499	91.3197853
Average		1087.145	58.68185803	92.19010692

Table 4.2: Using Database with 100 iterations and Round Robin decision making policy - traditional connection

Server A	Server B	Avg Response Time (ms)	CPU Usage %	Memory Usage %
50	50	163.35	2.567436978	83.5435908
50	50	154.31	2.41905589	83.5693876
50	50	163.33	1.758625627	83.5748293
50	50	158.78	2.566913101	83.5276352
50	50	155.9	2.212149397	83.4983903
50	50	149.78	2.711715433	83.5528293
Average		157.575	2.372649404	83.54444375

Table 4.3: Using Database with 100 iterations and Round Robin decision making policy - Non-traditional connection

Server A	Server B	Avg Response Time (ms)	CPU Usage %	Memory Usage %
56	44	350.43	5.459260341	83.5454013
57	43	315.8	1.972614039	82.9586011
57	43	312.31	1.954636692	82.9663439
37	63	352.33	1.912760897	83.0211267
50	50	310.1	1.870752326	83.0373443
37	63	314.36	2.222285752	83.0883875
Average		325.8883333	2.565385008	83.10286747

Table 4.4: Using Blockchain with 100 iterations and Random decision making policy

Server A	Server B	Avg Response Time (ms)	CPU Usage %	Memory Usage %
48	52	1126.12	52.96215045	91.6951103
51	49	1073.69	51.23559325	91.2148014
60	40	894.28	41.36450123	89.6292268
52	48	1047.08	49.50005645	91.0233381
47	53	1136.39	54.45525725	91.9794858
53	47	1022.82	48.37765522	90.9149873
Average		1050.063333	49.64920231	91.07615828

Table 4.5: Using Database with 100 iterations and Random decision making policy - traditional connection

Server A	Server B	Avg Response Time (ms)	CPU Usage %	Memory Usage %
51	49	154.93	2.777926719	83.10634
46	54	159.52	2.844795191	83.0365255
48	52	152.44	2.936936071	82.9861402
50	50	150.3	2.856918992	83.0430471
57	43	155.68	2.73933416	83.0164671
46	54	154.36	2.620824055	82.9637084
Average		154.5383333	2.796122531	83.02537138

Table 4.6: Using Database with 100 iterations and Random decision making policy - Non-traditional connection

round-robin protocol. The average performance, CPU usage, and memory usage were: 154.53ms, 2.79%, 83.02%, respectively, when using the random protocol. This model is quite unrealistic, and the reason for that is because this model does not provide a failover solution to the database. Also, in this scenario, if the two SSO servers are far away from each other, the system will introduce latency and delay, based on the location of the database to each server.

The blockchain was the second-best model in performance. This model result's average for performance, CPU usage, and memory was: 307.67ms, 2.01%, and 83.71%, respectively, when using the round-robin protocol, while it was 325.88ms, 2.56%, and 83.10% respectively when using the random protocol.

It was no surprise that the performance for the model that used the database in a traditional connection was the worst. In this layout, the database was interfacing with only one server, and hence, users were not able to use the second server. To show this failure in our experiment, the SSO network was made to return (2) seconds delay, 100% CPU usage, and 100% memory usage each time the application tried to use the second server. Therefore, the average results for this model's performance, CPU usage, and memory usage were: 1087.14ms, 58.68%, and 92.19%, respectively, when using the round-robin protocol. Using random protocol, the performance, CPU usage, and memory usage for the model were: 1050.06ms, 49.64%, and 91.07%, respectively.

As can be noticed from the results, using blockchain network added overhead to the process, which is what the time response and the memory usage indicated. The model with blockchain was slower in response and higher in memory usage compared to the non-traditional database model. Surprisingly, the CPU usage was less in the blockchain model than the non-traditional one. Blockchain model needed about 2.01% of the available CPU power while the database model needed 2.37% when using Round Robin protocol. One can see the same observation when

looking at the results of the two models that use the random protocol. It's worth mentioning that the memory and CPU usage stats were for the desktop machine that hosts the blockchain network as well as the SQL server.

CHAPTER 5

Conclusion and future work

5.1 Conclusion

Throughout the experiment, it has been noticed that using a non-traditional database network was the fastest and the least resource-demanding compared to other models. However, if the database server malfunctioned or disconnected from the rest of the system, the solution will fall apart. Therefore, this model has one critical point of failure, which makes it unrealistic in practical implementations. Also, attacks to the database servers are becoming popular, and attacking web applications using SQL injections increased significantly in the last decade, which is another drawback of using databases.

SQL injection attack is one of the most dangerous web application attacks and the most common and notorious attack on the databases [24]. This attack aims to steal private information or delete data from the database by exploiting the vulnerabilities in a web application. These vulnerabilities are associated with improper validation and sanitization to the data received from untrusted input. Web application attacks, including SQL injection attacks, were at the top of the list by many organizations such as WASP, MITRE, etc. The attacker will try to insert SQL statements as input to the page's input fields, hoping that the application will include them in its SQL queries. A successful attack can give the attacker access to the application's database and allow the attacker to alter the stored data [19]. Using databases comes with the risk of getting such attacks in

addition to other attacks as illustrated in [15, 16, 17, 20, 25, 28, 30, 31, 29, 32, 36].

On the other hand, the blockchain network offered benefits that the database did not provide. Security is a big concern when using databases. A simple programming issue could lead to a catastrophic problem, giving the attacker the leverage to alter data. As we discussed above, SQL Injection is the most common and notorious attack against web applications. Even after many years of discovering this exploit, there are still many web applications that are vulnerable to this attack. Using blockchain will eliminate the SQL Injection exploitation because blockchain uses a different approach to query the datastores. Also, as it was discussed in previous sections, any request to the blockchain will go through many verifications and validations before the blockchain network processes that request. Blockchain can distribute the data in multiple servers that can be in different geolocation, which is a nice feature to have, especially in the availability problem. If database networks are to be modified to implement the same features that blockchain offers, overhead and complexity will be added to the system. Besides, it will be putting lots of effort into implementing features that come by default with blockchain.

Blockchain, however, comes with a caveat, with the same feature that leverages blockchain over the database. This issue is the inability of deleting records. Therefore, attackers could flood the blockchain network with many requests to create new accounts, update accounts, or even reset the account's password. This behavior will make the blockchain storage overgrow, and when blockchain uses all available space, the blockchain functionality will break. The backup process is also not a straight forward process when it comes to blockchain, contrary to its peer, the backup process for the database is quite easy.

In any problem, there will be more than one solution. Each solution will fit a specific situation. However, any solution that will be decided on, there will be things that will be compromised. This research shed some light on a problem and

explored a solution that uses blockchain and compared it with a solution that uses the traditional and non-traditional database networks. The research also provided advantages and disadvantages of using each solution over the other in the hope that developers consider the weaknesses of each solution and try to address them to build a better model.

5.2 Future Work

Using a blockchain network has helped to solve the research problem, but it also introduced security vulnerabilities to the system, as mentioned in the above sections. To improve upon the proposed solution, one can investigate ways to mitigate the security vulnerabilities that were introduced by using blockchain. Also, backing up the blockchain data and archiving its old data are another avenue to explore. Enhancing the model's security and solving the backup and archive problems will help make the use of blockchain more practical in SSO applications and to similar applications to SSO as well. Also, finding an algorithm that speed up the read-/write time with blockchain networks will always be a good subject to study on to bring the blockchain applications to the next level.

REFERENCES

[1] Abraham, I., Gueta, G., Malkhi, D., Alvisi, L., Kotla, R., and Martin, J.-P. Revisiting fast practical byzantine fault tolerance. arXiv preprint arXiv:1712.01367 (2017).

[2] Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In Proceedings of the Thirteenth EuroSys Conference (2018), ACM, p. 30.

[3] Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In Proceedings of the Thirteenth EuroSys Conference (2018), ACM, p. 30.

[4] Bartlow, N. Username and password verification through keystroke dynamics. PhD thesis, West Virginia University, 2005.

[5] Bazaz, T., and Khalique, A. A review on single sign on enabling technologies and protocols. International Journal of Computer Applications 151, 11 (2016).

[6] Biswas, S., and Biswas, S. Password security system with 2-way authentication. In 2017 Third International Conference on Research in Computational

Intelligence and Communication Networks (ICRCICN) (Nov 2017), pp. 349–353.

[7] Cachin, C. Architecture of the hyperledger blockchain fabric. In Workshop on distributed cryptocurrencies and consensus ledgers (2016), vol. 310, p. 4.

[8] Cash, M., and Bassiouni, M. Two-tier permission-ed and permission-less blockchain for secure data sharing. In 2018 IEEE International Conference on Smart Cloud (SmartCloud) (2018), IEEE, pp. 138–144.

[9] Cohen, R. J., Forsberg, R. A., Kallfelz Jr, P. A., Meckstroth, J. R., Pascoe, C. J., and Snow-Weaver, A. L. Coordinating user target logons in a single sign-on (sso) environment, Jan. 23 2001. US Patent 6,178,511.

[10] Craine, D. A. Keyboard with programmable username and password keys and system, Apr. 17 2012. US Patent 8,161,545.

[11] Cusack, B., and Ghazizadeh, E. Evaluating single sign-on security failure in cloud services. Business Horizons 59, 6 (2016), 605–614.

[12] Daniel, K., Tran, T., and Wietfeld, C. Interoperable role-based single sign-on-access to distributed public authority information systems. In 2008 IEEE Conference on Technologies for Homeland Security (2008), IEEE, pp. 327–332.

[13] De Clercq, J. Single sign-on architectures. In International Conference on Infrastructure Security (2002), Springer, pp. 40–58.

[14] Gao, Y., and Nobuhara, H. A proof of stake sharding protocol for scalable blockchains. Proceedings of the Asia-Pacific Advanced Network 44 (2017), 13–16.

[15] Garrett, K., Talluri, S. R., and Roy, S. On vulnerability analysis of several password authentication protocols. Innovations in Systems and Software Engineering 11, 3 (2015), 167–176.

[16] Gouge, J., Seetharam, A., and Roy, S. On the scalability and effectiveness of a cache pollution based dos attack in information centric networks. In 2016 International Conference on Computing, Networking and Communications (ICNC) (2016), IEEE, pp. 1–5.

[17] Harish, P. D., and Roy, S. Energy oriented vulnerability analysis on authentication protocols for cps. In 2014 IEEE International Conference on Distributed Computing in Sensor Systems (2014), IEEE, pp. 367–371.

[18] Jakobsson, M., and Juels, A. Proofs of work and bread pudding protocols. In Secure Information Networks. Springer, 1999, pp. 258–272.

[19] Joshi, P. N., Ravishankar, N., Raju, M., and Ravi, N. C. Encountering sql injection in web applications. In 2018 Second International Conference on Computing Methodologies and Communication (ICCMC) (2018), IEEE, pp. 257–261.

[20] Khatwani, C., and Roy, S. Security analysis of ecc based authentication protocols. In 2015 International conference on computational intelligence and communication networks (CICN) (2015), IEEE, pp. 1167–1172.

[21] Li, B., Ge, S., Wo, T.-y., and Ma, D.-f. Research and implementation of single sign-on mechanism for asp pattern. In International Conference on Grid and Cooperative Computing (2004), Springer, pp. 161–166.

[22] Magyar, G. Blockchain: Solving the privacy and research availability tradeoff for ehr data: A new disruptive technology in health data management.

In 2017 IEEE 30th Neumann Colloquium (NC) (2017), IEEE, pp. 000135–000140.

[23] Mattila, J. The blockchain phenomenon. Berkeley Roundtable of the International Economy (2016).

[24] Mitropoulos, D., Louridas, P., Polychronakis, M., and Keromytis, A. D. Defending against web application attacks: Approaches, challenges and implications. IEEE Transactions on Dependable and Secure Computing 16, 2 (March 2019), 188–203.

[25] Morais, F. J. A., Roy, S., and Ahuja, S. P. A cache based dos attack on real information centric networking system. In 2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC) (2018), IEEE, pp. 1–7.

[26] Pashalidis, A., and Mitchell, C. J. A taxonomy of single sign-on systems. In Australasian Conference on Information Security and Privacy (2003), Springer, pp. 249–264.

[27] Revar, A. G., and Bhavsar, M. D. Securing user authentication using single sign-on in cloud computing. In 2011 Nirma University International Conference on Engineering (Dec 2011), pp. 1–4.

[28] Routh, C., DeCrescenzo, B., and Roy, S. Attacks and vulnerability analysis of e-mail as a password reset point. In 2018 Fourth International Conference on Mobile and Secure Services (MobiSecServ) (2018), IEEE, pp. 1–5.

[29] Roy, S. Denial of service attack on protocols for smart grid communications. In Security solutions and applied cryptography in smart grid communications. IGI Global, 2017, pp. 50–67.

[30] Roy, S., Das, A. K., and Li, Y. Cryptanalysis and security enhancement of an advanced authentication scheme using smart cards, and a key agreement scheme for two-party communication. In 30th IEEE International Performance Computing and Communications Conference (2011), IEEE, pp. 1–7.

[31] Roy, S., and Khatwani, C. Cryptanalysis and improvement of ecc based authentication and key exchanging protocols. Cryptography 1, 1 (2017), 9.

[32] Roy, S., Morais, F. J. A., Salimitari, M., and Chatterjee, M. Cache attacks on blockchain based information centric networks: an experimental evaluation. In Proceedings of the 20th International Conference on Distributed Computing and Networking (2019), ACM, pp. 134–142.

[33] Sandhu, R. S., Coyne, E. J., Feinstein, H. L., and Youman, C. E. Role-based access control models. Computer 29, 2 (1996), 38–47.

[34] Scherer, M. Performance and scalability of blockchain networks and smart contracts, 2017.

[35] SSO Benefits. Sso login: Key benefits and implementation. `https://auth0.com/blog/sso-login-key-benefits-and-implementation/`, 2016.

[36] Talluri, S. R., and Roy, S. Cryptanalysis and security enhancement of two advanced authentication protocols. In Advanced Computing, Networking and Informatics-Volume 2. Springer, 2014, pp. 307–316.

[37] Thongjul, S., and Tritilanunt, S. Analyzing and searching process of internet username and password stored in random access memory (ram). In 2015 12th International Joint Conference on Computer Science and Software Engineering (JCSSE) (2015), IEEE, pp. 257–262.

[38] Vujičić, D., Jagodić, D., and Ranđić, S. Blockchain technology, bitcoin, and ethereum: A brief overview. In 2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH) (2018), IEEE, pp. 1–6.

[39] Wiki: blockchain. Blockchain. https://en.wikipedia.org/wiki/Blockchain, 2018.

[40] Wiki: Satosho_Nakamoto. Satosho_nakamoto. https://en.wikipedia.org/wiki/Satoshi_Nakamoto, 2018.

[41] Wiki: Single sign-on. Single sign-on. https://en.wikipedia.org/wiki/Single_sign-on, 2018.

[42] Wood, D. L., Norton, D., Weschler, P., Ferris, C., and Wilson, Y. Single sign-on framework with trust-level mapping to authentication requirements, May 10 2005. US Patent 6,892,307.

[43] Xu, X., Pautasso, C., Zhu, L., Gramoli, V., Ponomarev, A., Tran, A. B., and Chen, S. The blockchain as a software connector. In 2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA) (2016), IEEE, pp. 182–191.

[44] Xu, X., Weber, I., Staples, M., Zhu, L., Bosch, J., Bass, L., Pautasso, C., and Rimba, P. A taxonomy of blockchain-based systems for architecture design. In 2017 IEEE International Conference on Software Architecture (ICSA) (2017), IEEE, pp. 243–252.

[45] Zheng, Z., Xie, S., Dai, H., Chen, X., and Wang, H. An overview of blockchain technology: Architecture, consensus, and future trends. In 2017 IEEE International Congress on Big Data (BigData Congress) (2017), IEEE, pp. 557–564.

VITA

Samuel Matloob is currently working as software developer and system administrator. He has more than seven years of experience in the field of computer science. Developing applications has always been interesting and exciting to Sam, from designing the application, implementing it, securing it and delivering it. He uses a handful programming languages to write applications such as C++, Java, PHP, and C#. He also has an abundance of experience in Linux, Windows and Mac operating systems. He works well in teams and individually. Sam is always enthusiastic about expanding his knowledge, learning new technologies, conducting research independently, and finding solutions to challenging problems.