Palestine Polytechnic University

Deanship of Graduate Studies and Scientific Research

Master of Informatics

A framework for securing web applications against injection attacks using Blockchain technology

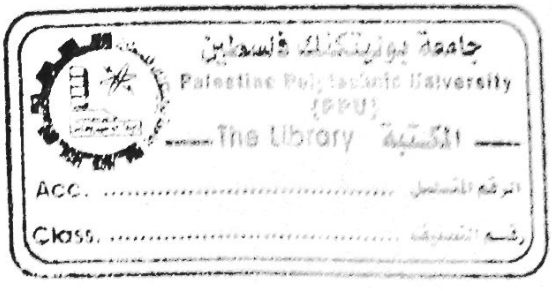
Submitted by

Mohammed Tmiezh

Thesis Supervisor: **Dr Ghassan Shahin**

A thesis submitted in partial fulfillment of the requirements for the degree of master of informatics

August - 2018



WE, THE UNDERSIGNED MEMBERS OF THE COMMITTEE, HAVE APPROVED THIS THESIS IN PARTIAL FULFILEMNT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF INFORMATICS:
COMMITTEE MEMBERS

Dr. Ghassan Shahin .....
Thesis Supervisor

Dr. Liana Al Tamimi .....
Internal Examiner

Dr. Mahmoud Al Awaiwi .....
External Examiner

Dr. Murad Abu Subaih .....
Dean of Graduate Studies And Scientific Research

A framework for securing web applications against injection attacks using Blockchain technology

by

Mohammed Tmiezh

Submitted to the Deanship of Graduate Studies and Research
on August 29, 2018, in partial fulfillment of the
requirements for the degree of
Master of informatics

Abstract

Web-applications of today are data-intensive. These applications have databases that store large amount of data, using almost SQL databases. On the other hand, such databases suffer from lack of security. One of the most common attacks is the injection attack that jeopardize the confidentiality and security of Web sites and, as a consequence, the databases. Recently Blockchain technology is spreading and developing in a fast pace. According to researches, it is considered as one of the most secured among other technologies. In this research, we propose a framework to solve or mitigate SQL and NoSQL injections problem, the proposed framework is used for prevention and detection of injection using Blockchain technology and query command Tokenizer to check the validity and permission of the commands. Therefore, providing better level of security for data. Results of simulation showed that proposed solutions attain sensible detection rate for many types of injections. In addition, results showed that Blockchain can provide a promising technology that can participate in securing web applications and databases.

Keywords: Ethereum smart contract, NoSQL injection, Query Tokenizer, SQL injection.

Thesis Supervisor: Dr Ghassan Shahin
Title: Assistant Professor

A framework for securing web applications against injection attacks using Blockchain technology

by

Mohammed Tmiezh

Submitted to the Deanship of Graduate Studies and Research
on August 29, 2018, in partial fulfillment of the
requirements for the degree of
Master of informatics

Abstract

just to save one page for arabic abstract

Thesis Supervisor: Dr Ghassan Shahin
Title: Assistant Professor

Dedication

This thesis is dedicated to my beloved mother, my dearest wife, and my beloved kids: Al-Mutaz, Ahmed, and Karmel.

Acknowledgments

Many people have directly or indirectly contributed to the successful completion of this thesis. They will all be remembered in my heart. First, I would like to take this opportunity to highly appreciate my thesis supervisor Dr Ghassan Shahin for his exemplary guidance and encouragement throughout my thesis research and writing.

I appreciate my family encouragement especially my grate mother and my wonderful wife. Their support and encouragement was the reason for all the success I have made.

I express my deep sense of reverence and gratitude to all of my respected teachers for their invaluable knowledge they imparted to me. And finally, my special thanks sholud go to Dr Khaled Tomizy for his constant inspiration and motivation.

Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

Acronym	**Definition**
ABI	Application Binary Interface
AES	Advanced Encryption Standard
CORS	Cross-origin resource sharing
DApps	Decentralized Applications
DAST	Dynamic Application Security Testing
DFA	Deterministic Finite Automaton
Ether	A fundamental cryptocurrency for operation of Ethereum
EVM	Ethereum Virtual Machine
Geth	A multipurpose command line tool that runs a full Ethereum node implemented in Go
Gwie	A denomination of ether (ETH), the cryptocoin used on the the Ethereum network. 1 Ether = 1,000,000,000 Gwei
HTTP	Hypertext Transfer Protocol
JSONP	JavaScript Object Notation Padding
NFA	Nondeterministic Finite Automaton
NoSQL	Not only SQL
NoSQLI	Not only SQL Injection
NoSQLIA	Not only SQL Injection Attack
PKI	Public Key Infrastructure
RPC	Remote Procedure Call
SQL	Structured Query Language
SQLI	SQL Injection
SQLIA	SQL Injection Attack
SVM	Support Vector Machine

Chapter 1

Introduction

In recent years, the internet has witnessed a notable growth of various types of online applications which have been developed for many purposes. Almost everyone is in touch with computer technologies especially with online applications. Those applications have their underlining databases that often contain confidential, sensitive, valuable information. This information can be highly valuable.

Furthermore, the web applications usually serve a huge number of users, great volumes of data are stored in those Web application databases. Generally, the users need to interact with the back-end databases via the user interfaces for several tasks such as updating data, making queries, extracting data, and so many others.

Since the back-end database often contains sensitive user data, it makes web application an ideal target for attacks. As a result, when developing such applications, the developers have to pay close attentions for the security of the stored data in the databases. A less secure Web application design may allow crafted injection and malicious update on the back-end database. Which may allow the unauthorized user to read, modify existing data, make the data unavailable to other users by deleting it, or even corrupt the database server.

An injection attack occurs when a malicious user manages to inject his own code into the program generated by the application. For example, this can be done based on the fact that applications accept input from users and integrate it into dynamically generated code [15]. For instance, a web application may ask the user to fill a form,

or submit his user name and password for authentication. The application then takes this input data and inserts it into a dynamically generated program. So, in the case of inserting a code instead of data, this code will be incorporated into original server side code forming a new modified code which can steal data, compromise database integrity, and/or bypass authentication and access control violating system correctness, security, and privacy properties.

1.1 Problem statement

The majority of the proposed methods for preventing many types of injection for relational or non-relational databases focused on filtering out common used characters that are used in injecting the code such as single quote and double dashes in Structured Query Language SQL of relational databases [24], or some other types of tokens such as $ne in Not only SQL (NoSQL) in non-relational databases like MongoDb [37]. This filtering in many cases add some limitations in the searching process since sometimes those prohibited characters are used or saved as a data inside the databases itself leading to an inaccurate results. As a result, when the searching filters omit those characters from the query, this will lead to abandoning those characters from the search process even the intent of using them is not malicious attempt. Additionally, many filters don't cover the new tokens that may use for injections purposes especially for the new type of NoSQL databases, due to the continuous upgrading of those types of databases.

Few works concern with authorization issues for queries that test whether the user is permitted to use some particular queries or not. The problem here is that the permission relation between users and queries will be saved either in the application server or the back-end databases [48] Som et. al.[45]. If the attacker gains a root access for the application server, or even inject the database, all security methods will be broken. As we figure out, we need a trusted place for saving the authentication relation for the users and their associated queries. Before application server establish the connections with the database, we have to make sure that query is not forged,

as well as the user that wanst to execute this query has the permission to use this particular query.

Furthermore, despite that traditional ways for securing data in relational databases does not work for non-relational databases or NoSQL databases, we can use a model or a structure that can solve many types of injection in both database types.

1.2 Research objectives

This research aims at developing a solution for many types of injections in both SQL and NoSQL databases. This research intends to address the following objectives:

- Developing a method that restricts the use of queries throughout linking the permitted query with their associated users, in other words, making an authorization rules over the used queries.
- Enhancing the method of tokenization for the queries that extract the query code from the run time query so as to be accurate in facing many type of attacks.
- To develop a Blockchain based model for securing database against injection.
- Validity of the proposed model.

1.3 Research questions

Research questions represent some sub-objectives of the research. We achieve our objectives of the research by implementing every sub-objective. Experiments were conducted to seek answers to the following research questions:

Research Question 1: How could authorization rules over the used queries enhance the effectiveness of eliminating database SQL and NoSQL injection attacks in web applications?

Research Question 2: What is the effect of using enhanced query Tokenizer for detecting SQL and NoSQL injection attacks in web applications?

Research Question 3: How can the use Blockchain technology improve the security of web applications against SQL and NoSQL injection attacks?

1.4 Motivations

Nowadays, as network security increased and became somehow not easy to breach for the operating systems of the servers. In spite of that, data injection such as SQL injection (SQLI) or NoSQL injection (NoSQLI) has been developed and become the most growing attacks in the recent years [38]. The inherited weakness in field of security of the early relation databases such as SQLI is still somehow exist in the new NoSQL databases. The founder of those databases emphasizes the speed of developing and producing new tools and technologies in term of performance to face the new challenge of dealing with huge amount of data and the revolution of the big data[31]. This leads to the fact that the attackers can leverage from the vulnerability to inject with a malicious queries in order to reach or modify the data.

Thus, there is an essential need to find an effective, feasible, and robust solution for this weakness in the database and computer security field. Detecting and preventing such type of data injections is an active research topic in the academia and industry. In our work we choose to use Blockchain technology to work as a layer that stores the permitted queries that are allowed to be executed on the back-end database. Using Blockchain is due to many reasons such as the fact that Blockchain is immutable so that the stored data can't be changed once it is added to the block. In addition, the Blockchain is a decentralized application that is run by many nodes on a decentralized network which designed to avoid any single point of failure.

1.5 Structure of the Thesis

Chapter 2 provides an overview, introduction, and background for SQLI and NoSQLI attacks, importance, types, and it consequences. It also gives a background for Blockchain technology, some of its used platforms, and an overview of Ethereum

smart contracts. Chapter 3 shows a review for many of existing works and the draw backs of them in the field of data injection, describing the recent proposed works and researches for mitigating, detecting, preventing SQLI and NoSQLI attacks using various approaches such as user input filter, query integrity and mitigation approaches. Moreover, it shows many Blockchain based approaches for security and privacy. Chapter 4 introduces what research methodologies are selected for this thesis, how we design the research, what experimental tools are used and the expected outcomes. In Chapter 5, we have proposed our framework approach to detect and prevent the SQLI and NoSQLI attacks at the run-time, including the framework architecture and design concept, the proposed query Tokenizer, the smart contracts architecture, the structure of the participating nodes, and the implementations costs in Ethereum Blockchain platform. Chapter 6 demonstrates successful SQLI and NoSQLI attacks conducted by various methods to evaluate our proposed technique, as well as the implementation details, and then we have discussed the results to show the effectiveness of our proposed work. Chapter 7 concludes the thesis along with the future work, followed by the references.

Chapter 2

Background

This chapter provides a background and introduction for SQLI and NoSQLI attacks, importance, types, and it consequences. Next, this chapter also gives an overview of Blockchain technology, some of its used platforms, and an overview of Ethereum smart contracts.

2.1 Database injection

Internet has rapidly been developed, internet users mainly browsing websites. Recently number of websites has grown to be more than 1.8 billion in 2018 based on [28]. Web application is the main software program to support website, web applications should have a repository of data that is called back-end database. Back-end database store information which is the most important business asset. Web applications are often vulnerable for attackers who often can easily gain an access to the application's back-end database. SQLI or NoSQLI attacks occurs when an attacker, through specifically crafted input, makes a web application to generate and send a query to the back-end database that functions differently than the programmer intended. Databases Injection Attacks in general have been known as one of the most common threats to the security of database-driven applications. According to statistics from OWASP top 10 [52], one of the most common and harmful attacks is SQLI.

2.1.1 SQL injection

Based on [47] about 120 SQLI attacks alleviation mechanism research papers were proposed between the years from 2004 and 2016, that were identified. Many of websites have been attacked using SQLI [49]. SQLI still one of the most used attacks in the recent years [18]. SQLI is an attack that when the attacker appends an SQL code into application user input parameters that are subsequent passed to a back-end database server for parsing and execution. When an attacker has the ability to manipulate and change the syntax of the SQL statement, the new modified statement will be executed with the same privileges and permissions as the legal SQL statement that has been done by the application developer.

2.1.2 NoSQL injection

Recently, the major internet companies like Amazon, Face-book, and Google, are using the NoSQL databases[26] such as Mongodb, Cassandra, CouchDB and HBase. The main reason behind using such databases is their ability to cope with huge amount of data. The NoSQL systems are non-relational, distributed databases, that designed to work on large-scale data storage and massively-parallel data processing across a large number of commodity servers, offering performance and scaling benefits. Nevertheless, those databases suffer from lack of security that make them dangerous and exposure to many types of attacks[38]. One of the attacks that those databases suffer from is NoSQLI. Since NoSQL databases use different types of query languages, this leads to that traditional SQLI techniques is not working in this type of databases. But this does not mean that NoSQL databases are immune for such type of injection[40]. As a result there still many ways to inject the NoSQL database. Many studies showed that NoSQL databases have security weaknesses and problems such as [31] showed that Cassandra and Mongodb have vulnerabilities for injection attaches as well as weak authentication between the client and the servers, therefore a lack of authorization also is exist.

NoSQLI vulnerability refers to one of the most common problems of database-

driven web applications [19]. In the past the relational databases suffer from the same threat which was known as SQLI. The SQLI or NoSQLI actually refer to the same concept. Both of the attacks share the leverage of the vulnerability in the database, where the attackers use a malicious code to manipulate the queries, changing them in which may leads to steal data, information leakage, compromise database integrity, bypass authentication, violating system correctness, security, and privacy properties. NoSQLI happen when a query is built with unsanitized user inputs containing malicious characters, then it is possible to modify the behavior of the executed query[35]. The main difference between SQLI and NoSQLI refers to that SQL has a standard language whereas each NoSQL database uses its own query language which consequently mean that SQLI techniques are irrelevant. Furthermore, the NoSQL databases such as, Mongodb which is ranked fifth among the 10 most popular databases in 2017 according to db-engines.com popularity ranking[10], use JavaScript Object Notation (JSON) and/or JavaScript as query languages. The ability to run JavaScript code add a new opportunity to the attackers to leverage from this point as demonstrated in[21].

2.2 Blockchain background

Recently, too many concerns in this technology have been appeared. Some of the main projects that are established using Blockchain are the Bitcoin and Ether crypto currencies. Blockchain is a chain of continuous data records called blocks which are linked together, and secured using cryptography. Furthermore, the saved data are immutable and can't be changed once it is saved which is considered as one of the main advantages of the Blockchain technology. Blockchain database is managed autonomously using a peer-to-peer network and a distributed time stamping server. Blockchain is a decentralized, distributed and public digital ledger that is used to record transactions across many computers. The block is holding batches of transactions that are encoded and hashed in a Merkle tree [25] as shown in figure 2-1 that leads to form the hash root of the whole transactions of that block. Each block

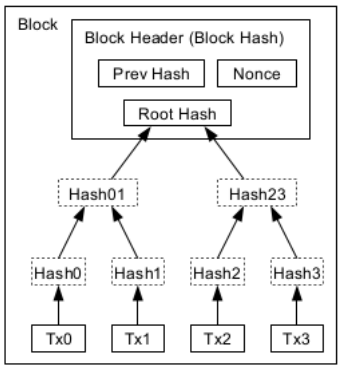


Figure 2-1: Transactions Hashed in a Merkle Tree. Image source [27]

contains a timestamp and connected with the chain by saving the cryptographic hash of the previous block in the chain of blocks as illustrated in figure 2-2. Each block in a Blockchain has two components, content which is the transactions, and the header. The content is the data from a source and the header contains the meta-data about the block itself. Also the figure displays how blocks are connected with the chain by saving the hash of its previous block.

Any modification of the transactions leads to break the chain. The participating computers or nodes in the Blockchain have the same Blockchain of the saved records which called public ledger, as a consequence any attempt to modify any transaction from a particular computer can be detected by consensus algorithm.

As we can illustrate, the Blockchain is decentralized application that cannot be manipulated by particular computer like what happened in the centralized application; which manipulates by the intermediary server that control the whole process. In the centralized applications any malicious user that can access to the intermediary computer-sever- can control the network, which is not the case in Blockchain since if the attacker wish to control the network he needs to control at least 51% of the participating nodes in that network of Blockchain [27].

The Blockchain uses public key cryptographic techniques to generate an append-only, immutable, time stamped chain of content. Copies of the chain are distributed so as each participating node in the Blockchain network has the same copy.

In Blockchain the new blocks are added by the miners. Miners are nodes in the

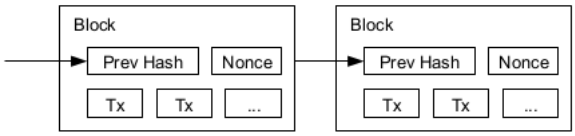


Figure 2-2: Chain of blocks. Image source [27]

Blockchain network that work to validate the new transactions and record them on the global ledger that called Blockchain. A miner executes the computation associated with each transaction being included in a block, resulting in an updated state. When the miner successfully mining a block, it broadcasts the block to the network of Blockchain. Then all of other miners and non-mining nodes verify the validity of the transaction computation and resulting state change before accepting the block as valid, adding the new block into their copy of the Blockchain, and moving on to the next block. Miners receive a reward when they solve the complex computation mathematical problem associated with each transaction. Furthermore, Miners can also receive rewards in the form of transaction fees[27].

2.2.1 Bitcoin

In the recent years, a new trend of digital accountable systems emerged. Bitcoin was the first such system that was introduced in 2009. The main aim for Bitcoin is to allow users to transfer currency securely without a third party or centralized regulator, using a publicly verifiable open ledger or Blockchain[27]. Bitcoin can generate a trustable record of Bitcoin transactions, without needing a central owner or manipulator such as bank. It demonstrated how Blockchain can serve other functions requiring trusted computing and auditability. In Bitcoin Blockchain to secure the content from being tampered Proof of Work algorithm [20] is used. The Proof of Work algorithm used to define and verify the requirements for the generation of a new set of transactions that form the block, this new created block will be added to the Blockchain.

2.2.2 Ethereum

Ethereum is to create an alternative protocol for building decentralized applications. Ethereum utilize the technology behind Bitcoin with modification to its capabilities (It enables users or nodes to create decentralized applications on Ethereum's Blockchain). It was described by founder Vitalik Buterin in 2013[8]. Ethereum has an internal currency called "Ether". Ether is the main internal crypto-fuel of Ethereum, and it is used to pay transaction fees. Furthermore, there are two types of accounts in Ethereum, the first type is the externally owned account, that controlled by private keys. The second type is contract account, which is controlled by its contract code. On Ethereum every account is represented by an address. Ethereum allows us to run decentralized applications (DApps) on top of it. The DApps are formulated using smart contracts[8]. More details on smart contracts will be in the next section.

Ethereum uses an alternative mining approach called Proof of Stake that is based on Proof of Work algorithm [7].

Smart contract

Smart contracts are programs that run using Blockchain platform such as Ethereum [8]. Contracts are written in a Turing-complete bytecode language, called Ethereum Virtual Machine (EVM) bytecode [53]. The contract composes of a set of functions. The user can send the contract to the Blockchain as a transaction to create it, invoke its functions, make decisions, or even to use that contract to send ethers or coins to other users. Contracts will exist and run as long as the whole network exists, and stop only in case of they run out of gas or if their destruct function is programmed to do so. Smart contracts are uniquely tamper-proof. Since Ethereum does not depend on a trusted central party, each transaction can be processed by a large network of mutually untrusted peers called miners, the miner is a node on the network that mines or works to process blocks on the Blockchain network.

Writing a smart contracts can be done using some different languages such as Solidity, which is like JavaScript and has (.sol) as a file extension. Serpent also can

be used, which is a Python-like. To call and execute a smart contract function in the decentralized network of Blockchain, we need to have access to one of the Blockchain clients. Also a special bit of code called Application Binary Interface (ABI) is required to interact with a smart contract. ABI is used to call the unreadable functions from the Ethereum contract address. ABIs are generally generated during compilation and will be placed into a separate file next to the Solidity source code.

Chapter 3

Literature review

Researchers have proposed many different defense techniques to address the problems of SQL injections attacks (SQLIA), and NoSQL injections attacks (NoSQLIA). The different techniques used to face such attacks can be of different in defense natures. The nature of defense type can be classified based on many aspects for instance it can be based on the type of attack such as Halfond et. al. [14], where the authors classify the SQLIAs to Tautologies, Union Query, Piggy-Backed Queries, Alternate Encodings, and many other SQLIA types. Whereas another study for Oussous et. al. [34] make NoSQLIA classification based on the type of used NoSQL database. In this study the authors define and compare four categories: key-value stores, document databases, column-oriented databases and graph databases. To conclude, the nature of defense figures out how a technique is going to defend the application from injections.

In our study we made a review of many papers in the field of injection in both SQLI and NoSQLI. In addition to that, we reviewed many papers that used the Blockchain technology to enhance the security and privacy in the field of computer. We classify the reviewed papers based on their proposed work or the goal of the research. We first made a category for the papers that provided a recommendation that aims to mitigate injection attacks. Furthermore, another category added for papers that aim to solve injection attacks by checking the integrity of the used query during run time phase. Next, we created a new category for the studies that aim to face inject attacks by

filtering the user input string. Finally, we studied the Blockchain based models and many works that aimed to enhance security and privacy in many aspects of computer related filed.

3.1 Mitigating injection based approaches

Ron et. al.[38] have a study that concerns with analysis and mitigation of NoSQL Injections. The authors studied how to mitigate injection risks in NoSQL databases like MongoDb and Cassandra. They showed that during code analysis in the application layer is insufficient to ensure that all threats are mitigated. In their study they recommended to use the mature databases that have a built-in security measures. In addition to that, they reveal that even using the secure data store does not guarantee the prevention from injection attacks due to the fact that the attackers use the vulnerabilities in the Web applications in order to access the data store. Furthermore, in this study the authors pointed to some challenges that make the code analysis techniques in the application layer is insufficient, initially the complexity of the applications that use cloud big data. In addition to the short gap between the development and productions of the modern code methodologies which result in lake of security. Finally, the fact of most of applications security testing tools do not follow the rapidly growing new programming languages for instance Scala[41] Haskel[16] are supported few security testing tools. To conclude, the study suggested that the security development should be addressed fully in an iterative way for the whole life cycle of the application in order to address as much as security threats. Whereas, Shar et. al.[43] stated in their study that better understanding of SQL injection leading to practically defenses against it. The study provides specification of good coding practices than consequently leads to mitigating of the SQLIAs.

Another study showed that the NoSQL databases are exposed to injection. In Ron et. al.[37]. The authors aimed to demonstrate how the popular JSON representation format allows new types of injection attacks. They used PHP web scripting language along with MongoDb NoSQL database as a case study to prove in their claims. They

showed that using associative arrays in PHP can be used to inject Mongodb which is similar to the case of SQLI. Another vulnerability that they showed is the exploitation by the attackers during the login phase to inject the database and bypass the login, using a correct username without password. Due to the ability of NoSQL to run JavaScript this also expose the database to real dangerous attacks. For instance MongoDB, CouchDB, Cloudant, and BigCouch are all NoSQL databases that execute JavaScript code. In this paper the authors showed that despite that the database is deployed in secure networks, the attackers can find a vulnerability that can be used to inject the database. Since some queries can be initiated from the browser using HTTP REST API, the attacker can trick a user to use one of his own websites that contain a malicious code to inject the database that the user permits to use. The authors made some recommendations for each of the threats in order to mitigate injection attacks. The authors recommended running Dynamic Application Security Testing (DAST) and static code analysis on the application in order to find any injection vulnerabilities. Some measurements can be adapted to mitigate REST API injections attacks such as disabling JavaScript Object Notation with padding (JSONP)[5] and Cross-origin resource sharing (CORS). Finally this paper recommends using a proper authorization and privilege isolation that helps to minimize the damage in case of data store exposure. Table 3.1 summarized the studied approaches in this section based on the year and types of used database.

Study	Year	Type of database
Ron et. al.[38]	2016	NoSQL
Ron et. al.[37]	2015	NoSQL
Shar et. al.[43]	2013	SQL

Table 3.1: List of studied approaches that used to mitigate injections

3.2 Queries integrity based approaches

Many works concerned with the integrity of the query at run-time phase, these studies aimed to test the run-time query to see if it is the same as the query at rest in the application server, the at rest query is the query that done by the developer during developing the application. Many approaches tried to test the integrity of the run-time query. For instance, a study for Shahriar et. al.[42] worked to detect SQLIA using an information-theoretic approach. In this study the authors defined entropy as a measure to understand the complexity of the static query written by programmers using a probability formula. This formula basically uses the number of reserved words that used in the query as input. When a malicious inputs successfully alters the static nature of the query, definitely the complexity value changes, which means that the number of words in the run-time phase has changed.

In addition, another study for Eassa et. al.[11], this study presented a testing tool for detecting NoSQLIAs which was called NoSQL Racket, this tool implemented as a PHP function for detecting the count of reserved words that used in the original query, compare the result with count of reserved words in the run-time query. In ordered to identify the reserved words the authors offers a table that contains those words classifying based on the NoSQL database type. Based on the comparison result the tool can identify if there is an attempt to implement a malicious code or not. In other words, the authors aim to check the integrity of the query during the run-time phase. This study covered many types of NoSQL databases such as MongoDB, Cassandra, CouchDB and Amazon DynamoDB.

Another study to validate the integrity of the results returned for the query, by validating the attributes that are locating after the where condition had conducted by Wang et. al.[50]. In the query (q) as shown in figure 3-1, the authors use keyed encryption algorithm. Part of query (q) will be encrypted when it is sent to the server. Since they have two keys, the query can be transformed into two different forms key.

Ntagwabira et. al.[30] also proposed a new text awareness approach. The authors

$$q^k: \quad \text{SELECT} * \text{FROM } T_s \\ \text{WHERE } a_1 = E_k(2) \text{ AND } a_3 < E_k(100)$$

$$q^{k'}: \quad \text{SELECT} * \text{FROM } T_s \\ \text{WHERE } a_i = E_{k'}(2) \text{ AND } a_3 < E_{k'}(100)$$

Figure 3-1: Encrypt part of the query. Image source [50]

implemented a text awareness method as a text Tokenizer. The query is split in to tokens, all tokens are saved in an array. The tokenization method is applied on the original query then it will be saved. After that the same process will be applied again for the run-time query then it will be saved in another array. Now by comparing lengths of the resulting arrays from the two queries the original query and the run-time query, if they are not with the same length then it is considered as injection attempt, otherwise it will be accepted. The tokens in this method are detected use the space, double quotes or double dashes.

Whereas another study for Kim et. al.[23] used another approach to verify the integrity of the run time query by using data mining, the used approach depends on artificial intelligent using Support Vector Machine (SVM) with queries at database level. In this study the web users address URL file was extracted, cleansed, formatted and classified to meaningful session for data mining analysis process, data mart was developed, this is as a result of the fact that the raw URL file that extracted is not well structured to be used directly for data mining. This study did not contain authorization roles associated with user. Furthermore, another proposed method by Aich [1] used a query parser and store the valid query structures in terms of an orders sequence of tokens in a linked list. This work uses this linked list as an intermediary layer between the database and the application server. It provided a solution for structured databases, in other words for SQLIAs.

As we can illustrate from table 3.2, many of the previous proposed studies focused on how to test the integrity of the run-time query using various methods. For instance, [42] utilized reserved SQL words to count number occurrence for those words in rest and run-time query. As well as [11] used similar approach for NoSQL databases but

with some modification, such that the used reserved words of the rest and run-time query will be saved in two arrays, then compare those two arrays for testing the integrity of the run-time query. Indeed, [50] used cryptography algorithm to cipher the value of attributes after conditional where in SQL queries. Moreover, [30] used Tokenizer method in both phases, at rest and run-time query then save the results in two array for further comparison purposes to identify the integrity of the query. Alternatively, [23] used artificial intelligent based method to certify that the intent of the query is not injection attack.

Study	Year	Type of database	Used method
Eassa et. al.[11]	2017	NoSQL	PHP function for detecting the count of reserved words that used in the original query
Kim et. al.[23]	2014	SQL	Artificial intelligent using Support Vector Machine (SVM)
Shahriar et. al.[42]	2012	SQL	Information-theoretic approach using probability formula
Ntagwabira et. al. [30]	2010	SQL	Text awareness method and comparing lengths of the original query and the run-time query
Aich [1]	2009	SQL	Used a query parser and store the valid query structures in terms of an orders sequence of tokens in a linked list
Wang et. al.[50]	2008	SQL	Validating and encrypting the attributes that are locating after the where condition

Table 3.2: List of studied papers that used integrity based approaches

3.3 User input filtering approaches

The user input filters is another way to detect and prevent injection attacks in both SQL and NoSQL databases. In this section we reviewed some studies that focused on this discipline, the studied papers used many approaches to overcome this problem. In Joseph et. al.[22], their work focused mainly on time based and blind based Boolean injection attack. They used JavaScript functions to exploit the MongoDB. In time based attack the attacker appends a JavaScript function which puts the database on hold. In Boolean attack the attacker does not have any knowledge of the contents of a collection, however he exploits the various functionalities provided by MongoDB to find information about the content of the database. The authors implemented an Automata based approach for the detection and prevention of MongoDB injections. They used Non-deterministic Finite Automata to check the user input match the expected inputs or not.

Another study for B. Hou et. al.[17] aimed to detect and prevent NoSQLIs. In this paper the authors proposed a defense analysis method for MongoDB the defense is composed of two ways. The first way concerned with input validation, which is to limit what users input using JavaScript on the client side. The second one, parameterized statement, which is concerns in checking and filtering the variables during the run-time of writing condition statements in the query, as a result user input variable not directly embedded into the query condition statement.

Som et. al.[45] proposes new approach for detection and prevention of SQLI. The proposed method consists of two phases: (i) Front-end Phase. (ii) Back-end Phase. In the front phase when a new client registers in the system, the server will create a hash code for the user name and for the password then concatenates both hashes in one string, after that the new string will be stored in the table of users. When a login process from any client initiated, the server will do the same process of hashing for the entered user name and password. Then sever will search for the hash values inside the users table that stored in the database side, and if they matches then this means that the user is authenticated and no injection attempts detected.

Another proposed solution by Balasundaram et. al.[4], this approach used AES cryptography algorithm to prevent SQLIAs. This method has three phases, registration Phase, login Phase, and verification Phase. In the registration phase after a new client entered the desired user name and password, the server will send these data to the database alongside with random generated secret key for each new registration process, this key should be unique for each user. Then save a copy of the username, password and the secret key in table inside the application server, after that it will send a copy of that secret key to that client or user. In the login phase the user name and password will be encrypted using AES algorithm by applying user secrete key in the client side, then the query will be sent to the server. Finally, in the verification phase, the server will receive the query result that sent by the user and performs the following steps, the server receives the login query and verifies the corresponding users secrete key using the data in the saved table. In case of secret key match, the corresponding username and password will be decrypted from the query by using this key. To conclude, if they match then accept the user, otherwise reject him and consider it as malicious attack.

Furthermore, Ali et. al.[2] proposed SQLI Protector for authentication, which considered as an efficient solution for preventing SQLI in the login phase. In this method the user name and password parameters are stored in a hashed way in the data base, when a login initiated by any client the username and password are hashed using the same hash function. When the attacker tries to inject the query the injecting code is converting to a hash value, which needs to be compared with the existing value. As a result, there is no way for the attacker to inject the query using this method. Also another study by Roy et. al.[39] for inspecting the user input parameters. The authors proposed a mechanism to prevent and detect SQLI in a web application using SQL Meta Character Filter which placed in between the user and the application server to intercept the entire request coming from the user. The Meta filter parses the input URL to detect attack patterns.

As we can illustrate from the table 3.3 that summarized the reviewed studies in this section. In [22] the proposed work is working for NoSQL databases mainly

Study	Year	Type of database	Used filter
B. Hou et. al.[17]	2016	SQL	JavaScript filter on the client side and parameterized statement on the server side
Som et. al.[45]	2016	SQL	Hash code for the user name and for the password during the login process then matches the result with saved hash data
Joseph et. al.[22]	2015	NoSQL	Automata based approach for the detection and prevention of MongoDB injections
Balasundaram et. al.[4]	2011	SQL	AES cryptography algorithm in Registration Phase, login Phase, and verification Phase.
Roy et. al.[39]	2011	SQL	Meta Character Filter which placed in between the user and the application server
Ali et. al.[2]	2009	SQL	User name and password parameters are stored in a hashed way in the data base then compare then with the hashed run time user name and password

Table 3.3: List of studied papers that used filter based approaches

for Mongodb, the work aimed to check the user input inside the application server using Non deterministic finite automata. However, some questions still need to be answered, for instance, what if the entered input by the user is invalid, even if it is not attack-intended, it will be flagged as injection attempt. Also the developer has to specify the accepted pattern of user input for all attributes that used by queries which need considerable efforts. Nonetheless, it is not clear when to use deterministic finite automaton (DFA) since the authors said that they could use it rather than NFA in some cases. On the other hand, in [2], [4] and [45] the proposed works can serve in case of login phase to protect against injection. [2], [4], and [39] proposed methods to face SQLIAs, where as [45] used for NoSQLIAs and mainly for Mongodb. But in [17], the work has two stages. First stage is at client side in face of JavaScript code same idea as validation controls. The second stage is at server side which is based on saving the parameters of the condition of the query in variables and performs some

validations rules against them. Though, the validation controls in the client side can be bypassed by the attackers using traffic manipulation tools such as Browser modification extensions or proxy servers. Moreover, the proposed work did not pay attentions for the time matter, and still a need to study the time consumed to check each user input variable in the used layers, in order to study the efficiency of the used method.

3.4 Blockchain based approaches for security and privacy

Recently many works have been done based on Blockchain concept. In this section we made a review for some of those researches that aimed to use this technology in order to enhance security and privacy. Zyskind et. al.[55] proposed a novel decentralized personal data management system that ensures the user ownership of their sensitive data without trusted third party. This system is implemented using Blockchain. The system can protect the data against these privacy issues such as Data ownership, Data transparency, and Fine-grained access control. This work converts Blockchain to adapt non financial transaction. The transactions can either save data or save permissions for the services. Based on that, in case of updating the permissions for any service, the user can change the permissions granted to a service at any time by issuing a new transaction with a new set of permissions. In [29] the author proposed a Blockchain based anti-malware system called BitAV. This system uses a decentralized approach of Blockchain to propagate malware identifiers that less susceptible to be targeted by denial-of-service attacks. The identifier of the malware is the signature of the male-ware using cryptography algorithms that focus on the creation of a bridge between the currently deployed anti-malware, and future of anti-malware systems to create the optimal anti-malware solution. The main aims of this solution is to make scanning speed better with less memory usage, decentralized updates and maintenance, scalable, and can be implemented easily on all type of devices. Another

proposed work in [3] which introduced Blockchain based method for public key infrastructure (PKI) as a privacy-aware PKI while simultaneously improving the reliability of conventional PKI. In this work the authors described a method for the user to update his public key without linking it to his ID in the system. This Blockchain based system is used to management of update, registration and revocation of keys. Blockchain technology is used in this work to ensure that biometric data are stored in an environment that ensures the security and integrity of such information. Ekblaw et. al.[12] proposed a electronic health medical record system called MedRec based on Blockchain technology. This work gives the patients an immutable, comprehensive log as well as a secure and easy access to their medical information that are scattered across many providers and treatment sites. In this work manages authentication, confidentiality, accountability and data sharing are the main contributions of using Blockchain based approach. This work used Ethereum platform of the Blockchain.

There are other Blockchain based works and researches that proposed aiming to enhance and preserving authorization in addition to meet new Internet of Things (IoT) security requirements, for instance [33]. In this work the authors proposed an access control framework implementing a new distributed framework based on Blockchain technology, the framework named FairAccess. This framework provides an access control mechanisms over constrained environments for IoT where users are able to control and master their own privacy. Furthermore, framework introduces new types of transactions that are used to grant, get, delegate, and revoke access. In this work the authors used the Bitcoin test network. The Bitcoin transactions are used to provide access control, and the Blockchain used it for storing and reading the permissions. Many applications such as electronic voting systems or bank information systems require a robust and tamper-proof logging system. An applied solution was proposed by Cucurull et. al.[9] for supporting such Field. The authors in this work introduced a proposal and implementation to immutabilize integrity proofs of the secure logs based on Blockchain. The proposed work took advantage from the Blockchain to distributedly publish and immutabilize log integrity proofs. The implementation had done on the top of the Bitcoin's Blockchain. Using Blockchain

provides additional integrity and non-repudiation security properties resilient to log truncation and log regeneration in cases in which the logger or its signing key gets compromised.

As we can illustrate the Blockchain is a promising technology that can be used in many fields and mainly in those fields that are seeking for integrity, privacy, anonymity, and security.

3.5 Summary

Too many valuable efforts have been spent in the field of database security, mainly in data injection detection and prevention. Some of those works concerned with how to mitigate injection by giving recommendations for developers to be taken into account during the early software life time. Others implemented filter based approach that aimed to find the suspected characters, malicious codes or even tokens that may be used for injection purposes in order to omit or escape them. Alternative works focused on how to face injection threats during login phase by implementing hashing techniques on the login parameters and adding salts such as secret key that is saved in the application server. Furthermore, some of the proposed works focused on the integrity of the used query during the run-time. But what if the filters based approaches omit characters or token that the user used without any intent to make any malicious behavior. Does this lead to accurate results in searching process or queries? On the other hand, what about the approaches that aim to protect against injection during the login phase. Are those solutions enough? And does this mean that authenticated users are safe and after they get login. Is there any guarantee that they are not going to inject the data? Another issue arises related to the integrity based approaches is that whether or not the used methodologies are immune to injection and give the desired results in facing the injection threats while they neglect the white spaced characters that are embedded inside the query in order to bypass the malicious code. On the other hand, the majority of the reviewed works utilized the application server to test the validity of the query during the run time, but what

if the attacker gain access for the application server?

In our work we aim to overcome the databases injection problem by enhancing the authorization rules associated with the users which can be done by assigning permissions over the queries for each type of user. In additions, to save those rules in Blockchain, smart contracts can be used rather than the application server or database server so as to add a layer that is independent, secure, immutable, and isolated from the application and database servers.

Chapter 4

Methodology

Our proposed work aims to make a hybrid model that joins the application server and the Blockchain together in order to save the permissions details for the application users and their associate queries in a Blockchain network. This will add positive effects for enhancing the security against data injection such as SQLI or NoSQLI. In addition to that, this model will enhance the authorization as any user will use the queries that are dedicated for him.

4.1 Framework design

The proposed framework consists of two main components. The first component is the Blockchain which is a layer that will be used to save the user types and their associated queries, in other words, the queries that an individual user has the permission to use. The second component is the query Tokenizer which has the responsibility to identify the query string that is used, namely to extract the words or tokens in that query string that are used as a command, reserved word, or sometimes even it is a table name or an attribute. Those extracted tokens will be chosen based on a predefined table of tokens. After that, the resulting new string will be hashed using SHA256 cryptographic algorithm. Finally the result will be send to the Blockchain for two purposes for saving or for checking purposes.

The aforementioned components will be used together inside the application server

in order to enhance the detection and prevention for many types of database injection attacks. More details will be presented in the following sections about those components.

4.1.1 Blockchain layer

In this section we aim to provide a layer between the application servers and back-end database server. This layer will serve as an authenticator one which means that any attempt to open connection between the application server and database server should be first take permission from this layer.

We used Ethereum Blockchain platform since it builds into the Blockchain a Turing-complete instruction and programming language that set to allow smart-contract programming and a storage capability to accommodate on-chain state. Smart contracts are account holding objects on the Ethereum Blockchain. Those objects are created when deploying smart contracts on Ethereum Blockchain network. They contain code functions and can interact with other contracts, make decisions, store data, and send ether currency to others.

Smart contracts are used in two stages: At the first stage we will use the smart contracts to save the polices or permissions for a specific user. Each user who wants to use the database should have an associated smart contract that contains a hashed values for the queries that he has a permission to use them. The next stage of using the smart contracts is using a smart contract that serves as index book. It will save all types of users alongside with Ethereum account address for their dedicated users smart contracts so that any type of user will be mapped to his Ethereum address of his smart contract. Policies coded into the index contract should regulate registering or adding new type of users or even changing the mapping of existing ones in case of modification. As well as the user smart contract should be able to add new hashed values or modify existing hashed values.

Adding new users or new hashed queries should be restricted to the user how creates and owns the smart contracts that used in this framework.

To build this framework private Ethereum Blockchain is initialized. The partici-

pating servers or nodes should first joint to the Blockchain of the private Ethereum. Then it can create an Ethereum account address. After that, it can use that address to create the smart contracts.

After node or server participate in the Ethereum Blockchain the developer or the administrator of the application should save the hashed values of all queries using SHA256 cryptographic hash algorithm, and specify whom users that are permitted to use them. The saved data should be placed in a dedicated Ethereum addresses for that server. The administrator or the developer might add the users, and their hashed queries directly in Ethereum console screen or using application interface that we will develop for that purpose.

4.1.2 Query tokenizer

The hashed queries that will be saved in Blockchain will be generated from the original queries that are supplied by the developer. But those queries will not be hashed directly. First, those queries will be splitted into tokens using text awareness method or what we called here the Tokenizer. Then the reserved words as well as some other attributes will be extracted to be hashed and saved on the Blockchain.

Choosing the attributes depends on the type of used database as structure database (SQL), and semi-structured or unstructured database (NoSQL). To solve this issue, the reserved words and the chosen attributes of the dedicated or used database will be firstly grouped in a table. As a result the Tokenizer will choose the wanted tokens based on that table.

Finally, the application server contains the queries that are going to be implemented in the back-end database. In this work, the focus is on the embedded queries. At the run-time phase the same process of splitting query into tokens that are used before in the previous stage will be used here. so, the run-time embedded queries will be splitted and hashed as the same use method before. Then before establishing the connection with the back-end database, the run-time hashed query will be sent to Blockchain with the associated user that going to run it. The Blockchain will check if that user is permitted to use it or not. Consequently, the decision of establishing

the connection based on the return result from the Blockchain. As a result if there is no matching between the hashed query for that user, the run time query will be treated as injection attempt and will be rejected from the application server.

4.2 Experiment procedures

The experiment research procedure is divided into three phases. At the begging we will use a web application that uses SQL database to test its vulnerability for injection attacks before and after adapting the proposed framework. Secondly, we are going to repeat the same previous experiment but with a web application that has NoSQL database. Finally the time consumed during interacting with Ethereum smart contracts will be measured to test the efficacy of using Blockchain in the proposed work.

In order to test website vulnerability for SQL injection attacks many methods can be used to accomplish this goal, for instance, a direct manual or automated methods can be used. The automated method is done using many tools that can perform an SQL or NoSQL penetrating tests, many open source tools can be adapted to perform this test.

In our experiments and simulations we choose to use both options. In the automated option an open source tool called Sqlmap is used. Sqlmap is a Python-based tool, it can automatically detect SQL vulnerabilities, fingerprint database, exploit vulnerabilities, extract data from database, access the file system of the web server and execute program commands on the operating system. We choose to use this tool since it is one of the most effective SQL injection attack penetration tools, based on a previous study [54] that made a comparison between many such tools for SQLIAs, the study showed that Sqlmap is one of the most effective tools that has been used for this purpose. Additionally, the manual testing will be used to cover some types of attacks that are not covered by the chosen tool suach as tautology attack that mentioned in table 6.1.

The method of injection testing for the NoSQL website in this research is the man-

ual testing. Since to the best of our knowledge, there is no general tool for this purpose due to the absence of a common standard query language for NoSQL databases. The time consumed during interacting with smart contracts will be measured using code simulation by sending different number of transactions to Ethereum client. Mainly, we focus on the call of functions that will be used to check the permission of using some queries, since those functions will have a direct impact on the efficiency of the proposed framework.

For implementing the prototype PHP is used as web programming language for the web application, MYSQL, and MongoDB as databases for storing the data. Thre prototype is implemented on PHP because it is considered to be one of the strongest and most significant web development languages that support a wide range of databases. Additionally, it can be used on all major operating systems. As well as, it has useful text processing features and parsers[36], that can be useful for the proposed query Tokenizer.

MySQL is used in the prototype implementation for SQL injection testing. It is used since it is easy to learn and to use, compared to other databases. Furthermore, it is considered as highly available secure database [32]. Besides that, many recent studies showed that it is one of the most commonly used databases. For instance, a recent survey in 2018 conducted by Stack Overflow's developers showed that it has gained 58.7 percent market share the recent years [46]. On the other hand, for NoSQL injection testing purposes MongoDB database is used in the prototype. Using it in the prototype comes from its raise of popularity among other NoSQl databases. In fact, it is ranked as fifth among the 10 most popular databases in 2017 according to db-engines.com popularity ranking[10].

We expect to obtain the objectives of this research by testing our proposed work through experiments and simulation. We also wish that our verified experiment results provide valuable references to web developer, websites and security community.

Chapter 5

Framework Design

5.1 Introduction

In this section we will provide an overview of the proposed framework. The framework is focusing on preventing and detecting SQLI or NoSQLI attacks mainly for the embedded queries in the web applications. As can be illustrated from the studied works many of those works achieve their goal in detecting and preventing those types of injections by focusing on the integrity of the used queries such as [42], [11], [50], [30], [23], and [1]. But to the best of our knowledge none of those works use Blockchain technology to save the valid queries, and work as filter layer to identify any malicious attempt to inject the database. Despite of that, many works such as [55], [29], [3], [12], [33], and [9] used Blockchain for enhancing security issues in many computer security related fields.

This research provides a framework that uses query integrity based approach to achieve the detection and prevention for many types of such attacks using a hybrid approach that combines the query parsing approach with the Blockchain technology. The Blockchain is used to save the valid queries that are used as legal queries.

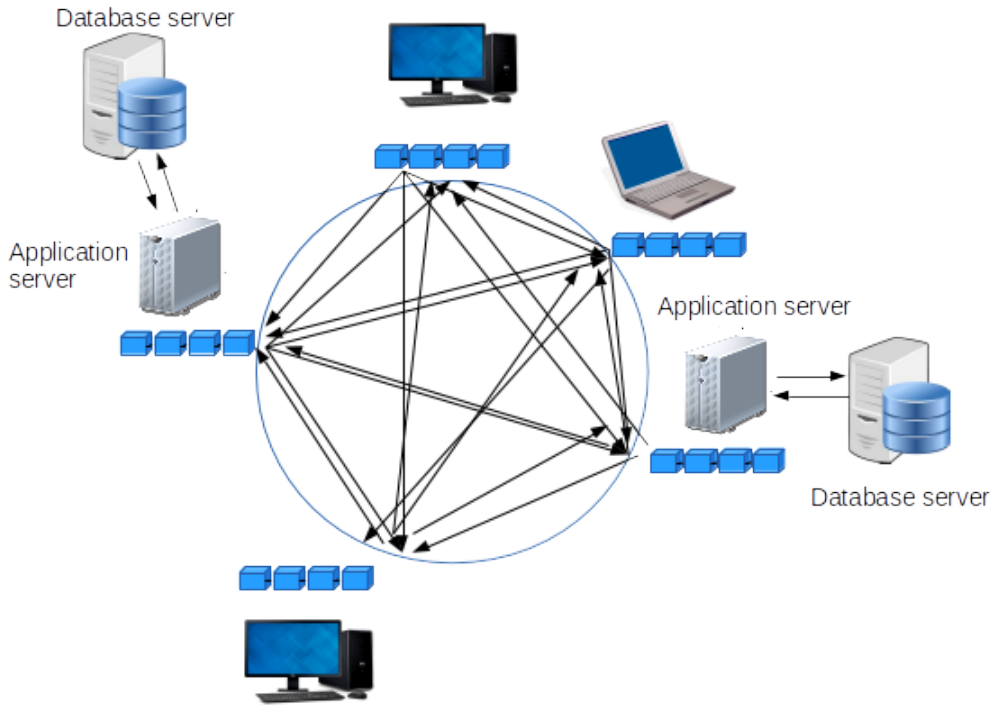


Figure 5-1: General architecture of the proposed framework

5.2 Framework architecture: Design concept

Figure 5.1 shows general overview of the framework architecture. As it can be illustrated in the figure any application server should participate in the Blockchain network in order to get the benefits of using such framework. Using Blockchain in the framework is for saving data that can be used to detect and prevent database injection attacks. The Blockchain network can be public or private network. A public Blockchain which is a Blockchain that operating on a distributed network of nodes with data that anyone in the world can read, and validate data. On the other hand, the private Blockchain offers the same level of data integrity as a public Blockchain, but does not need the same level of effort to maintain or either initialize the network of Blockchain as well as, they does not rely on a distributed public network of nodes, they only deal with trusted nodes that participate in that private network [6]. In addition to that, the private Blockchain can easily generate currency which can be

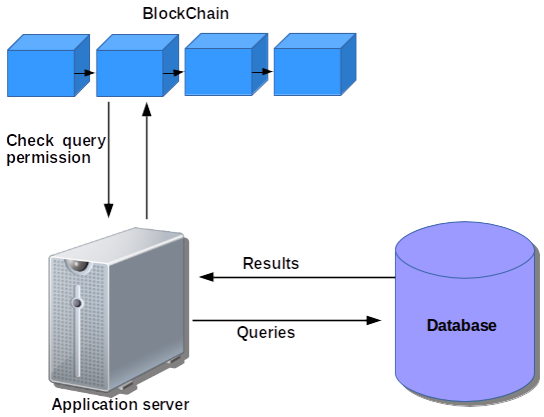


Figure 5-2: Application server with Blockchain

used to initiate and deploy transactions in that Blockchain.

A closer look to figure 5-2 that shows the application server with the Blockchain. In this figure the Blockchain is used beside the application server that contains the web application. As can be illustrated, any attempt to deal with the database should be first checked by the Blockchain to identify if it's an attack attempt or not. The decision of whether establishing a connection with database or not will be based on the returned result from the Blockchain. So no connection will be established directly to the database, any connection should be proceeding by permission from the Blockchain.

5.2.1 Query tokenizer

The query string for any application has two states. The first state is called the rest state which describes the query string that has been written by the developer, and resides inside the application. It could contains some variables or parameters that will be filled at later time when the application is in run time phase. The second state of the query string is the run-time state which is the query that at the real run-time of the application, it is the query that is running on the database at any moment of time. The main goal for the Tokenizer is to split the query in to individual tokens as showen in figure 5-3, then select some of those tokens and add them together in a new query string. Choosing some tokens from the query to add them in the new query

Original query:

select * from table where name ='ddd';

After split the query in to Tokens:

[select] [*] [from] [table] [where] [name] [=] ['] [ddd] ['] [;]

Figure 5-3: Tokenization example: splitting query into tokens

string will be based on a predefined table that contains some tokens. This framework provides a string parser which is called query Tokenizer. Figure 5-4 shows the flow chart of the tokenization process. This Tokenizer is used to split the query into tokens based on a table of tokens. That table should have the tokens which consist of the reserved words of the used query language or database engine for instance, MySQL, MongoDB, or any other used engine. In addition to that, the table could contain any attribute, table name or collection name that used inside the created or used database. So, table should have all reserved words for the used language, also it could have any other attribute or table name that the developer wants to add. The query at rest state will be sent to the Tokenizer. The Tokenizer will extract the tokens of query based on the table of tokens that saves the intended tokens. Then the tokens will be appended into one new query string. After that this new string will be hashed in SHA256 cryptographic hash function. The same process will be used later on the run-time queries, note that those queries contains the real values for the used variables or parameters so those values will be ignored by the Tokenizer unless if those parameters contain any modification for the original query string, in that case those new added commands that were added to the query will be considered as part of the query string. To avoid reading the values of the parameters -right hand side- as tokens, the tokenization process is performed by detecting the space, single quotes and double dashes as an example of SQL databases, and some other symbols like colon and $ for MongoDB as an example of NoSQL databases.

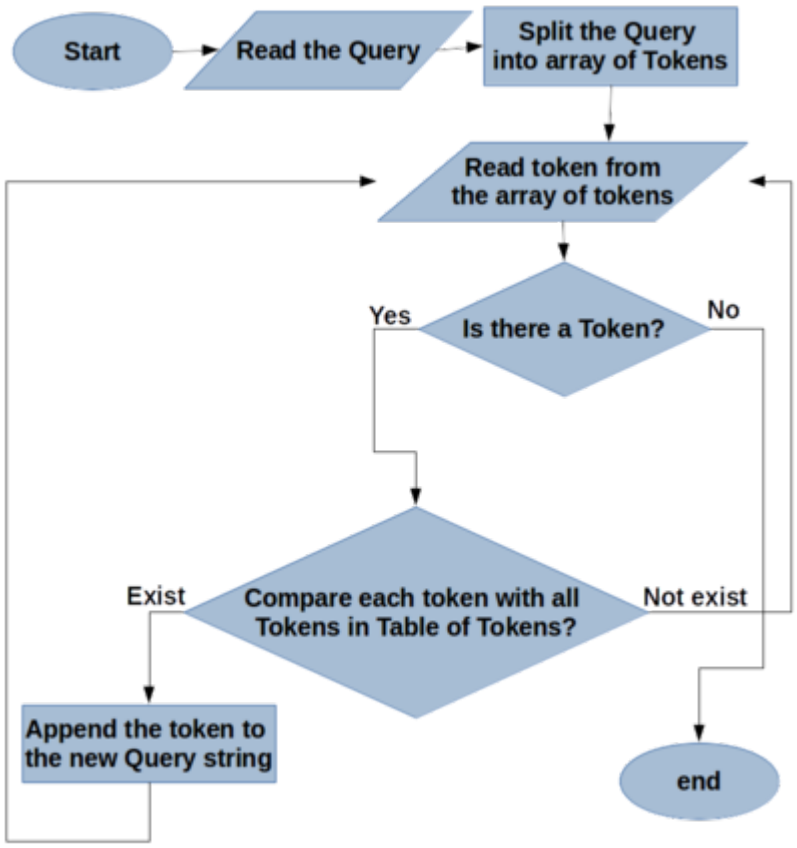


Figure 5-4: Flow chart for query tokenizer stages

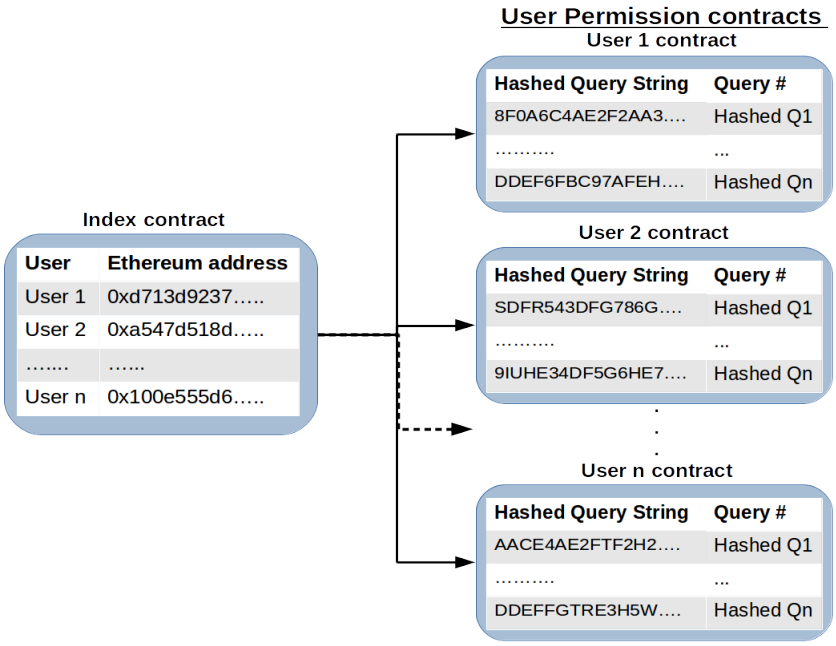


Figure 5-5: Smart contracts architecture

5.2.2 Smart contracts architecture

The Ethereum smart contracts are used to save the intended queries with their associated users. As can be demonstrated from figure 5-5 which has two types of smart contracts that are used in the proposed work. The first contract is the Index contract, this smart contract is used for saving the intended users of the application, those users that have the right or authority to deal with the database of the application that need to secure against data inject attacks. The second contract is the User Permission contract, this smart contract is used for saving the hashed strings of the queries that are allowed for a specific types of users. More details for those smart contracts will be seen in the following sections.

The predominantly used language for writing smart contracts in Ethereum is Solidity programming language. Which is a JavaScript similar language developed specifically for writing smart contracts[44]. Solidity language is selected in this work due to the previous reasons.

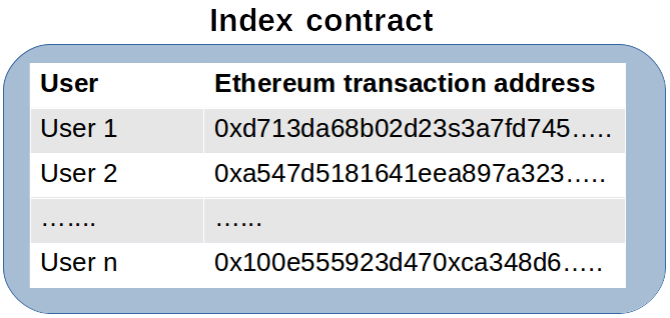


Figure 5-6: Index contract architecture

Index contract

This is a smart contract in Ethereum which is used to save the users of the application in a mapper. For any applications there are users, those users can be classified based on some rules for instance, if a registration portal for a university is chose, it has many users that are classified in groups such as students, teachers, department chairmen, registration employees, and many other classifications. So all of these groups that are mentioned in the example should be saved in the Index contract. This contract is a transaction inside Ethereum Blockchain. It has an Ethereum address in the Ethereum network that can be used to deal with that contract. The smart contract can be initiated using an Ethereum account. In Index contract this account will be the only account that has the authority to use the contract in order to save, retrieve, or update the data. Restricting the usability of the Index contract to one account has a great positive effect in securing the data that the contract has. The owner of the contract is the Ethereum account address that initiates it.

Figure 5-6 displays the general structure of index contract. As shown in the figure the used mapper maps the user name to an Ethereum address. This address is an address of another type of smart contract which is the Permission contract. More details of this contract will be found in the next section.

User Permission contract

In addition to the aforementioned smart contract, there is another type of smart contract that is used to save, retrieve, the hashed queries which is the User Permission

smart contract. it is created using the same Ethereum account that used to create the Index contract as a result it is the only account that can manipulate and use this contact.

For each user that is saved in the Index contract, there is a User Permission contract. So that the permitted queries for any user should be firstly specified by the administrator of the server, or even by the application developer. After that those queries should be parsed using the Tokenizer and should be stored in the intended User Permission contract that is specified for that user.

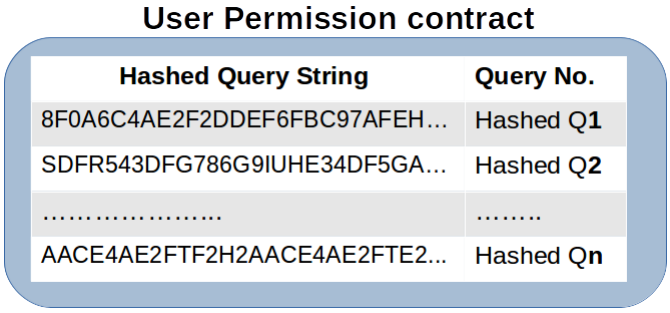


Figure 5-7: User Permission contract architecture

As figure 5-7 shows the queries should be saved as hashed string inside a mapper that resides in a User Permission contract. In this work SHA256 cryptographic hash function is used. Ethereum Blockchain is already using SHA256 in the Proof of Work algorithm [53]. The hashing process can be done either in the application server or in the the User Permission contract itself. In this work the hashing is done inside the application server then will be send to the contract in order to save it. Choosing the application to do this task is due to cost and response time reasons. This work focusing on the process of hashing the queries in the reading rather than saving phase, since the hashed queries will be saved once but it will be retrieved multiple times.

5.2.3 Structure of participant nodes in the framework

This framework consists of many nodes that are connected to Blockchain network, those nodes can be participated in the global or the private Blockchain. The nodes

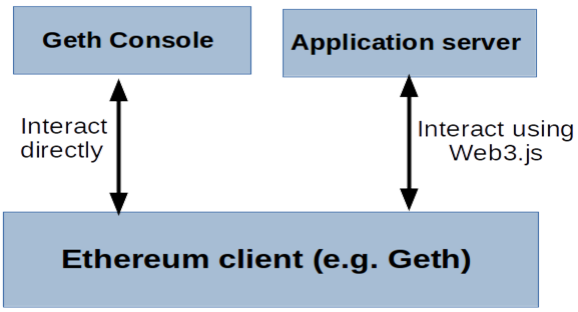


Figure 5-8: Interact with Ethereum client e.g. Geth

might be the application server or any other computer. A prototype for simulation purposes is done, this prototype uses the private Blockchain. The node in that private Blockchain should start with a custom block called Geneses block, which is the first block in the Blockchain network or block 0. The participant nodes cloud be the servers that aim to use the Blockchain to prevent the database injections and this node will be called *primary node* in the proposed framework. Another nodes can be participated to work as miners in the Blockchain. In addition to any other nodes that are participating in the Blockchain network to maintain the network. Each node will have the same copy of the Blockchain data. The Blockchain works as database that contains the contracts, inside those contracts the rules and permissions that restrict the use of the real database.

Primary node structure

This node is the basic component of the proposed work. Since in the used Ethereum Blockchain infrastructure the participant nodes are peers that are connected as peer to peer nodes. The primary node handles a broad set of tasks, such as connect to the Ethereum Blockchain network, save a local copy of the Blockchain, sending and receiving transactions, and encoding the transactions. In the implantation phase for initiating Private Ethereum Blockchain network or being joined to it, the node should have Geth program which serves as a client node for the Ethereum Blockchain, by using this program the user can mine Ether and create software which runs on the Ethereum Virtual Machine (EVM).

In addition to that, the primary node which represents the application server in this framework should have a way to interact with the Ethereum node. An intermediate layer is chosen to interacts with the Geth program (Ethereum client) which is a JavaScript like code one. This layer makes Remote Procedure Call (RPC) over HTTP via Web3.js library. Web3.js [51], which is part of Node.js, that is a collection of modules which allow user to speak or interact with a local or remote Ethereum using JavaScript and JavaScript run-time environment. As illustrated in figure 5-8, the primary node have additional functionality over other nodes, which is focusing on how interfacing with Ethereum client Geth for creating, reading from and deploying contracts or other ethereum transactions.

Furthermore, to implement smart contract in Ethereum, the primary node should use Solidity compiler such as Remix Browser-based IDE compiler which can be integrated with Ethereum client. Also another options can be used to compile the smart contract code of Solidity such as Solc compiler. In general, contracts are rendered as sets of functions in Solidity, then the solidity compilers turns code into Ethereum Virtual Machine (EVM) bytecode, which then can be sent to the Ethereum network as transaction to be deployed [8].

Other participant nodes structure

As shown in 5.1 the nodes that participate in Blockchain to maintain the network are any computer that works as Blockchain client by installing the program that intends to connect that node with the Blockchain. Those nodes cloud be used as miners to mine the transactions. In this work the used Blockchain is the Ethereum Blockchain, so Geth program have to be installed on all participant nodes.

5.2.4 Smart contracts implementation and costs

When there is a need to executing a smart contract code or update the state of Ethereum Blockchain, it should be remembered that it is far different than doing the same thing on a simple server. Smart contracts are executed on Blockchain EVM. The

EVM is virtual machine that operates based on bytecode language. Since Ethereum is a decentralized application all of its nodes have the same EVM specification. The smart contract code and other transaction are executed in a distributed manner across all the nodes that validate the transactions which called miner nodes.

In EVM Each low level operation is called opcode. For instance, getting balance of an account, adding two integers, or multiply two integers are all called opcodes. Each opcode has a number associated with it called Gas. The Gas is an abstract number that represents the complexity of opcode operation.

This framework can be implemented either in public or private Blockchain. In the following subsections both scenarios of implementation are discussed.

Using of Public Blockchain

A public Blockchain is a Blockchain operating on a distributed network of nodes with data that anyone in the world can read and validate. The nodes that are participating in the network are preselected and can approve the transaction. As a result implementing the smart contract transaction on public Ethereum will be secure since each node in the Blockchain system has copy of all the data. On the other hand, fees need to be payed for deploying the smart contracts or any other transaction.

Operation	Gas estimation	Ether	US dollars
Creating index contract	501993 gas	0.0100399	4.54807
Creating User Permission contract	628228 gas	0.0125646	5.69176
Add new user	47874 gas	0.0009575	0.43375
Add a Hashed Query	73500 gas	0.00147	0.66591
Delete a Hashed Query	73227 gas	0.0014645	0.66342

Table 5.1: Transaction fees (gas costs)

An important issue that should be considered when using the public Ethereum Blockchain, this issue is about the estimations of gas that is needed to deploy the proposed contracts. The estimation of gas depends on the functions that the contracts

have. Generally, the estimations of gas will depend on the structure of the function itself, also it depends on the length of the actual parameters that the function receives. Table 5.1 gives estimations for using the two proposed contracts. The Gas estimations are calculated using Remix tool.

In order to find the estimation gas cost for the used transactions or operations the researches looked up the gas price form (Eth Gas Station) [13]. At the time of writing this research, the median gas price was in the realm of 20 Gwei from (Eth Gas Station). Taking into account that 1 Ether equals 10^9 Gwei. After that, this price used to find out the transaction price is US dollar and Ether currency. This done by multiplying the Gas price by the estimated amount of gas for each operation or transaction. The estimated results is shown in table 5.1. Note that use less Gas price can be used but this will be in trade-off the mean time to deploy and confirm the transaction. Consequently, less Gas price means that the mean time of the transaction to be deployed will be reasonably increased. Since usually those operations will take place at developing time it will not be a problem to use a cheaper Gas price to do those operations.

On the other hand, this research intensely focus on the transactions or operations that are used for reading data from Blockchain or what are called the call methods. The call method is used to call a contract's method on EVM with the current state to retrieve the hashed queries or the uses in this framework. It doesn't broadcast a transaction. To read data, there is no need to broadcast the call method to Blockchain because there will be a local copy of Blockchain to read from. This will offer two advantages for the proposed work. First the reading time will be at a very high speed. The second advantage is that there will be no cost for reading transaction because of using the local copy of Blockchain.

Using of Private Blockchain

Private Blockchain offers the same level of data integrity like what public Blockchain offers and still offers a high degree of security, but with less nodes in the network comparing with public Blockchain, which means data can be read or stored at high speed

between and within the nodes. Like Public Blockchain, data in private Blockchain can be read from any node in the network if the node is accessible. Moreover, the local currency will be easily gained by miners. As a consequence, there is no need to pay for deploying the smart contracts or any other transactions. Note that adding new nodes to the system is quick and easy compared with adding a client to a server. The experiments or simulations for this research is implemented using Private Ethereum Blockchain.

Chapter 6

Implementation, Results and Discussion

6.1 Introduction

This chapter shows the practical approach of the implementation by applying SQL and NOSQL injection attacks to the used prototypes for web applications.

In order to show the effectiveness of the proposed framework, it is adapted with a testing prototypes, then these prototypes are executed in the localhost serve. The injection attacks are applied with and without adapting the framework. Results are discussed at the end of this chapter.

The rest of this chapter is organized as follows. Section 6.2 introduce the testing prototype which consists of two web applications that are used SQL and NoSQL databases. Section 6.3 describes the steps of implementing the proposed framework for the used prototypes. Section 6.4 shows the practical experiment of attacking the used prototypes and discussing the results. Section 6.5 studies and evaluates the response time for the proposed contracts in Ethereum Blockchain. Section 6.6 concludes the experiment results.

```
$username     = $_POST['username'];
$pass          = md5($_POST['password']);

$sql_query = "select * from login where username ='$username'
              and password ='$pass' and status ='ACTIVE' ";
```

Figure 6-1: The login SQL query for Student Online Voting System

6.2 Experimental tools

This section gives a brief introduction about the tools or prototypes that are used to test the work in both SQL and NoSQL cases. Two application prototypes are used as tools to test the proposed framework one for SQL and the other is for NoSQL. The following sections show more details about the used prototypes.

6.2.1 Experimental SQL tool

To test the framework in the case of SQL injection, the first used prototype is a PHP application web with MySQL database. This prototype is for a Student Online Voting System. It has two types of users, the admin user, and the student user. Each type of users have specific privileges and rules. For instance, the admin can login into administrator account and also can search for any user that participates in the system, along with some other features. The student can login and vote for the candidate he wants.

Figure 6-1 shows the login code which uses the POST method in order to send data to the server. The searching process uses the GET method in order to request data from a specified resource as shown in figure 6-2. These two methods have vulnerabilities that can be exploited to make SQL injection and retrieve the database content. Besides this, the application has no security tools or any other methods inside its code to face any type of SQL injections.

```
$var = @$_GET['search'];
$trimmed = trim($var);

$sql_query = "select distinct  * from student
              where username like '%$trimmed%' ";
```

Figure 6-2: The search SQL query for Student Online Voting System

6.2.2 Experimental NoSQL tool

The same software or prototype is used in NoSQL injection testing phase, but the used database in this case is MongoDB. The POST method is used In the login phase, and the GET method is used in the search page which is an admin page that used to search for the system users. Unlike SQL databases, NoSQL databases have no standard query language. For instance, Mongodb has it is own syntax, besides that it uses JavaScript as its main query language. To connect to the PHP project with Mongodb a special PHP driver needs to be installed. In this prototype the server side JavaScript is used as the languages for querying the database.

```
$username    = $_POST['username'];
$pass         =md5($_POST['password']);
$query_condition="function(){return(this.username)=='$username'
                  &&    (this.password)== '$pass'   }";

$code = new MongoDB\BSON\Javascript( $query_condition );
$query=new MongoDB\Driver\Query(['$where'=>$code]);
```

Figure 6-3: The login NoSQL query for Student Online Voting System

```
$var = @$_GET['search'];
$trimmed_username = trim($var)
$query_condition=" function() {this.username == '$trimmed_username'}";

$code = new MongoDB\BSON\Javascript( $query_condition);
$query = new MongoDB\Driver\Query(['$where'=>$code]);
```

Figure 6-4: The search NoSQL query for Student Online Voting System

Figure 6-3 shows the used query for login as well as figure 6-4 shows the syntax of the search query in the used prototype. As can be demonstrated from these figures the used code for the queries is the JavaScript language code, this code is sent to Mongodb in order to execute it.

6.3 Implementation of the framework

Implementation of the proposed framework is done for both prototypes. More details of implementations are demonstrated in this section. Implementing the framework

for NoSQL prototype is somehow similar to SQL, this comes from the fact that the experiments use the same prototype with different databases.

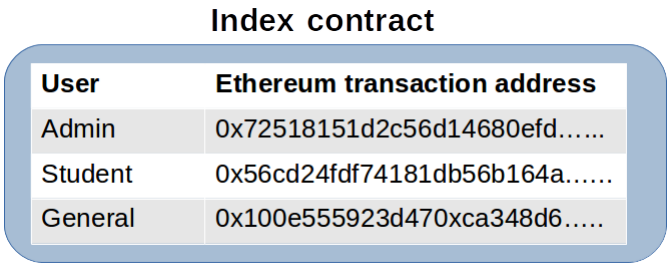


Figure 6-5: Index smart contract for Student Online Voting System

At the beginning of the experiments a private Ethereum Blockchain network is initialized. Then the proposed contracts also is deployed in that Blockchain. Using the Web3.js a new web application is created to interact with Ethereum Blockchain for adding new users with their associated hashed queries to Index and User Permission smart contacts respectively. The first smart contract which is the Index smart contract as shown in figure 6-5 contained two types of users which the used prototype has, the Admin user, the student user, and additionally another user is added that called the General user, it is used to describe the state of any user that opens the system but still not login. Each user is saved in this contract with the address of his associated Ethereum User Permission contract as shown in figure 6-6. This address is like a pointer that points to the Ethereum address of the User Permission smart contract inside the Ethereum Blockchain. The User Permission smart contract for each user is used to save the hashed queries for that user.

Secondly, the proposed Tokenizer is added to both prototypes as a PHP class. The class members are the query string, the table of tokens, the set query function, the fill table of tokens function, and the function of the tokenization process. At this stage, the table of tokens is filled with the reserved words of MySQL in the first case where the SQL is used. For NoSQL case the filled words or tokens are the JavaScript words that are used to form query command syntax for Mongodb because it is easily adapted by the proposed Tokenizer.

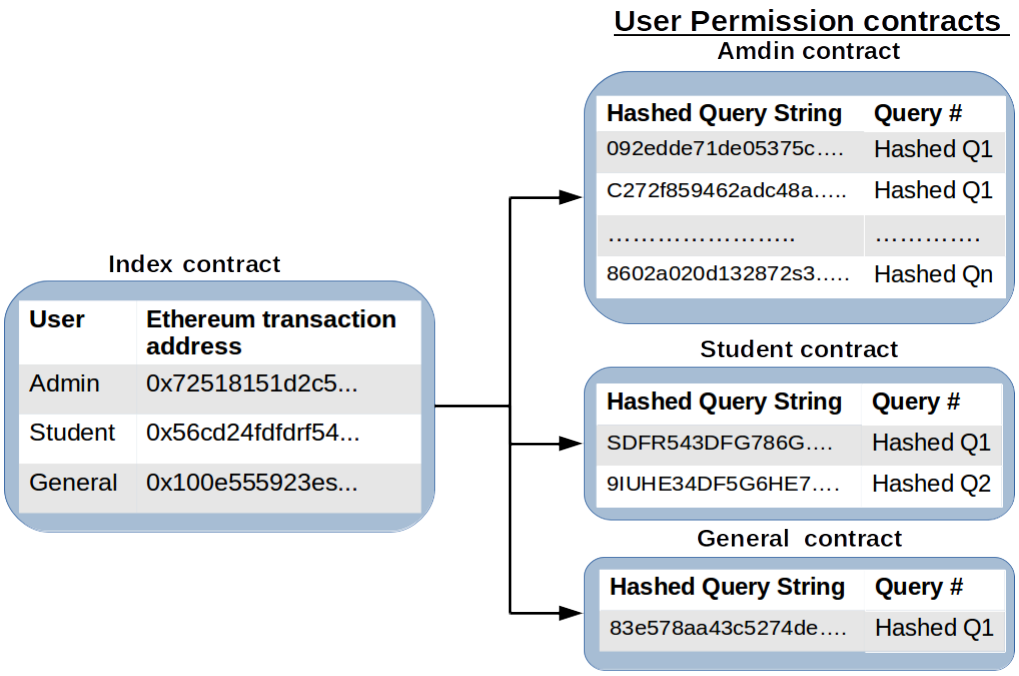


Figure 6-6: Smart contracts architecture for Student Online Voting System

As shown in figure 6-6, each user has a specific set of hashed queries. These queries extracted from the queries at rest state of the application. Rest state query describes the state of original query that system has. So each rest state query needs to be sent to the Tokenizer by the developer of the system. Then, result is hashed in SHA256. After that, it is sent to the corresponding User Permission smart contract based on its user. For example, the General user has only one query which is the login query, the student user has two queries in this system, one is for voting process, and the other is for changing the user details such as the password, and so on for the Admin user.

The main concerns of this research is the run-time query, it is used to describe the query that need to be executed during the runing time of the system. Whenever any attempt to execute a run-time query or to make a connection with the database, the Tokenizer class should be used to make a new object. Then the run-time query is sent

to this object in order to make the tokenization process for it. After that the result from that query is hashed using the SHA256 cryptography algorithm. Finally, this result is sent to the Blockchain to see if it exists inside the intended User Permission contract. The decision to establish a connection for executing run-time query is based on the returned result from the Ethereum Blockchain. In this experiment the application is connected directly to Private Ethereum. The connection is made between them using Web3.php which is a collection of libraries that allow the application to interact with a local or remote Ethereum node.

6.4 Experiment Results and Discussion

To test the proposed framework two experiments are conducted based on the type of used database SQL or NoSQL. This is done by using the aforementioned prototypes. The injection attack is done before and after using the framework. The following sections describes the experiments results with more details based on the type of used database.

6.4.1 SQL Experiment Results and Discussion

Before adapting the proposed framework, The Sqlmap tool is used in order to test the Student Online Voting System prototype by making SQLIA penetration tests. The results of testing shown in figure 6-7, which displays a screen shot for the Log file that shows the attack results. As illustrated from the figure, the system can be attacked using this tool and making the database totally insecure. Consequently, the data could be retrieved or altered. From that the Log file all database names can be seen. As a result, all names are retrieved even if they did not belong to the voting system, This happened because the connection between the voting system and MySQL is created with vulnerabilities to the SQL injection. These vulnerabilities are exploited to attack the whole database.

```
sqlmap identified the following injection point(s) with a total of 548 HTTP(s) requests:
---
Parameter: username (POST)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause
    Payload: username=-4401" OR 9451=9451-- rTof&password=pass&login-php-submit-button=Login

    Type: error-based
    Title: MySQL >= 5.0 OR error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
    Payload: username=user" OR (SELECT 1889 FROM(SELECT COUNT(*),CONCAT(0x71717a7671,(SELECT
             (ELT(1889=1889,1))),0x7171627a71,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS
             GROUP BY x)a)-- wznD&password=pass&login-php-submit-button=Login

    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 OR time-based blind
    Payload: username=user" OR SLEEP(5)-- EMYk&password=pass&login-php-submit-button=Login
---
web server operating system: Linux Ubuntu 16.04 or 16.10 (yakkety or xenial)
web application technology: Apache 2.4.18, PHP 7.0.30
back-end DBMS: MySQL >= 5.0
available databases [9]:
[*] car
[*] information_schema
[*] mysql
[*] performance_schema
[*] scuola
[*] simple
[*] student
[*] sys
[*] test
```

Figure 6-7: Screen shot for the log file of SQLIA using Sqlmap tool

Sqlmap tool by default tests only The GET parameter. As mentioned before, the system used the POST parameters in the login process. The default state of Sqlmap is changed to perform POST request injection.

In additions to Sqlmap tool, manual injection attacks are used. Table 6.1 shows the list of used injections. It shows the type of injection, the targeted query, the Payload string, and the result of the attack.

Table 6.1 illustrated that the system has vulnerabilities to many types of SQLIAs as listed in the table. These vulnerabilities made it easy to attack the database and tampered with it. In the Union attack of the manual injection seven attributes are used because the number of expected results from the original searching query in the used system are seven.

Type of injection	Payload example	Attack result before framework	Query	Attack result after framework
Tautology	'or '1'='1';– -	Succeeded	Login	Failed
Tautology	'-0‖'	Succeeded	Login	Succeeded
Order by	1' order by 10– -	Succeeded	GET	Failed
Union	1' UNION select 1,@@version,3,4,5,6,7– -	Succeeded	Search	Failed
Union	1' UNION select 1,table-name,3,4,5,6,7 from information-schema.tables– -	Succeeded	Search	Failed

Table 6.1: List of manual SQLI attacks for login and search queries in the testing system

After adapting the proposed framework the same attacks are repeated using Sqlmap tool and the manual attacks that are listed in table 6.1. The results showed that Sqlmap didn't find any vulnerability in the system. On the other hand, the second manual Tautology attacks that listed in table 6.1 succeeded to attack the system although the framework did not detect that, because the Tokenizer didn't recognize the token (-0‖) as part of the query. As a result, when checking that query from Blockchain without that token the result is positive, which means that the attack succeeded and bypassed.

To solve this problem the token is added to the table of tokens, so the Tokenizer will recognize it. As can be noted, the table of tokens should not be limited to the reserved words of the used SQL engine. In this work the table of tokens gives developers the ability of adding new tokens to the table, which makes the Tokenizer more flexible to deal with any new type of attacks.

6.4.2 NoSQL Experiment Results and Discussion

Table 6.2 shows many types of NoSQL injection that successfully attacked the system. The table shows the type of injection attack, payload which is the syntax that used for the attack, the result of the used attack, and the targeted query which the attack is deployed on, for instance Denial of Service attack is used which results in infinite loop that consume the resources of the server. Based on the results listed in the table, it can be inferred that Mongodb is not immune to injection attacks.

When the framework is adapted the same attacks repeated again, the results showed that none of these attacks worked. This does not means that the framework can cover all types of injection attacks in Mongodb without modification to the Tokenizer, since sometimes the queries parameters in Mongodb can be in the form of associative arrays.

6.5 Blockchain time response evaluation

Since this work depends on Ethereum smart contracts, a new experiment is conducted to evaluate the response time for reading from both proposed contracts. Table 6.3 shows the response time in milliseconds for reading data from Index and User Permission smart contracts. The used server in this experiment is a desktop with 4 GB DDR3 RAM and an Intel i3 processor. The results showed that the speed of response time is relatively acceptable with the low capability of the used computer in the experiment. On the other hand, in the real environment of application servers the hardware specifications will be much more than the used machine in our experiment. This will positively affect in enhancing the response time.

Table 6.3 shows the response time of the used contracts. It can be inferred from the table that the response time of Index contract is relatively less than that of the User permission contract. This difference in response time is due to that the Index contract has less complex operations than the other contract. The reading operations are done using the local copy of Ethereum Blockchain. This means that the results will not be significantly affected in case of using public Ethereum.

Type of injection	Payload example	Attack result before framework	Query	Attack result after framework
Tautology	1' || this.username.match(/.*/) ;'	Succeeded	Search	Failed
Tautology	Admin'|| '1'=='1	Succeeded	Login	Failed
Comment Injection Attack	a'|| 'true'; $comment='	Succeeded	Login	Failed
Timing-based Blind attack	aaa"|| sleep(1000);'	Succeeded	Search	Failed
Timing-based Blind attack	1';it=new Date();dopt=new Date();while(pt-it<500);'	Succeeded	Search	Failed
Blind attack	8';return(false);var foo='bar	Succeeded	Search	Failed
Blind attack	Mohammed' ; $comment:'successful MongoDB injection';'	Succeeded	Search	Failed
Denial of Service attack	1'; while(1); var foo='bar	Succeeded	Search	Failed

Table 6.2: List of manual NoSQLI attacks for login and search queries in the testing system

Number of transactions	User Permission contract Average time (ms)	Index contract Average time (ms)
1	5.87	4.14
100	150	126
200	317	230
300	477	340
400	606	497
500	715	575

Table 6.3: Response time in milliseconds for reading data from the proposed smart contracts

6.6 Conclusion

After the framework is adapting in both types of the used prototypes, the results showed that the majority of the used injection attacks did not work. This framework shows that it is reliable, effective, and can be used to detect and prevent the SQLIA for the embedded queries. Furthermore, using the authorizations rules over queries is a good technique in facing SQLI and NoSQLI attacks. Moreover, using the enhanced tokenizer shoed that it can prvent and detect many types of injection attacks in both SQL and NoSQL databases. In additions, using Blockchain will add more security in facing such attacks, and will not affect the total response time of the server because based on results Blockchain has a good response time.

Chapter 7

Conclusion and Future Work

This thesis proposed a general framework that handle SQLI and Many types of NoSQLI in web applications. It has two approaches. The first approach designed and implemented a query syntax parser called Tokenizer. It intercepts the entire request coming from the client side, parsing the query text, and then extracts the intended tokens based on a prepared table that contains a collection of intended tokens after that the result will be hashed using SHA256 algorithm. The second approach is Blockchain based approach. This approach is based on Ethereum Blockchain smart contract which used to save the original quires that are permitted to be used as legal queries alongside with the types of it's authorized user.

The results showed that using authorization rules over the used queries can positively impact in eliminating database SQL and NoSQL injection attacks which answers the first research question. This done by checking the list of permitted queries for any user before open the connection with the database. This step is done inside the proposed smart contracts by sending the intended user and query to Ethereum in order to check if that user is authorized to use the query or not.

Furthermore, the proposed work shows a good detection ratio with only one false positive injection attack, which is considered to be a good accuracy achievement. The efficiency of the Tokenizer is based on the table of tokens. The proposed work shows that it can detect and prevent many types of injection in both SQL and NoSQL web applications. Over and above, it shows that the proposed Tokenizer can be modified

to detect new types of injecting by adding the malicious token to the table of tokens which provides an answer for the second research question.

Moreover, using Blockchain has many advantages to meet the continuous on demand for seeking of isolate, secure, and immutable place to save the credential information. The sensitive information in the proposed work is the hashed queries and the types of users, Index and User Permission proposed contracts used for this purpose. The results showed that using Blockchain offers new technique that is secure to save this sensitive information among other used methods which partially answer the third research question and the response time for reading data from the proposed contracts provides an acceptable results. In those contracts we used mapper data-type instead of arrays to avoid the iteration process that used to check the authorization rules. Iteration or looping needs to be limited, not only for speed, but potentially for security issues also for instance, to avoid Denial of service attack. As a result Blockchain can also offer speed efficiency along side with security. Mapper works as key value concept, in our case the key is a legal hashed query and the value is a time stamped value, this value will be used in the web application to prove that the used run-time query is not changed or injected.

As future work, an evaluation for the proposed work using different types of NoSQL databases is needed. Moreover, the Tokenizer need to be improved so as to accommodate many types of NoSQL query languages due the absence of a common query language in NoSQL databases. On the other hand, the consumed time should be investigated more to seek for a better response time in order to enhance the framework efficiency. Finally, more security testing is needed to test the libraries that are used for interacting with Blockchain client.

Bibliography

[1] Dibyendu Aich. Secure query processing by blocking sql injection. Master's thesis, National Institute of Technology Rourkela, 2009.

[2] Shaukat Ali, SK Shahzad, and Huma Javed. Sqlipa: An authentication mechanism against sql injection. *European Journal of Scientific Research*, 38(4):604–611, 2009.

[3] Louise Axon. Pb-pki: a privacy-aware blockchain-based pki. *SCITEPRESS*, 6, 2016.

[4] Indrani Balasundaram and E Ramaraj. An authentication mechanism to prevent sql injection attacks. *International Journal of Computer Application (IJCA)*, 19(1):30–33, 2011.

[5] Lindsay Bassett. *Introduction to JavaScript Object Notation: A to-the-point Guide to JSON.* " O'Reilly Media, Inc.", 2015.

[6] Greg Bohl and John F Dickson. Private blockchains in automotive safety white paper. *Automotive Software Technologies For The Connected Car*, 2017.

[7] Vitalik Buterin. Medium: a proof of stake design philosophy. Available: `https://medium.com/@VitalikButerin/a-proof-of-stake-design-philosophy-506585978d51`, 2016. [Accessed: 2018-6-29].

[8] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *https://www.ethereum.org/ pdfs/EthereumWhitePaper.pdf/*, 2014.

[9] Jordi Cucurull and Jordi Puiggalí. Distributed immutabilization of secure logs. In *International Workshop on Security and Trust Management*, pages 122–137. Springer, 2016.

[10] Db-engines ranking web site @ONLINE. Available: `https://db-engines.com/en/ranking`, 2017. [Accessed: 10-12-2017].

[11] Ahmed Eassa, Omar H Al-Tarawneh, Hazem El-Bakry, and Ahmed Salama. Nosql racket: A testing tool for detecting nosql injection attacks in web applications. *International Journal of Advanced Computer Science and Applications*, 8:614–622, 11 2017.

[12] Ariel Ekblaw, Asaph Azaria, John D Halamka, and Andrew Lippman. A case study for blockchain in healthcare:"medrec" prototype for electronic health records and medical research data. In *Proceedings of IEEE Open & Big Data Conference*, volume 13, page 13, 2016.

[13] ETH Gas Station: gas-time-price estimator for transactions. Available: `https://ethgasstation.info/`, 2018. [Accessed: 2018-6-29].

[14] William G Halfond, Jeremy Viegas, Alessandro Orso, et al. A classification of sql-injection attacks and countermeasures. In *Proceedings of the IEEE International Symposium on Secure Software Engineering*, volume 1, pages 13–15. IEEE, 2006.

[15] William GJ Halfond and Alessandro Orso. Amnesia: analysis and monitoring for neutralizing sql-injection attacks. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 174–183. ACM, 2005.

[16] Haskell purely functional programming language home page@ONLINE. Available: `https://www.haskell.org/`, 2014. [Accessed: 10-12-2017].

[17] Boyu Hou, Kai Qian, Lei Li, Yong Shi, Lixin Tao, and Jigang Liu. Mongodb nosql injection analysis and detection. In *Cyber Security and Cloud Computing (CSCloud), 2016 IEEE 3rd International Conference on*, pages 75–78. IEEE, 2016.

[18] Gaspar Modelo Howard, Christopher N Gutierrez, Fahad A Arshad, Saurabh Bagchi, and Yuan Qi. psigene: Webcrawling to generalize sql injection signatures. In *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*, pages 45–56. IEEE, 2014.

[19] Imperva. imperva web application attack report @ONLINE. Available: `http://www.imperva.com/docs/HII_Web_Application_Attack_Report_Ed4.pdf/`. [Accessed: 10-12-2017].

[20] Markus Jakobsson and Ari Juels. Proofs of work and bread pudding protocols. In *Secure Information Networks*, pages 258–272. Springer, 1999.

[21] B. sullivan. server-side javascript injection @ONLINE. Available: `https://media.blackhat.com/bh-us-11/Sullivan/BH_US_11_Sullivan_Server_Side_WP.pdf`. [Accessed: 20-12-2017].

[22] Swathy Joseph and KP Jevitha. An automata based approach for the prevention of nosql injections. In *International Symposium on Security in Computing and Communication*, pages 538–546. Springer, 2015.

[23] Mi-Yeon Kim and Dong Hoon Lee. Data-mining based sql injection attack detection using internal query trees. *expert systems with applications*, 41(11):5416–5430, 2014.

[24] Diallo Abdoulaye Kindy and Al-Sakib Khan Pathan. A survey on sql injection: Vulnerabilities, attacks, and prevention techniques. In *Consumer Electronics (ISCE), 2011 IEEE 15th International Symposium on*, pages 468–471. IEEE, 2011.

[25] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 369–378. Springer, 1987.

[26] ABM Moniruzzaman and Syed Akhter Hossain. Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. *arXiv preprint arXiv:1307.0191*, 2013.

[27] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Available: `https:www.bitcoin.org`, 2008. [Accessed: 1-12-2017].

[28] Netcraft web server survey. Available: `https://news.netcraft.com/archives/2018/01/19/january-2018-web-server-survey.html`, 2018. [Accessed: 22-4-2018].

[29] Charles Noyes. Bitav: Fast anti-malware by distributed blockchain consensus and feedforward scanning. *Arxiv preprint arxiv:1601.01405*, 2016.

[30] Lambert Ntagwabira and Song Lin Kang. Use of query tokenization to detect and prevent sql injection attacks. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, volume 2, pages 438–440. IEEE, 2010.

[31] Lior Okman, Nurit Gal-Oz, Yaron Gonen, Ehud Gudes, and Jenny Abramov. Security issues in nosql databases. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*, pages 541–547. IEEE, 2011.

[32] Oracle MySQL: top 10 reasons to choose mysql for web-based applications @ONLINE. Available: `http://www.oracle.com/us/products/mysql/mysql-wp-top10-webbased-apps-461054.pdf/`, 2011. [Accessed: 2018-7-2].

[33] Aafaf Ouaddah, Anas Abou Elkalam, and Abdellah Ait Ouahman. Fairaccess: a new blockchain-based access control framework for the internet of things. *Security and Communication Networks*, 9(18):5943–5964, 2016.

[34] Ahmed Oussous, Fatima-Zahra Benjelloun, Ayoub Ait Lahcen, and Samir Belfkih. Comparison and classification of nosql databases for big data. In *Proceedings of International Conference on Big Data, Cloud and Applications*, 2015.

[35] Owasp. testing for nosql injection @ONLINE. Available: `https://www.owasp.org/index.php/Testing_for_NoSQL_injection/`, 2014. [Accessed: 10-12-2017].

[36] Php web site@ONLINE. Available: `http://php.net/`. [Accessed: 10-12-2017].

[37] Aviv Ron, Alexandra Shulman-Peleg, and Emanuel Bronshtein. No sql, no injection? examining nosql security. *Arxiv preprint arxiv:1506.04082*, 2015.

[38] Aviv Ron, Alexandra Shulman-Peleg, and Anton Puzanov. Analysis and mitigation of nosql injections. *IEEE Security & Privacy*, 14(2):30–39, 2016.

[39] Sangita Roy, Avinash Kumar Singh, and Ashok Singh Sairam. Detecting and defeating sql injection attacks. *International Journal of Information and Electronics Engineering*, 1(1):38, 2011.

[40] Ebrahim Sahafizadeh and Mohammad Ali Nematbakhsh. A survey on security issues in big data and nosql. *Advances in Computer Science: an International Journal*, 4(4):68–72, 2015.

[41] Scala programming language web site@ONLINE. Available: `https://www.scala-lang.org/`, 2002. [Accessed: 10-12-2017].

[42] Hossain Shahriar and Mohammad Zulkernine. Information-theoretic detection of sql injection attacks. In *High-Assurance Systems Engineering (HASE), 2012 IEEE 14th International Symposium on*, pages 40–47. IEEE, 2012.

[43] Lwin Khin Shar and Hee Beng Kuan Tan. Defeating sql injection. *IEEE Computer Society*, 46(3):69–77, 2013.

[44] Blockonomi: what is solidity?guide to the language of ethereum smart contracts. Available: `https://blockonomi.com/solidity-guide`. [Accessed: 2018-6-28].

[45] Subhranil Som, Sapna Sinha, and Ritu Kataria. Study on sql injection attacks: Mode detection and prevention. *International Journal of Engineering Applied Sciences and Technology*, 1(8):23–29, 2016.

[46] Stack Overflow: stack overflow annual developer survey @ONLINE. Available: `https://insights.stackoverflow.com/survey/2018#technology/`, 2018. [Accessed: 2018-7-2].

[47] Stu Steiner, Daniel Conte de Leon, and Jim Alves-Foss. A structured analysis of sql injection runtime mitigation techniques. *Hawaii International Conference on System Sciences 2017 (HICSS-50)*, 2017.

[48] Qais Temeiza, Mohammad Temeiza, and Jamil Itmazi. A novel method for preventing sql injection using sha-1 algorithm and syntax-awareness. In *Information and Communication Technologies for Education and Training and International Conference on Computing in Arabic (ICCA-TICET), 2017 Joint International Conference on*, pages 1–4. IEEE, 2017.

[49] Wei Tian, Jing Xu, Kun-Mei Lian, Ying Zhang, and Ju-feng Yang. Research on mock attack testing for sql injection vulnerability in multi-defense level web applications. In *Information Science and Engineering (ICISE), 2010 2nd International Conference on*, pages 1–5. IEEE, 2010.

[50] Haixun Wang, Jian Yin, Chang-shing Perng, and Philip S Yu. Dual encryption for query integrity assurance. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 863–872. ACM, 2008.

[51] Ethereum/web3.js: ethereum javascript. Available: `https://github.com/ethereum/web3.js`. [Accessed: 2018-6-22].

[52] Dave Wichers. Owasp top-10 2013. *OWASP Foundation,The Open Web Application Security Project*, February 2013.

[53] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 151:1–32, 2014.

[54] Alex Zhu and Wei Qi Yan. Exploring defense of sql injection attack in penetration testing. *International Journal of Digital Crime and Forensics (IJDCF)*, 9(4):62–71, 2017.

[55] Guy Zyskind, Oz Nathan, et al. Decentralizing privacy: Using blockchain to protect personal data. In *Security and Privacy Workshops (SPW), 2015 IEEE*, pages 180–184. IEEE, 2015.