

Investigation of SQL Injection Attacks in Web Applications: Types, Attack Stages and a Comparison of Existing Countermeasures

Emmanuel Abiodun Oyeneyin

School of Science and Technology, Nottingham Trent University, United Kingdom

Published August 2022

Abstract

In this digital age, the web application is one of the core aspects of technology that is transforming the world into one village. Many commercial companies rely heavily on web applications for their daily business transactions as well as educational initiatives and a variety of other activities. Unfortunately, this platform has attracted hackers because of the nature of data on a database. An attacker could retrieve, update, and even destroy the data on a database using a crafted technique known as SQL injection. SQL injection attacks can affect the confidentiality, integrity and availability of data on a web application. This paper investigates SQL injection threats in web applications. A web application that is vulnerable to SQL injection was developed and used to examine the common types of SQL injection attacks (tautology, piggy-backed query, logically incorrect query, union query, stored procedure, alternate encoding and inference-based), the stages through which an attack passes, and the existing techniques proposed against it. The existing techniques are compared against the types of SQL injection and on the basis of their tool requirements and limitations. The paper also reviews hash and encryption algorithms as countermeasures, and finds the Bcrypt hash algorithm and the Advanced Encryption Standard (AES) to be the more robust options for protecting the data on a database.

Keywords: SQL injection; web application security; SQLIA; cryptography; Bcrypt; Advanced Encryption Standard; database security

1 Introduction

1.1 Background

In recent years, the World Wide Web has grown drastically. Businesses, individuals, and governments have discovered that web applications may provide effective and dependable solutions to communication and commerce difficulties.

Because of the flexibility and availability that web services offer, certain corporations and individuals are dependent on the services provided (Halfond et al., 2008b). Unfortunately, most web applications are exposed to a range of web-based attacks. One of the deadliest types of web-based attacks is Structured Query Language Injection Attacks (SQLIA). SQLIA vulnerabilities are still considered the most discovered vulnerability in web applications according to the Open Web Application Security Project (OWASP). OWASP has classified SQL injection as one of the top ten security threats to web applications for the last ten years (Li & Zhang, 2019). Because web applications interact with databases that store data like client names, email addresses, credit card numbers, passwords, and other sensitive data, this type of attack poses a significant security risk to companies.

Researchers have presented several approaches to dealing with SQLIAs, with the conclusion that cryptography has been demonstrated to be more secure than other methods in dealing with SQL injection attacks. Cryptography is the art of modifying messages to make them encrypted and resistant to attacks (a term having Greek roots that means “secret writing”) (Singh & Supriya, 2013).

1.2 Statement of the problem

Some system administrators and system development teams are unaware of injection attacks. Hence, one aspect of this study is to increase awareness about SQL injection attacks by investigating common SQL injection attacks on a developed web application that is vulnerable to SQLIA.

1.3 Motivation

The main motivation of this study is to address the most dangerous web application vulnerability, Structured Query Language Injection, by identifying the different tactics used in Structured Query Language Injection attacks.

A successful Structured Query Language Injection attack on a web application can compromise the main security principles of confidentiality, authentication, authorization, and integrity because a website has an underlying database that stores sensitive information such as home addresses, credit card details, and even government secrets. Government databases have always been the target of hackers. Their goal is not to obtain personal financial gain, but to release unauthorized information thereby mocking security services and causing national threat, just like the case of the Lulz-sec group attacks (Padmanabhan, 2012). Researchers have proposed a variety of ways of dealing with Structured Query Language Injection attacks but many of these have setbacks resulting in SQLIAs persisting. These concerns inspired the author to examine the attacks and compare the approaches proposed against them, to help web developers and researchers combat it.

1.4 Aim and objectives

The aim of the project from which this paper is derived is to investigate SQL injection threats in web applications and to implement hybrid encryption of Bcrypt hashing and AES encryption for password and other sensitive data in a database. This paper reports the investigation component, which covers the following objectives:

- **Objective 1:** Create a web application in the PHP programming language that is exposed to SQL injection attack.
- **Objective 2:** Identify the common types of SQL injection.

Table 1 describes each objective and the key actions that are expected to result in each objective.

Table 1. Detailed task for each objective.

Objective ID	Task
Objective 1	To create a login table and employee information table. Insert values into the tables and provide column names. To create a login form (registration.php and login.php), as well as an employee information web form, and link these to the login and employee tables.

Objective ID	Task
Objective 2	To review published research articles on the many types of SQL injections and how to prevent them with a focus on encrypting database values. To commence the thesis write-up based on research and to revise the thesis based on hashing and encryption. To evaluate why Bcrypt hash and Advanced Encryption Standard algorithms are preferable.

1.5 Research questions

The following research questions are addressed by the literature review reported in this paper:

1. According to the literature review, why is the Bcrypt hash algorithm more efficient than the traditional hash algorithm for safeguarding a web application's login?
2. Based on the literature review, why is the Advanced Encryption Standard (AES) 256 algorithm more appropriate for securing the sensitive data in our databases?

1.6 Limitation of the study

Due to the sensitive nature of this study, the different types of SQL injection attacks were obtained from the OWASP website and were carried out carefully on a developed web application. Not all types of SQL injection attacks were available on OWASP, as a result not all types of injection attacks can be demonstrated on the developed web application.

1.7 Structure of the paper

The next section describes the web application that was developed to examine the attacks. The types of SQL injection attacks and the stages of an attack are then set out, followed by a review and comparison of the existing techniques against SQL injection attacks. Hash and encryption algorithms are then reviewed as countermeasures, the ethical and legal issues of the work are considered, and the final section concludes the paper.

2 The vulnerable web application used for the investigation

We built a vulnerable web application (an Employee management system) which was linked to a database to imitate a real SQL injection attack scenario. The system is made up of an Apache web server, a PHP-based web application, and a MySQL database. A database containing a human resource user table and an employee table was established using PhpMyAdmin. A login form (registration.php and login.php) and an employee information web form were created and linked to the login and employee tables.

The developed web application was hosted on a Linux machine (Kali Linux 5.7.0.1) in a virtual machine and supported by the XAMPP server (PHP 8.0.8, Apache 2.4.48, back-end DBMS: MySQL >= 5.0.12 (MariaDB fork)). The server-side code was written in Hypertext Preprocessor (PHP), and HTML, CSS, and Bootstrap were utilised for the front-end coding. The HTTP requests coming from a client system were captured using OWASP Zed Attack Proxy (ZAP), which was also used to obtain cookies and other information for the purpose of testing SQL injection attacks.

3 Types of SQL injection attacks

The types of SQLIAs listed below were drawn from the literature and examined consecutively on our custom web application.

3.1 Tautology

From an attacker's perspective, tautology attacks are regarded as the easiest and most used SQL injection attacks. Halfond et al. (2008a) define a tautology attack as an attacker using a susceptible input field that contains a query's WHERE conditional to perform this type of injection. The tautology's conditional logic is assessed to true as the database examines row by row. Because the condition is a tautology, all the table's entries are matched and returned by the database rather than just one, as it would normally do if injection were not present. A tautology-based attack's overall purpose is injecting malicious code into one or a compound conditional statement, bypassing the authentication and gaining full access into the database.

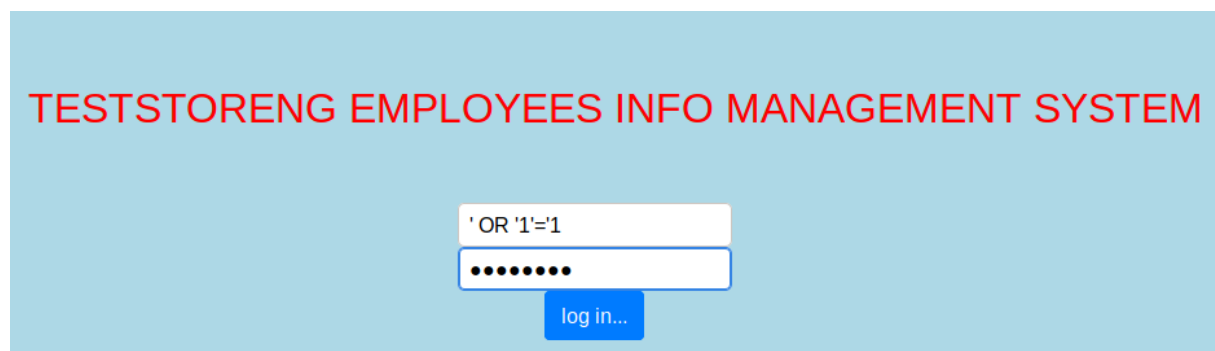


Figure 1. Example of a tautology SQL injection.

In Figure 1, the user submits "or '1=1'---" as login credentials into the login input field. The statement ignores other fields. "or '1=1'---" is injected into the condition, transforming the WHERE clause into a tautology logic causing the web application to grant the user authentication.

3.2 Piggy-backed query

Piggy-backed query attacks are distinct from the other types of SQL injection attacks. Instead of modifying the original SQL query, the attacker inserts new queries into it, leading the database to receive extra queries. The first portion of the query runs normally, while the second half, which contains the injected queries, runs in parallel with the first. According to Halfond et al. (2008b), piggy-backed queries are a particularly dangerous sort of SQL injection attack since the attacker can include any type of SQL command, including a stored procedure, in the original query. A piggy-backed attack's general goal is to extract data, add or modify data or cause a denial of service.

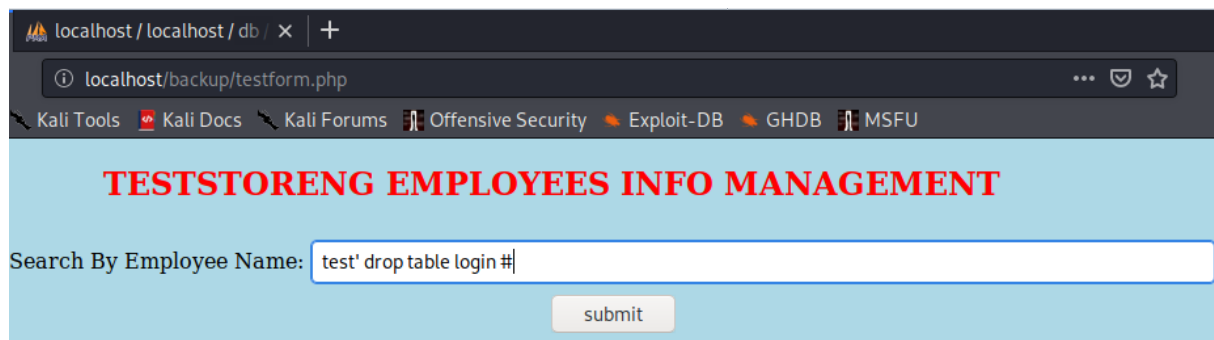


Figure 2. Example of a piggy-backed query SQL injection.

In the piggy-backed query attack example shown in Figure 2, the query string is split into two queries by the query delimiter (;), thereby prompting the database to drop the login table.

3.3 Logically incorrect query

According to Halfond et al. (2008b), an attacker identifies injectable parameters on a vulnerable web application by injecting a malicious statement into the web application, causing a logical error from the database. Normally, error messages are designed for developers to test or debug their application. However, attackers can use them to learn about the kind and structure of a web application's back-end database. The goal of a logically incorrect query attack is to reveal the names of tables and columns that originate from the error message as well as the names of the tables and data types of certain columns.

3.4 Union query

Union query SQL injections are a more advanced type of SQL injection attack that an attacker can use to extract specific data from a database. In a union query, an attacker takes advantage of the SQL UNION operator in fetching data from a specific table. The attacker merges a malicious statement into the original statement by concatenating two or more select queries using the UNION keyword. Because attackers have complete control over the second (injected) query, this strategy can fool the application into providing data from a table other than the one intended by the developer. In Figure 3, an attacker manages to retrieve the table name, database name and column name by appending 'a' and 'b' into the third and fourth column.

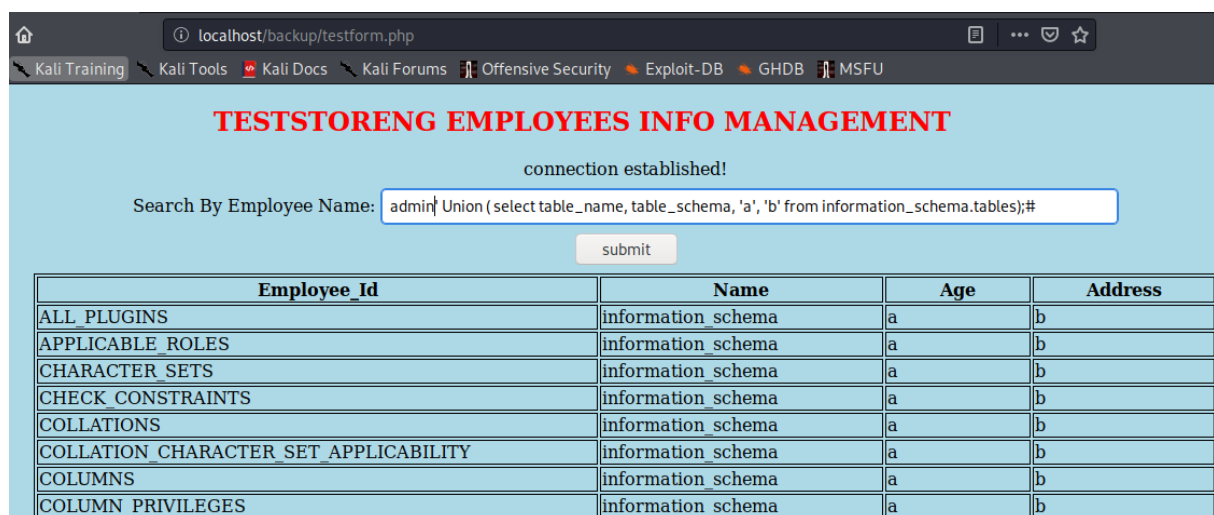


Figure 3. Example of a union SQL injection.

3.5 Stored procedure

A stored procedure is a prepared SQL statement that is saved in the database data dictionary so that it can be reused. It is a usual misperception that stored SQL procedures are secured against SQL injections. Developers are often astonished to find out that stored procedures are vulnerable to SQL injection attacks also. Halfond et al. (2008b) stated that SQL injection attacks are designed to run stored procedures offered by a particular database, including operations that interact with the operating system, once the attacker has figured out which database is being used. The main goals of a stored procedure attack are performing denial of service attacks and privilege escalation. The stored procedure provides a true or false response indicating whether or not the user's credentials were successfully authenticated. An attacker can alter the query string in a login form by simply injecting “;SHUTDOWN --” into the username or password field to launch a stored procedure SQL injection attack.

3.6 Alternate encoding

Special characters are required in many SQL injection attacks (for example, single quotes, dashes and comment operators). Because of the defensive coding practice of scanning for these bad characters, an attacker uses hexadecimal, ASCII and Unicode character encoding to get around this defence (Halfond et al., 2008b). According to Halfond et al. (2008b), implementing a successful code-based defence against alternate encodings is tough because it necessitates a thorough examination of all conceivable encodings that can influence query strings as they pass through various application layers. In Figure 4, the char() function and ASCII hexadecimal encoding are used. The char() function takes an integer or hexadecimal encoding of a character as a parameter and returns a character instance. The ASCII hexadecimal encoding of the stream of integers in the second section of the injection, when executed, becomes SHUTDOWN.

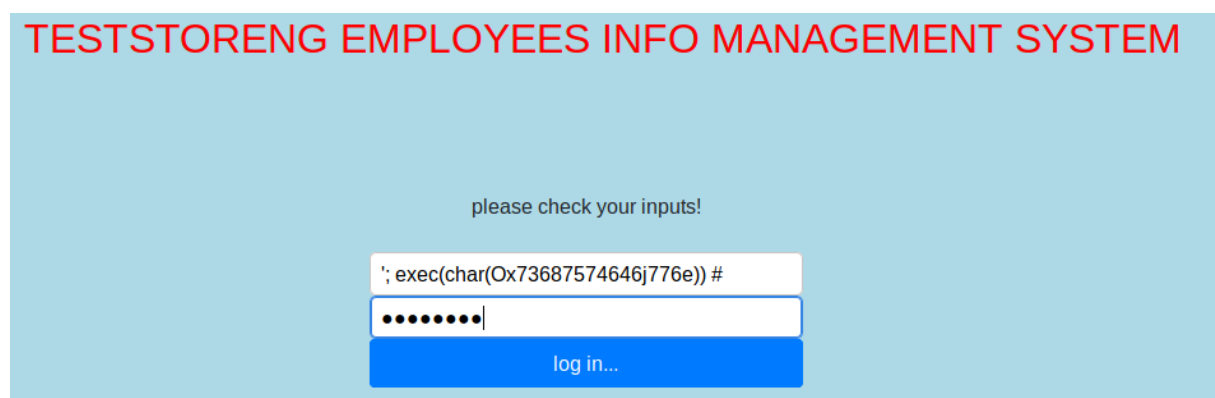


Figure 4. Example of an alternate-encoding SQL injection.

3.7 Inference-based

In this type of attack, the attacker aims at extracting data, determining the database schema and detecting vulnerable parameters.

Halfond et al. (2008a) stated that in an attempt to breach a website that is sufficiently safeguarded, if an injection succeeds, feedback will not be available in the form of database error messages. Rather, the attacker obtains responses from the database using a different method by injecting commands into the website and observing changes. In situations where there are no responses to the query to the attacker, it is feasible to see the query's side effects and derive results. According to Halfond et al.

(2008b), there is a particular type of inference-based SQL injection known as a 'timing attack' which the attacker uses to gather information about the content of the database. A timing attack is triggered when attackers construct the injected query as an if/then statement with the branch corresponding to questions about the database's contents. The attacker then involves WAITFOR/SLEEP keywords with a branch causing a database delay in its response by a given amount of time. By calculating the increase and decrease in database response time, the attacker can figure out which branch was chosen and answer the injected query.

The attacker was able to instruct the web application to sleep for 10 seconds as seen in Figure 5. By extending the period, such an injection might be exploited as a denial of service.

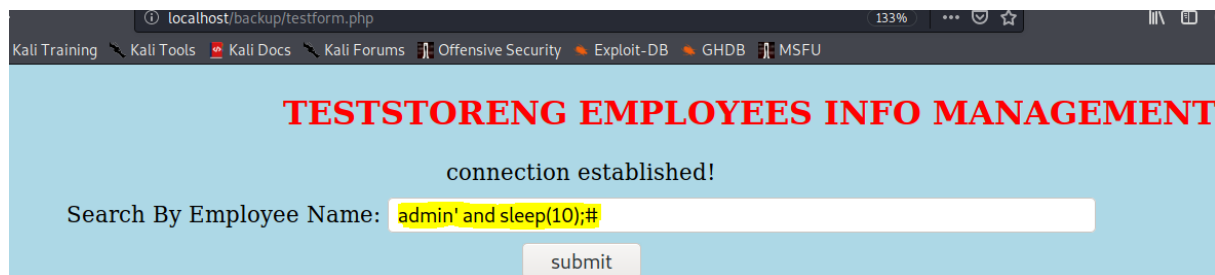


Figure 5. Example of an inference-based SQL injection.

4 Stages of a SQL injection attack

When launching a SQL injection attack, the attack passes through different stages. We will discuss the stages of a typical web application before going into the different stages of SQL injection attacks. In a typical web application, a web server receives a user's request, starts the appropriate script, and provides the parameters for it to process. The script handles all the processing and dynamically generates HTML output. The result is sent to the web server, which subsequently passes it on to the client. For server-side programming, various programming languages are available, as well as various databases for data storage. Other than inserting a forged SQL query, an attacker goes through the same stages. According to Kao et al. (2019), the authors describe in five stages how an attacker inputs malicious code in-between an unintended SQL statement using a web form:

1. The attacker sends malicious HTTP input to a web server.
2. Malicious SQL code is constructed on a web server and sent to the database.
3. Responses from the database are sent to the web server.
4. The web application confirms the attacker.
5. The attacker gains access into the database.

In Figure 6, the arrows circulating depict the web flow during an SQL injection attack.

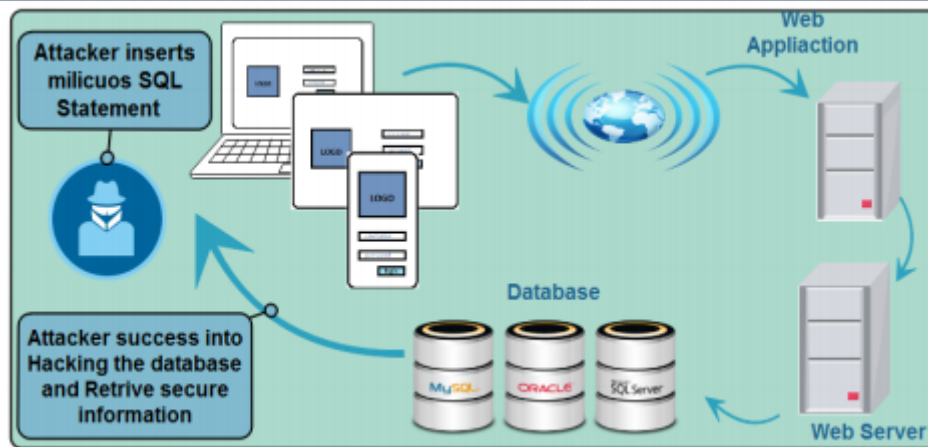


Figure 6. SQL injection attack procedure (Elshazly et al., 2014).

5 Existing techniques against SQL injection attacks

Durai et al. (2021) stated that input strings are identified by positive tainting (trusted strings) and negative tainting (untrusted strings) which are formed by initialization and are categorized. Instead of untrusted data, syntax-aware evaluation is based on the identification, marking, and tracking of trusted data. However, this causes incompleteness in identifying the data to be marked as relevant. Incompleteness has extremely different repercussions in negative and positive tainting, which is one of the primary issues when designing a security solution based on dynamic tainting.

The AMNESIA technique (Analysis and Monitoring for Neutralizing SQL Injection Attacks) is a combination of static and dynamic parts for detecting SQL injection during runtime. According to Dehariya et al. (2016), the AMNESIA technique was proposed to trace SQL input flow and start attack input using SQLInjectionGen for discovering hotspots and JCrasher for producing test cases. During the experiment for this technique, SQLInjectionGen generated two false negatives from an attempt using two MySQL databases. AMNESIA produces precise findings, but it necessitates several stages and the use of several tools.

Aziz et al. (2016) presented an article identifying SQL injection vulnerabilities using a search-based software technique. In the article, datasets were generated by a genetic programming approach for testing applications against SQL injection. The demonstration was carried out using open-source kits, an evolutionary computation system (ECS) for genetic programming, MySQL as a database, and SuiteCRM for testing. The testing of the approach relied on a general SQL parser for identifying malicious users between the target system and databases. During the demonstration, test cases resulted in syntax errors, and several generated inputs resulted in failure to initiate the attack.

Bhattacharyya et al. (2017) describe the Reverse Insertion Algorithm (RIA) as a method used to defend against login credential retrieval. The authors further stated that the proposed solution terminates user input injection at the login page by filtering SQL commands and storing data in an encoded pattern. The authors mentioned that user input data are first reversed and split into groups containing two characters. Special characters are inserted in-between the groups before saving in the database.

Philippaerts et al. (2011) proposed Code Pointer Masking, an effective technique against SQL injection using a C compiler in a Linux environment. The article mentions that CPM prevents SQL injection by modifying code pointers, bringing changes to the control flow. Code Pointer Masking provides

masking for return addresses, the global offset table, and function pointers. Masking depends on the logic operator to avoid SQL injection in the memory of the system; however, masks will not be calculated without the support of additional input.

In a paper presented by [Fu et al. \(2007\)](#), the authors proposed SAFELI, a tool for detecting SQL injection during compiling. The paper states that the SAFELI tool contains a white-box static framework and a hybrid constraint solver. SAFELI happens to be a better tool than a black-box scanner because of its ability to detect vulnerabilities, but it works only on ASP.NET web applications.

[Shahriar and Zulkernine \(2012\)](#) describe Candidate evaluation for Discovering Intent Dynamically (CANDID), from the developer's perspective, as a tool that relies on benign input for dynamically generating the query structure of the programmer-intended query, comparing the user query structure with the benign query structure, and automatically rejecting SQL injection from the web application. As a result, it eliminates the need to manually modify the application in order to generate prepared statements.

[Dubey and Gupta \(2016\)](#) proposed an integrated model of proxy filtering and the SQLIPA technique by combining two existing techniques. In the first stage, query strings from the client web server are checked for string legitimacy, relying on the proxy server for filtering of a query string. In the second stage, each newly registered username and password is converted into a hash value and added separately to the user table. During login, the entered username and password values are compared with the values stored in a database. However, this was merely a model that was proposed and never executed or demonstrated.

[Ali et al. \(2009\)](#) proposed a defence method that uses a hash algorithm to validate the security of usernames and passwords. [Ali et al. \(2009\)](#) created their strategy by transforming both the user login and password data into hash values and storing them in separate database columns. The server decrypts the hash value once the client computer sends the hash value of both the username and password through an HTTP request. [Bansal and Park \(2015\)](#) recorded a 0% threat of SQL injection attack on their multi-hashing technique from 20 different users who attempted various sorts of SQL injection attacks. During registration, [Bansal and Park \(2015\)](#) used the SHA-512 hashing algorithm to hash the login and password, known as first-level hashing. The login query is encrypted using the same hash technique and kept in separate databases resulting in an identical hash algorithm, known as second-level hashing.

Data Encryption Standard (DES), Rivest Cipher 4 (RC4), Advanced Encryption Standard (AES), Diffie-Hellman (DH), Elliptic Curve Cryptography (ECC), and Rivest-Shamir-Adleman (RSA) are types of encryption algorithms that [Musliyana et al. \(2015\)](#) consider to be more efficient than hashing algorithms. [Nagpal et al. \(2017\)](#) mentioned a two-level encryption technique using AES and RSA algorithms to avoid SQLIA. Throughout this process each client is issued a separate secret key, while the server applies a mixture of private key and public key. In this implementation of two-level encryption, a symmetric algorithm was applied during registration while the login query uses an asymmetric algorithm. Another technique against SQLIA was proposed by Selvamani and Kannan (2011) using policy-based access control and the DES encryption method, which can be described as an algorithm that repeats its procedure 16 times before producing the final cipher. Data from the client is encrypted in DES as soon as it reaches the server. Policy-based access control was applied to determine encryption and decryption. [Agrawal \(2015\)](#) suggested an approach to prevent SQLIA by converting SQL queries into tokens and encrypting them with the AES algorithm. Tokenization was applied to recognise single and double quotes, which are subsequently saved on the client side. Data

and column names are encrypted and delivered to the server, which decrypts and turns them to tokens before storing them in a dynamic table. A hybrid encryption technique was proposed by Gupta et al. (2019) for encrypting the data on a database using a combination of two encryption schemes: Advanced Encryption Standard (AES) and Elliptic Curve Cryptography. AES was used in encrypting the password in the user login table while ECC was used for encrypting the sensitive data stored in the database. Gupta et al. (2019) described ECC as a form of public-key cryptography which uses both a public key and a private key based on the algebraic structure of the elliptic curve for creating cryptographic keys, unlike another form of public key which generates keys through prime number factorization. ECC was deployed on a database, encrypting the actual data in the database into non-readable text (encrypted data).

5.1 Comparison of existing techniques against different types of SQL injection

Table 2 gives a summary of different proposed techniques addressing each type of SQL injection attack, which were collected from different articles (Muttuqin et al., 2020; Varshney & Ujjwal, 2019; Tajpour & Shooshtari, 2010). None of the comparisons listed were empirically conducted. In the table, ✓ identifies a technique which can terminate a type of SQLi, X identifies a technique which is unable to terminate a type of SQLi, and a blank cell indicates that no mark was recorded.

Table 2. Comparison of existing techniques against different SQL injection attacks.

SQLi techniques / Types of SQLi	Tauto-logy	Logically In-correct Query	Union Query	Piggy-Backed Query	Stored Pro-cedure	Infer-ence	Alternate encoding
Amnesia	✓	✓	✓	✓		✓	✓
Reverse Insertion Algorithm	✓	✓	✓				
Code Pointer Masking	X	X	X	X	X	X	X
Safeli		✓	✓	✓	✓	✓	✓
Candid	✓						
Positive tainting and syntax-aware evaluation	✓	✓	✓	✓	✓	✓	✓
Multi hashing algorithm (Bansal & Park, 2015)	✓	✓	✓	✓	✓	✓	✓
Blowfish	✓	✓	✓	✓		✓	
Message Digest (MD5)	✓	✓	✓	✓	✓	✓	
Advanced Encryption Standard (AES)	✓	✓	✓	✓	✓	✓	✓
AES + MD5	✓	✓	✓	✓	✓	✓	✓
Tokenization technique with AES	✓	✓	✓	✓	✓	✓	✓

Note: ✓ = attack type addressed by the technique; X = not addressed; blank = not reported.

5.2 Comparison of existing techniques based on tool requirements and limitations

Table 3 compares several articles concentrating on the infrastructural tools required by each existing technique as well as its limitations. Each of the existing techniques is classified as a detection or a prevention type.

Table 3. Comparison of existing techniques based on tools requirements and limitations.

SQLi technique	Detection time	Location	Type	Limitations
Amnesia	Runtime	Server	Prevention	Amnesia can be less accurate if the code is obfuscated. Performance is dependent on the reliability of the static analysis technique (Halfond et al., 2008b).
Reverse Insertion Algorithm	Runtime	Server	Prevention	
Code Pointer Masking	Runtime	Server	Detection	CPM is dependent on input; without the support of additional input, masks will not be calculated (Philippaerts et al., 2011).
Safeli	Compile / Runtime	Server / Client	Detection	Safeli is only compatible with ASP.NET web application vulnerabilities (Durai et al., 2021).
Candid	Runtime	Server	Prevention	Cannot handle external function declarations. Fails to execute code containing loops (Chen et al., 2021).
Positive tainting and syntax-aware evaluation	Runtime	Server	Detection / Prevention	Because of the persistent storing of trusted strings, there may be a likelihood of second-order attack (Durai et al., 2021).
Multi hash algorithm	Runtime	Server	Prevention	Because testing of this implementation was done by 20 persons with 10 trials, we cannot assume that all types of SQL injection attacks were attempted (Bansal & Park, 2015).
Tokenization with AES	Runtime	Server	Prevention	

6 Cryptography as a countermeasure against SQL injection

6.1 Hash algorithms

A hash function is defined by Jiang et al. (2014) as a component that takes any amount of data and produces a finite-length message digest, making it difficult to recover the input from the hash value because the hash function runs a one-way transformation phase. Hash functions are frequently used in conjunction with digital signatures to ensure data integrity. If the input is altered even slightly, a hash function will generate a different hash value. Marton et al. (2010) mentioned that the signature would indicate if the hash value was being manipulated, as well as the hash will indicate if the input was being updated with digital signatures. It is difficult but not impossible to find a different message that generates the same hash. This can be due to collision effects. According to Bidhuri et al. (n.d.), a pre-calculated hash value could be used to decipher a hashed password. MD5, SHA1, and SHA256 are common ways of hashing passwords. These traditional hash algorithms, which allow for speedy calculation and digestion of large amounts of data, are vulnerable to collision threats using

differential cryptanalysis (Savage, 2008). Therefore, dictionary attacks, brute force attacks, rainbow attacks, lookup table attacks, and a variety of other software tools can all be used against the MD5 and SHA1 hash algorithms.

One way to secure passwords with a traditional hash algorithm from these attacks is by applying salt to the algorithm. Marton et al. (2010) define salt as a secure arbitrary string of bits derived from pseudo-randomness in which the generated value is applied to the plain message prior to the hashing process. When the hash algorithm is used multiple times, the salt is calculated separately each time to ensure that each hash value is unique. Bidhuri et al. (n.d.) highlight the strength of salt processing as being resilient to rainbow table and lookup table attacks but susceptible to dictionary and brute force attacks. This weakness is because of optimization of speed. An attacker can reverse the hash function using modern GPUs by calculating every potential password. Thus, the concept of the key derivation function (KDF) was introduced to be purposely slow in order to prevent brute-force or dictionary attacks on passwords. According to Turan et al. (2010), the key generation process in the key derivation function approach contains a computation that stretches the time it takes to generate a hash value, slowing down the operation; as a result, an attacker would spend more time attempting a brute force attack.

The Bcrypt algorithm is a type of key derivation function that is widely recognized as one of the most successful and popular password-protection methods available. Malvoni and Knezovic (2014) stated that Bcrypt generates hashes from passwords by running them through the Blowfish cipher numerous times, which improves password encryption to prevent attacks; hence Bcrypt is developed from the EksBlowfish technique. As shown in Figure 8 (Algorithm 2), there are two stages of executing the Bcrypt algorithm.

The first stage of Bcrypt involves invoking the function EksBlowfishSetup (Algorithm 1 in Figure 7) along with three parameters: the cost, salt, and encryption key. According to Malvoni and Knezovic (2014), the cost determines how expensive the key setup configuration is, the salt (a 128-bit random value) ensures that hash values of the same password do not always match, and the encryption key is the password that is chosen.

Algorithm 1 EksBlowfishSetup(cost, salt, key) [13]

```

1: state  $\leftarrow$  InitState()
2: state  $\leftarrow$  ExpandKey(state, salt, key)
3: repeat ( $2^{\text{cost}}$ )
4:   state  $\leftarrow$  ExpandKey(state, 0, salt)
5:   state  $\leftarrow$  ExpandKey(state, 0, key)
6: return state

```

Figure 7. EksBlowfish set-up (Malvoni & Knezovic, 2014).

In the second stage, EksBlowfish in Electronic Codebook (ECB) mode is applied to cipher the 192-bit value OrpheanBeholderScryDoubt 64 times.

Algorithm 2 bcrypt(cost, salt, pwd) [13]

```

1:  $state \leftarrow EksBlowfishSetup(cost, salt, key)$ 
2:  $c_{text} \leftarrow \text{"OrpheanBeholderScryDoubt"}$ 
3:  $repeat(64)$ 
4:    $c_{text} \leftarrow EncryptECB(state, c_{text})$ 
5:  $return Concatenate(cost, salt, c_{text})$ 

```

Figure 8. Bcrypt set-up (Malvoni & Knezovic, 2014).

Many researchers have proven the Bcrypt algorithm to be a robust algorithm. In the Bcrypt security experiment on passwords based on character type carried out by [Batubara et al. \(2021\)](#), the result stated that the Bcrypt algorithm was successful in fending off brute force attacks on passwords stored in alphabetic characters.

6.2 Encryption algorithms

We can conclude from the literature that the Advanced Encryption Standard is more widely used because it occurs in quite a number of related works. As a result, this section focuses on the Advanced Encryption Standard, explaining why it is more efficient than other encryption methods and providing a description of it.

According to [Musliyana et al. \(2015\)](#), encryption and the hash function are two linked fields whose methodologies cannot be substituted for one another. Encryption is a two-way algorithm, unlike the hash function, which operates in one direction only. In encryption, a sender transmits data to the recipient, and the data is only accessible to the recipient on the condition that the sender has given them a key to decode it. The encryption process converts plaintext into ciphertext, which is dependent on the key, by making various substitutions and alterations ([Xuelong, 2011](#)). According to [Idrus et al. \(2008\)](#), encryption is made up of a variety of algorithms that have been created by various developers. The various existing types of encryption algorithms are the Data Encryption Standard (DES), Rivest Cipher 4 (RC4), the Advanced Encryption Standard (AES), Diffie-Hellman (DH), Elliptic Curve Cryptography (ECC) and Rivest-Shamir-Adleman (RSA).

[Hardy et al. \(2020\)](#) mentioned that the Rijndael algorithm was chosen as the Advanced Encryption Standard (AES) by the National Institute of Standards and Technology (NIST) due to its great reliability and performance. In 2001 NIST selected AES as the succeeding encryption standard to supersede DES, because as computational resources developed, attacks against the earlier cipher became more plausible ([Hardy et al., 2020](#)). The US government confirmed in June 2003 that AES can be used to encrypt private information after the National Security Agency evaluated AES contenders ([Hardy et al., 2020](#)). The architecture of AES, as well as the strength of all key lengths, can protect sensitive data up to the Secret level.

AES uses a substitution-permutation design approach with a constant block size of 128 bits and a key size of 128, 192, or 256 bits. The state of AES is a four-by-four column-order byte matrix ([Hardy et al., 2020](#)), unlike most ciphers that have a long line of bytes. Figure 9 shows how a message of 16 bytes (the equivalent of 128 bits) is represented.

$$\begin{bmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix}$$

Figure 9. Representation of the 4×4 AES grid of 16 bytes (Hardy et al., 2020).

According to [Heron \(2009\)](#), the key size of the AES cipher determines the number of repetitions required to turn the message into the final ciphertext. For 128-bit keys, 10 rounds are required, 192-bit keys require 12 rounds, and 256-bit keys require 14 rounds. During AES procedures everything is organized in a four-by-four column order as seen in Figure 9. The rounds are divided into three stages: the initial round, intermediate rounds, and the final round. The initial round is the initialization, which involves the plaintext and an XOR operation, while each intermediate round involves four steps:

1. Substitute Bytes transformation
2. Shift Rows transformation
3. Mix Columns transformation, and
4. Add Round Key transformation

[Heron \(2009\)](#) mentioned that steps 1–4 are repeated up to the number of rounds defined by the key size, leaving one round for the final round. At the final round only the Substitute Bytes, Shift Rows and Add Round Key transformations are used, leaving out the Mix Columns transformation because the permutation level does not affect the final round.

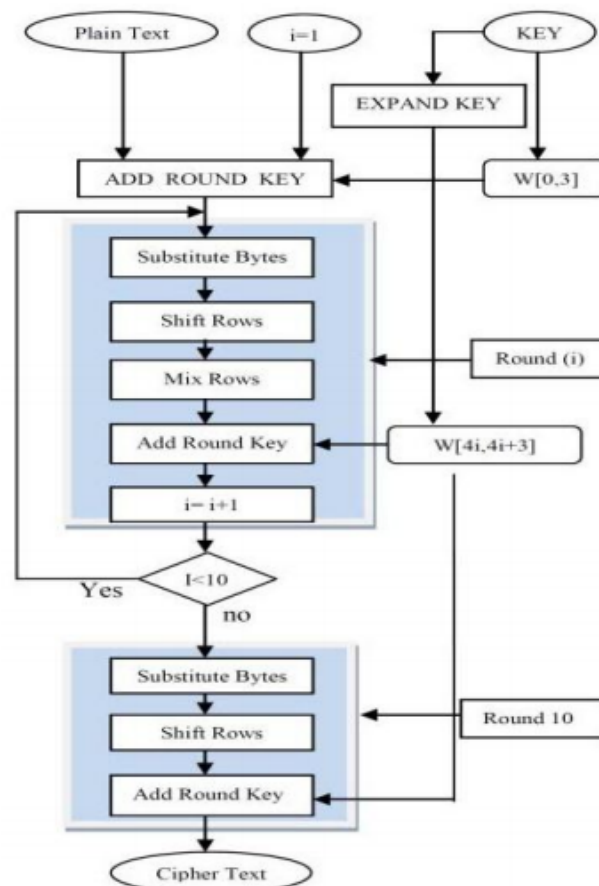


Figure 10. Example of an AES block diagram (Singh & Supriya, 2013).

Figure 10 shows the AES process during 10 rounds of iteration, which explains the level of secrecy provided by the Advanced Encryption Standard. Many researchers have found AES to be superior to alternative encryption schemes. Ganesh Kadam and Khairnar (2015) claim that no successful brute-force attack on AES has been discovered so far. The demonstration by Padmavathi and Kumari (2013), comparing AES, DES, and RSA using the LSB substitution technique (a steganographic algorithm) built in C#, found that AES was effective and used less buffer compared to RSA and DES. Álvarez-Sánchez et al. (2018) stated that the AES block cipher has the feature of being hardware accelerated on current computing platforms, hence AES could safeguard data from specialized hardware or GPUs. AES also has the advantage of supporting higher key sizes than 3DES and DES; because of the huge key size, AES is less vulnerable to attack (Ganesh Kadam & Khairnar, 2015).

7 Ethical and legal considerations

SQL injections are not only regarded as illegal but also unethical. During the evaluation of the various SQL injection attacks, this work recognised the Menlo principles published to add consistency to the ethical monitoring of cybersecurity research work (Dittrich et al., 2013). Hence, the four core ethical principles of respect for persons, beneficence, justice, and respect for law were observed. No personal data or vulnerability was leaked. All test cases were hypothetically created and carried out on a developed web application over a private network.

SQL injections are prohibited in the United Kingdom under various laws and regulations stemming from the Data Protection Act 1998, now the Data Protection Act 2018, which guards against any

unauthorized access and unauthorized modification of personal data. All test cases for this work were therefore conducted on a purpose-built web application and on the Open Web Application Security Project (OWASP) platform, which is known as an organizational standard for testing SQL injection attacks. Other branches of law such as intellectual property and computer misuse were also considered.

8 Conclusion

SQL injection attack happens to be an emerging attack according to the OWASP report and, considering the limitations and comparison in the literature, many techniques have been proposed to fully mitigate the entire injection attack. This paper has identified the common types of SQL injection, described the stages through which an attack passes, and compared the existing techniques against the types of attack and on the basis of their tool requirements and limitations.

The Bcrypt hashing algorithm and the Advanced Encryption Standard are the more robust algorithms in the literature reviewed. The Bcrypt hash technique uses salting and stretching to make passwords difficult to crack, and cracking a 256-bit encrypted message requires 2^{256} combinations, which makes it difficult to crack. A model that uses these two algorithms therefore aims for data confidentiality and integrity in addition to preventing SQL injection. The design, implementation and evaluation of such a model is reported in a companion paper.

References

- [1] Agrawal J (2015). Mitigation of SQL Injection Attacks in PHP Web Applications. 04(12), 2291–2294. <http://203.190.148.228:8080/xmlui/bitstream/handle/123456789/682/jyoti.pdf?sequence=2&isAllowed=y>
- [2] Ali S, Rauf A, Javed H (2009). SQLIPA: An authentication mechanism against SQL injection. *European Journal of Scientific Research*, 38(4), 604–611.
- [3] Álvarez-Sánchez R, Andrade-Bazurto A, Santos-González I, Zamora-Gómez A (2018). AES-CTR as a password-hashing function. *Advances in Intelligent Systems and Computing*, 649, 610–617. https://doi.org/10.1007/978-3-319-67180-2_59
- [4] Aziz B, Bader M, Hippolyte C (2016). Search-based SQL injection attacks testing using genetic programming. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9594(March), 183–198. https://doi.org/10.1007/978-3-319-30668-1_12
- [5] Bansal Y, Park JH (2015). Multi-hashing for Protecting Web Applications from SQL Injection Attacks. *International Journal of Computer and Communication Engineering*, 4(3), 187–195. <https://doi.org/10.17706/ijcce.2015.4.3.187-195>
- [6] Batubara TP, Efendi S, Nababan EB (2021). Analysis Performance BCRYPT Algorithm to Improve Password Security from Brute Force. *Journal of Physics: Conference Series*. <https://doi.org/10.1088/1742-6596/1811/1/012129>
- [7] Bhattacharyya P, Sastry HG, Marriboyina V, Sharma R (2017). Smart and Innovative Trends in Next Generation Computing Technologies Communications in Computer and Information Science 828. <http://www.springer.com/series/7899>
- [8] Bidhuri V, Heffernan N, Heffernan N (n.d.). Enhancing Password Security Using a Hybrid Approach of SCrypt Hashing and AES Encryption MSc Internship Cyber Security National College of Ireland Supervisor :
- [9] Chen D, Yan Q, Wu C, Zhao J (2021). SQL Injection Attack Detection and Prevention Techniques Using Deep Learning. *Journal of Physics: Conference Series*, 1757(1). <https://doi.org/10.1088/1742-6596/1757/1/012055>
- [10] Dehariya H, Kumar P, Ahirwar M (2016). A Survey on Detection and Prevention Techniques of SQL Injection Attacks. *International Journal of Computer Applications*, 137(5), 9–15. <https://doi.org/10.5120/ijca2016908672>

- [11] Dubey R, Gupta H (2016). SQL filtering: An effective technique to prevent SQL injection attack. *2016 5th International Conference on Reliability, Infocom Technologies and Optimization, ICRITO 2016: Trends and Future Directions*, 312–317. <https://doi.org/10.1109/ICRITO.2016.7784972>
- [12] Durai KN, Subha R, Haldorai A (2021). A Novel Method to Detect and Prevent SQLIA Using Ontology to Cloud Web Security. *Wireless Personal Communications*, 117(4), 2995–3014. <https://doi.org/10.1007/s11277-020-07243-z>
- [13] Elshazly K, Fouad Y, Saleh M, Sewisy A (2014). A Survey of SQL Injection Attack Detection and Prevention. *Journal of Computer and Communications*, 02(08), 1–9. <https://doi.org/10.4236/jcc.2014.28001>
- [14] Fu X, Lu X, Peltsverger B, Chen S, Qian K, Tao L (2007). A static analysis framework for detecting SQL injection vulnerabilities. *Proceedings - International Computer Software and Applications Conference*, 1(Compsac), 87–94. <https://doi.org/10.1109/COMPSAC.2007.43>
- [15] Ganesh Kadam K, Khairnar V (2015). Hybrid Rsa-Aes Encryption for Web Services an Implementation of Hybrid Encryption-Decryption (Rsa With Aes and Sha256) for Use in Data Exchange Between Client Applications and Web Services. *International Journal of Technical Research and Applications*, 31(31), 51–56. www.ijtra.com
- [16] Gupta H, Mondal S, Ray S, Giri B, Majumdar R, Mishra VP (2019). Impact of SQL Injection in Database Security. *Proceedings of 2019 International Conference on Computational Intelligence and Knowledge Economy, ICCIKE 2019*, February 2020, 296–299. <https://doi.org/10.1109/ICCIKE47802.2019.9004430>
- [17] Halfond WGJ, Orso A, Manolios P (2008a). WASP: Protecting web applications using positive tainting and syntax-aware evaluation. *IEEE Transactions on Software Engineering*, 34(1), 65–81. <https://doi.org/10.1109/TSE.2007.70748>
- [18] Halfond WGJ, Viegas J, Orso A (2008b). A Classification of SQL Injection Attacks and Countermeasures. *Preventing Sql Code Injection By Combining Static and Runtime Analysis*, 53.
- [19] Hardy DW, Richman F, Walker CL (2020). Advanced Encryption Standard. *Applied Algebra*, 275–290. <https://doi.org/10.1201/9781439894699-15>
- [20] Heron S (2009). Advanced Encryption Standard (AES). *Network Security*, 2009(12), 8–12. [https://doi.org/10.1016/S1353-4858\(10\)70006-4](https://doi.org/10.1016/S1353-4858(10)70006-4)
- [21] Idrus SZS, Aljunid SA, Asi SM, Sudin S, Ahmad RB (2008). Performance Analysis of Encryption Algorithms Text Length Size on Web Browsers. *IJCSNS International Journal of Computer Science and Network Security*, 8(1), 20–25.
- [22] Jiang GL, Wu J, Liu K, Fu BZ, Chen MY (2014). Implementation and performance analysis of SDN controller. *Dongbei Daxue Xuebao/Journal of Northeastern University*, 35, 313–318.
- [23] Kao DY, Lai CJ, Su CW (2019). A Framework for SQL Injection Investigations: Detection, Investigation, and Forensics. *Proceedings - 2018 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2018*, 2838–2843. <https://doi.org/10.1109/SMC.2018.00483>
- [24] Li Y, Zhang B (2019). Detection of SQL Injection Attacks Based on Improved TFIDF Algorithm. *Journal of Physics: Conference Series*, 1395(1). <https://doi.org/10.1088/1742-6596/1395/1/012013>
- [25] Malvoni K, Knezovic J (2014). Are your passwords safe: Energy-efficient Bcrypt cracking with low-cost parallel hardware. *8th USENIX Workshop on Offensive Technologies, WOOT 2014*, Algorithm 1.
- [26] Marton K, Suciu A, Ignat I (2010). Randomness in digital cryptography: A survey. *Romanian Journal of Information Science and Technology*, 13(3), 219–240.
- [27] Musliyana Z, Arif TY, Munadi R (2015). Security enhancement of Advanced Encryption Standard (AES) using time-based dynamic key generation. *ARNP Journal of Engineering and Applied Sciences*, 10(18), 8347–8350.
- [28] Muttaqin MF, Yaddarabullah, Permana SDH (2020). Implementation of aes-128 and token-base64 to prevent sql injection attacks via http. *International Journal of Advanced Trends in Computer Science and Engineering*, 9(3), 2876–2882. <https://doi.org/10.30534/ijatcse/2020/60932020>
- [29] Nagpal B, Chauhan N, Singh N (2017). A survey on the detection of SQL injection attacks and their countermeasures. *Journal of Information Processing Systems*, 13(4), 689–702. <https://doi.org/10.3745/JIPS.03.0024>
- [30] Padmanabhan S (2012). Hacking for Lulz: Employing Expert Hackers to Combat Cyber Terrorism. *Vand. J. Ent. & Tech. L.*, 15(1), 191–226. <http://www.jetlaw.org/wp-content/pdf/Padmanabhan.pdf>

-
- [31] Padmavathi B, Kumari SR (2013). A Survey on Performance Analysis of DES, AES and RSA Algorithm along with LSB Substitution Technique. *International Journal of Science and Research*, 2(4), 2319–7064. www.ijsr.net
- [32] Philippaerts P, Younan Y, Muylle S, Piessens F, Lachmund S, Walter T (2011). Code pointer masking: Hardening applications against code injection attacks. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6739 LNCS(July), 194–213. https://doi.org/10.1007/978-3-642-22424-9_12
- [33] Savage A (2008). MD5 and SHA-1 Collision Attacks: A Tutorial. 5, 1–4. <https://koclab.cs.ucsb.edu/teaching/crcn/project/2008/savage.pdf>
- [34] Shahriar H, Zulkernine M (2012). Information-theoretic detection of SQL injection attacks. *Proceedings of IEEE International Symposium on High Assurance Systems Engineering*, 40–47. <https://doi.org/10.1109/HASE.2012.31>
- [35] Singh G, Supriya S (2013). A Study of Encryption Algorithms (RSA, DES, 3DES and AES) for Information Security. *International Journal of Computer Applications*, 67(19), 33–38. <https://doi.org/10.5120/11507-7224>
- [36] Tajpour A, Shoostari MJZ (2010). Evaluation of SQL injection detection and prevention techniques. *Proceedings - 2nd International Conference on Computational Intelligence, Communication Systems and Networks, CICSyN 2010*, July, 216–221. <https://doi.org/10.1109/CICSyN.2010.55>
- [37] Turan MS, Barker E, Burr W, Chen L (2010). Recommendation for password-based key derivation: part 1: storage applications. *NIST Special Publication*, December, 800–132. <http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf>
- [38] Varshney K, Ujjwal RL (2019). LsSQLIDP : Literature survey on SQL injection detection and prevention techniques. *Journal of Statistics and Management Systems*, 22(2), 257–269. <https://doi.org/10.1080/09720510.2019.1580904>
- [39] Xuelong (2011). Implementation of an Improved Data Encryption Algorithm in a Web Based Learning System. *Physics International*, 2(1), 31–35. <https://doi.org/10.3844/pisp.2011.31.35>