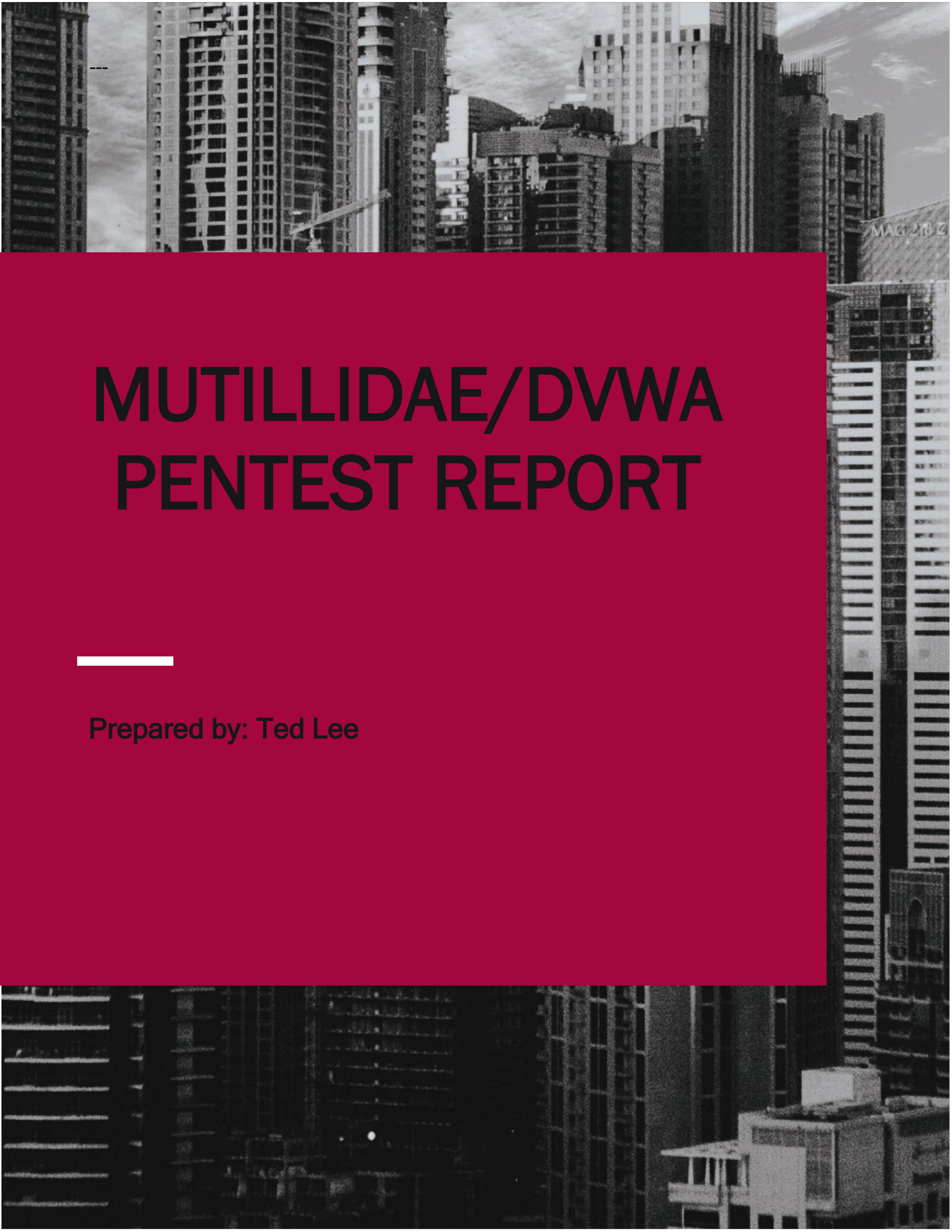



MUTILLIDAE/DVWA PENTEST REPORT

Prepared by: Ted Lee

1 Executive Summary

In this assessment, the team operated on two intentionally vulnerable web applications: Mutillidae and DVWA. Both of these applications are hosted on a controlled Metasploitable 2 (MS2) virtual machine (VM) and was operated on a Kali Linux VM. The purpose of these tests was to explore common SQL injection techniques and understand how improper input validation and weak password protections can lead to complete data compromise.

First, the assessment team exploited an error-based SQL injection against Mutillidae's User Info page which allowed us to obtain the records of 16 users to include their usernames, passwords, and profile signatures. We then exploited a union-based SQL injection against DVWA to reveal the data from both the users and guestbook database. We were then able to obtain the password hashes from the users database and leverage generative AI (ChatGPT) to translate the hashes into plaintext. Finally, we changed the SQL injection statements in order to extract the data from the guestbook database as well.

These activities highlight two critical issues: the absence of proper server-side input sanitization, which enabled error- and union-based injection attacks, and the use of weak password-hashing mechanisms that make credential theft easily recoverable. Such vulnerabilities pose serious real-world implications as attackers can bypass authentication, exfiltrate sensitive data, and compromise user privacy.

To protect against these types of attacks, the assessment team recommends prioritizing input validation/sanitization, leveraging parameterized queries/prepared statements, and enforcing least privilege access. We also recommend implementing modern, salted hashing algorithms (e.g., bcrypt or Argon2) in order to further enhance the encryption of password hashes. Future work should include secure-coding training for development teams, routine automated scanning of web applications, and red-team exercises to validate the effectiveness of these changes.

Lastly, we would like to thank the Computer Science (CS) department at The University of Texas at San Antonio (UTSA) and the faculty and staff of the penetration testing (pentesting) class for providing the platform and resources to complete this assessment.

Contents

Figures/Tables/Links

2 Introduction

Quick introduction to this section. This section includes…

The section includes the background which explains the purpose of the pentest. In addition, it provides the scope of the assessment and the organization that each section belongs to.

2.1 Background

The CS department at UTSA offers a pentesting class in which the assessment team was designated to perform a multitude of SQL injections in order to test the security of the Mutillidae and DVWA web application.

2.2 Scope

For the assessment, the scope of the test was to be contained in Kali Linux, MS2 and the IP address given by MS2. No other IP addresses were not to be interfered with. Other web applications that were displayed were off-limits and only Mutillidae and DVWA were in-bounds. Any other form of SQL vulnerabilities or links provided by the web applications that were not specified in the videos provided in the instructions were also off-limits.

2.3 Report Organization

The rest of this report is organized as follows: Section 3 provides an overview of the NARO facilities, systems, and processes used. Sections 4 and 5 describe the methodology used by the team and the activities they conducted. The assessment results are found in Section 6, with conclusions and recommended follow-on activities in Section 7. Appendices include additional information related to the assessment.

3 System Overview

The purpose of assessment was to exploit two intentionally vulnerable web applications, Mutillidae and DVWA, which is hosted on a MS2 VM running on Apache, PHP 5, and MySQL 5.0.51a-3ubuntu5. Both applications use default credentials and weak, unsalted MD5 hashing, mimicking common real-world misconfigurations. This controlled setup enabled systematic error- and union-based SQL injection attacks to extract user credentials, database records, and guestbook data, highlighting the severe risks of inadequate input validation and weak password hashing in web applications.

3.1 Mutillidae

Mutillidae is an OWASP-aligned training application that runs on Apache with PHP 5 and connects to a MySQL database. It provides a variety of intentionally vulnerable pages designed to illustrate common web application security flaws. In particular, the User Info page fails to sanitize user inputs before embedding them in SQL queries, making it an ideal target for error-based SQL injection.

Mutillidae: Born to be Hacked

Version: 2.1.19 Security Level: 0 (Hosed) Hints: Disabled (0 - I try harder) Not Logged In

Figure 1: Image of Mutillidae

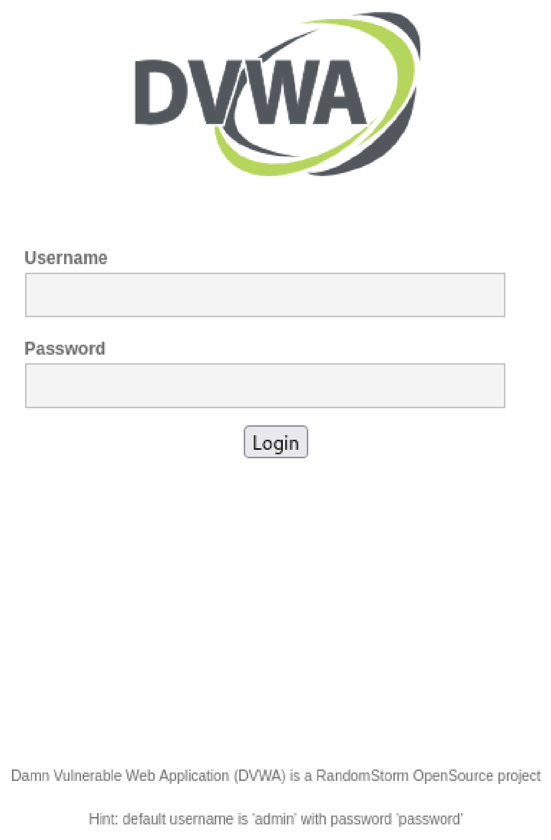
When exploited, this vulnerability surfaces detailed database error messages, which can be leveraged to enumerate schema information and extract sensitive user credentials. This controlled demonstration highlights how inadequate server-side input validation can lead to complete compromise of application data.

Mutillidae is originally configured to look for certain vulnerabilities but must be changed before completing this assessment and this configuration can be found in Section 5.1 Error-Based Injection. The assessment activity can also be found in Section 5.1 Error-Based Injection.

The full results of the compromised data can be found in Appendix A: Full Summary of Error-Based Injection Records.

3.2 DVWA

The Damn Vulnerable Web Application (DVWA) is a PHP and MySQL-based platform specifically designed to educate and test common-web application vulnerabilities. It features three different security configuration levels – low, medium, and high. This adjusts the difficulty in which a user can test the security of the application. For this assessment, the security level of DVWA was set to low which disables input filtering and other protections which exposes classic SQL injection, cross-site scripting, and command-injection flaws. The application uses default credentials that are provided in small-sized font at the bottom of the log-in page which is displayed in the image below.



Damn Vulnerable Web Application (DVWA) is a RandomStorm OpenSource project

Hint: default username is 'admin' with password 'password'

Figure 2: Image of DVWA Log-in Page w/ Default Credentials Shown

This configuration is purposely utilized in order to simplify initial access to the application to shift the focus on exploiting backend weaknesses rather than authentication challenges. DVWA’s user- and guestbook-management databases are both embedded in their SQL injection vulnerability module and is the focal point for the union-based injection.

The assessment activities can be found in Section 5.2 Union-Based Injection along with the guestbook database which can be found in Section 5.4 Guestbook Database.

The complete results of this activity can be found in Appendix B: SQL Statements of Union-Based Injection.

4 Assessment Methodology

The team used the Penetration Testing Execution Standard (PTES) methodology located at: http://www.pentest-standard.org/index.php?Main_page . This methodology is a commonly utilized industry practice that breaks down an assessment into several different parts which provides consistency and thoroughness throughout an assessment.

The PTES methodology is separated into 7 phases:

- Pre-Engagement Interactions
- Intelligence Gathering
- Threat Modeling
- Vulnerability Analysis
- Exploitation
- Post-Exploitation
- Reporting

In the pre-engagement interactions phase, the scope, objectives, and rules of engagement were clearly defined in the instructions. The assessment was limited to two target systems which were Mutillidae and DVWA that are hosted on MS2 and a Kali Linux machine where we were able to utilize their web application. The team was unable to utilize or exploit any other vulnerabilities that were available except for the ones specified in the videos below:

https://www.youtube.com/watch?v=5ulehtDTuvE (Mutillidae).

https://www.youtube.com/watch?v=TITMDw1KD_8 (DVWA).

During the intelligence gathering phase, we were able to identify the vulnerabilities via the video links that were provided above. The YouTuber NFE Systems Ltd explained that entering characters into the Mutillidae database resulted in an error message that exposed the vulnerability and used the URL in the DVWA system to expose the vulnerability. He explained techniques that could be employed to understand how user inputs are processed which set the stage for identifying injection points.

Based off the information obtained from the intelligence gathering phase, two threat models were established:

- Threat model (Mutillidae)
  - Target: Mutillidae Input Validation
  - Attack Vector: Error message contains SQL injection method

    - Impact Potential: Data exfiltration, unauthorized authentication, and release of sensitive, personal information
- Threat model (DVWA)
    - Target: DVWA URL Input
    - Attack Vector: Change the URL to utilize SQL injection
    - Impact Potential: Data exfiltration, release of sensitive, personal information, unauthorized authentication

The risk was determined to be high as there was a serious potential for data leakage that could lead to the loss of personal information.

There are two main vulnerabilities that were identified in the videos:

- The Mutillidae application displays a diagnostic information section that displays the SQL statement that can be exploited to bypass the authentication.
- The DVWA application contains a vulnerability in the URL where we can utilize an SQL statement to display pertinent information.

During the vulnerability analysis phase, we manually tested inputs with common SQL injection payloads to validate whether inputs were properly sanitized. The presence of SQL error messages and successful information leaks confirmed the existence of both error- and union-based injection vulnerabilities.

In the exploitation phase, we successfully used SQL injections to retrieve sensitive information from both applications. For Mutillidae, we were able to extract 16 full user records which can be found in Appendix A: Full Summary of Error-Based Injection Records. For DVWA, we used union-based injections to enumerate schema information and dump data from both the users and guestbook table.

Post-exploitation included using generative AI (ChatGPT) to attempt to crack password hashes retrieved from the users database. All of the hashes were successfully cracked, demonstrating the risk of using unsalted or outdated hashing algorithms in user authentication systems. Although no privilege escalation or lateral movements was attempted, the actions taken simulated real-world data exfiltration scenarios.

Finally, the reporting phase involved organizing the activities, findings, and remediation recommendations into this report. Each section is aligned with the PTES methodology and designed to communicate the technical and operational impact of the vulnerabilities discovered during the assessment.

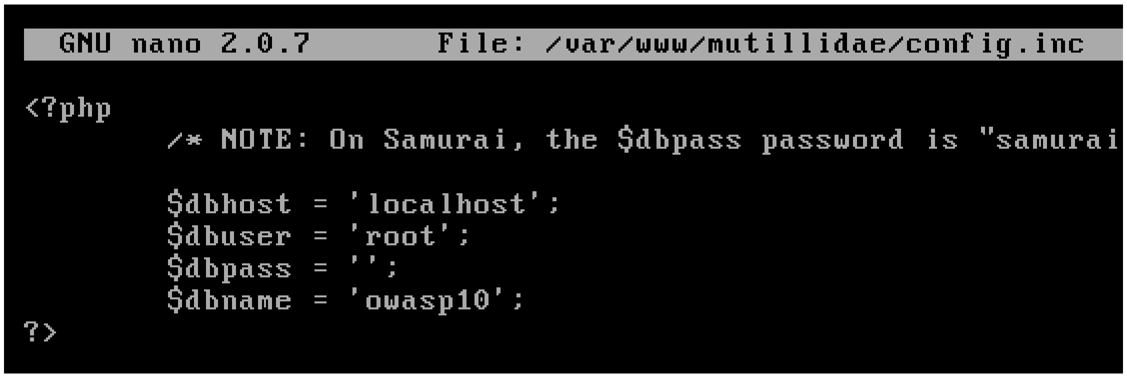
This summary was written with the help of ChatGPT.

5 Assessment Activities

This section covers the high-level activities performed by the assessment team during the penetration test of the Mutillidae and DVWA applications. Each subsection outlines a distinct phase of the testing process, including application configuration, exploitation techniques, and data extraction efforts.

5.1 Error-Based Injection

At the beginning of the assessment, the team had to reconfigure the database name on Mutillidae from “metasploit” to “owasp10” on the MS2 VM in order to expose the SQL injection vulnerability as shown in the image below.



```
  GNU nano 2.0.7        File: /var/www/mutillidae/config.inc

<?php
        /* NOTE: On Samurai, the $dbpass password is "samurai

        $dbhost = 'localhost';
        $dbuser = 'root';
        $dbpass = '';
        $dbname = 'owasp10';
?>
```

Figure 3: Image of Mutillidae Reconfiguration

Once the team reconfigured Mutillidae, we were able to access the vulnerability on the web application. We then navigate to the OWASP Top 10 tab, the A1 – Injection Tab, the SQLi – Extract Data tab and finally, the User Info tab.

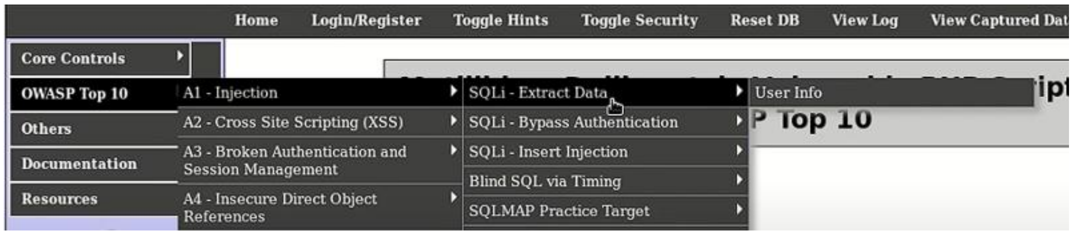
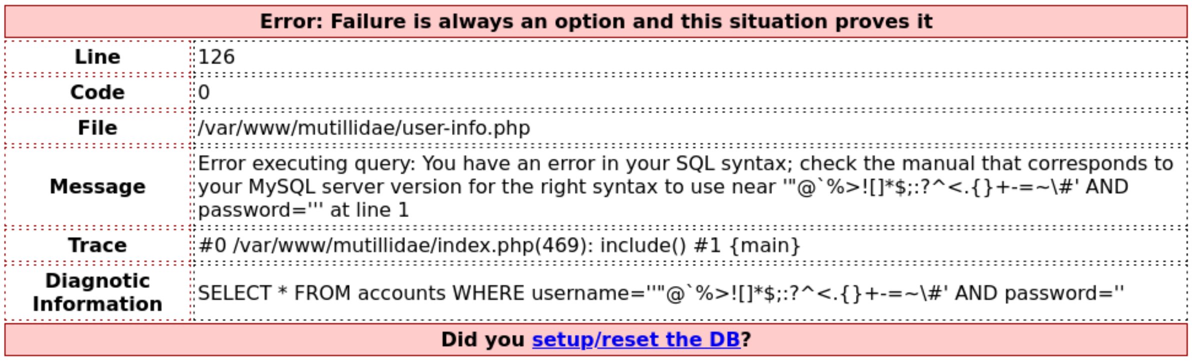


Figure 4: Image of Navigation to vulnerability

We then enter in a sequence of special characters, which are '"@`%>![]*$;:?^<.{}+-=~\#, in order to have the application display an error message. This error message provides the entryway for the team to make their injection.



Error: Failure is always an option and this situation proves it	
Line	126
Code	0
File	/var/www/mutillidae/user-info.php
Message	Error executing query: 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 '"@`%>![]*$;:?^<.{}+-=~\#' AND password='' at line 1
Trace	#0 /var/www/mutillidae/index.php(469): include() #1 {main}
Diagnotic Information	SELECT * FROM accounts WHERE username=''"@`%>![]*$;:?^<.{}+-=~\#' AND password=''
Did you setup/reset the DB?	

Figure 5: Image of Error message on after login attempt

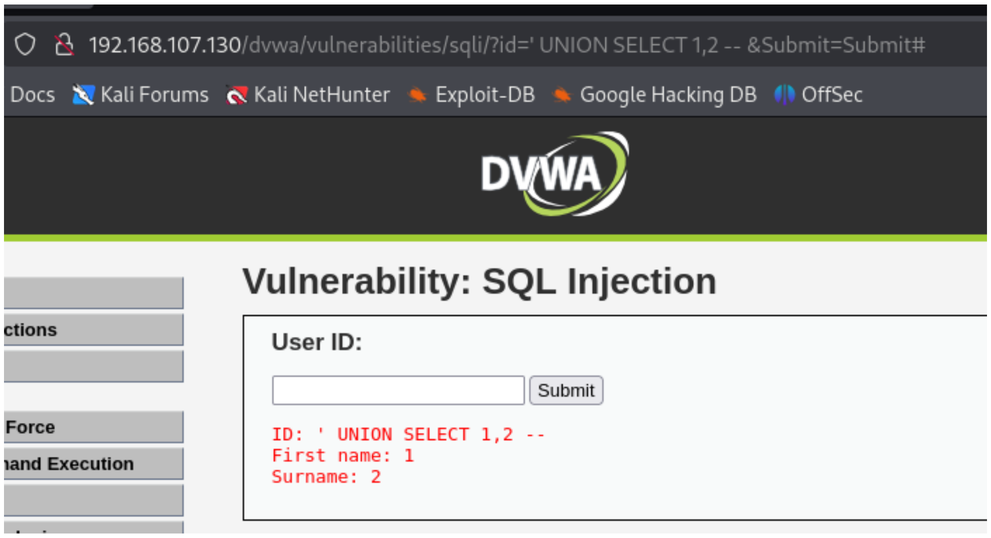
From this error message, we can see that in the Diagnostic Information portion there is an SQL statement that shows what the error was. We can then leverage this statement to input a statement that will always be true which is (’ or “a”=“a” #). This statement claims that the username input is empty or “a” equals “a” which is always true. To bypass the password authentication, we can use the “#” syntax which acts as a comment. Making the username portion always true and negating the password portion allowed the team to access 16 records that contained usernames, passwords, and user-specific signatures. The full list can be found in Appendix A: Full Summary of Error-Based Injection Records. To look up specific records, the team only needs to replace the aforementioned statement to (admin’ #) to look up the record for admin for example.

5.2 Union-Based Injection

For this injection, the team first has to log into the DVWA web application and were met with a log in screen. At first glance, it seemed like there was no avenue of approach to take to bypass this security measure. However, the team soon realized that the credentials were located in small print at the bottom of the login screen with the default username being “admin” and the password being “password”. We learned that this lack of security is purposeful as the main goal of this web application is to focus on other vulnerabilities. Once the team was able to access the web application, similarly to the Mutillidae web application, we had to reconfigure the security level from high to low in order to exploit the union-based vulnerability.

After the reconfiguration, we access the SQL Injection tab to start our assessment. The team first enters a single character in order to see what the web application displays. The application displays a message that reads “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”. In the URL, we noticed a portion where we could exploit a union-based injection. Next to the “id=” part, we noticed the single character that we inputted which was ‘. We then add a space and input the statement: order by 5 -- . The “-- " is another syntax on SQL for a comment. We used this statement to try and determine how many columns are in the database and we discovered that there were 2 columns. The video instructions used a tool called “Hackbar” but were not recommended to use it as it was deemed untrustworthy.

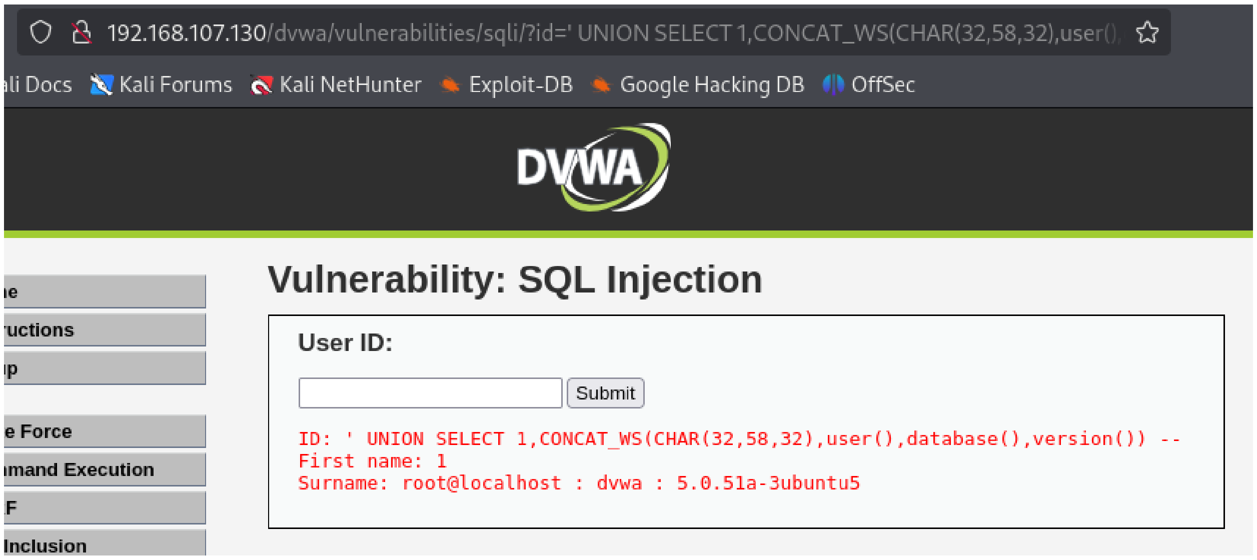
After discovering the number of columns in the database, we switch the statements to "UNION SELECT 1,2" which displays the first name and surname column of the database. We use the surname column of the database to display the information that we are searching for.


192.168.107.130/dvwa/vulnerabilities/sqli/?id=' UNION SELECT 1,2 -- &Submit=Submit#
Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
OffSec
DVWA
ctions
Force
and Execution
Vulnerability: SQL Injection
User ID:
Submit

```
ID: ' UNION SELECT 1,2 --
First name: 1
Surname: 2
```

Figure 6: Image of Union Select

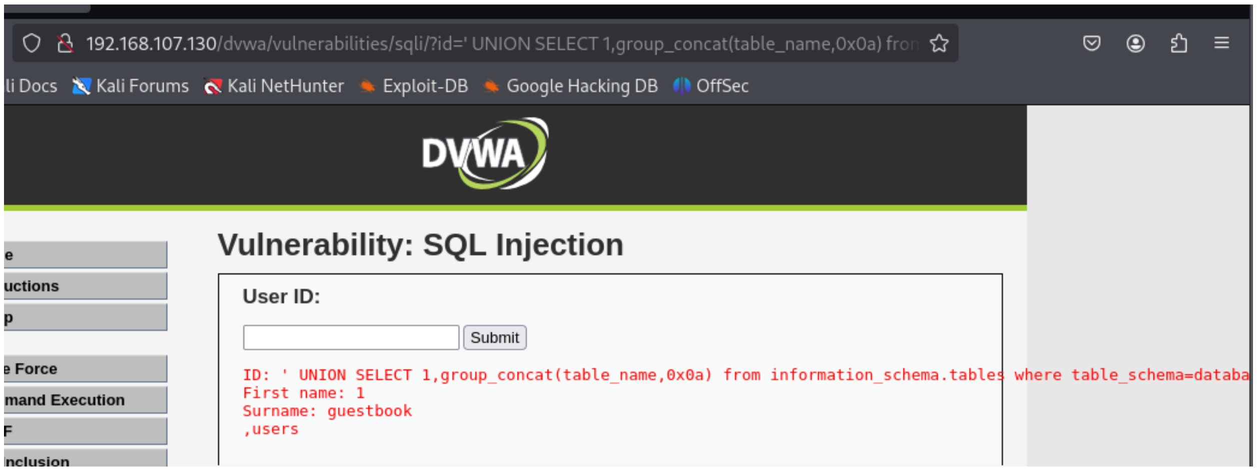
We replace the number 2 in the Union Select statement with a statement that displays the user, database, and the version in the surname column.



```
ID: ' UNION SELECT 1,CONCAT_WS(CHAR(32,58,32),user(),database(),version()) --
First name: 1
Surname: root@localhost : dvwa : 5.0.51a-3ubuntu5
```

Figure 7: Image of Database Information

After gathering information about the database, the team moved onto discovering the tables that were contained in this database which turned out to be "users" and "guestbook".

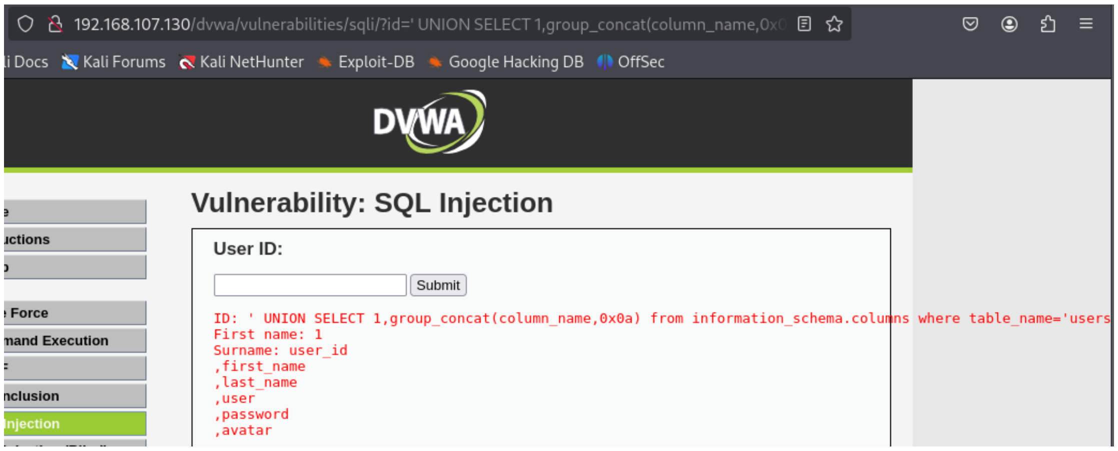


Vulnerability: SQL Injection

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

Figure 8: Image of Tables in Database

The main objective of this assessment was to obtain the records displayed in the “users” table which is what we pursued first. We were able to display all of the columns in the table which contained user_id, first_name, last_name, user, password, and avatar.



Vulnerability: SQL Injection

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

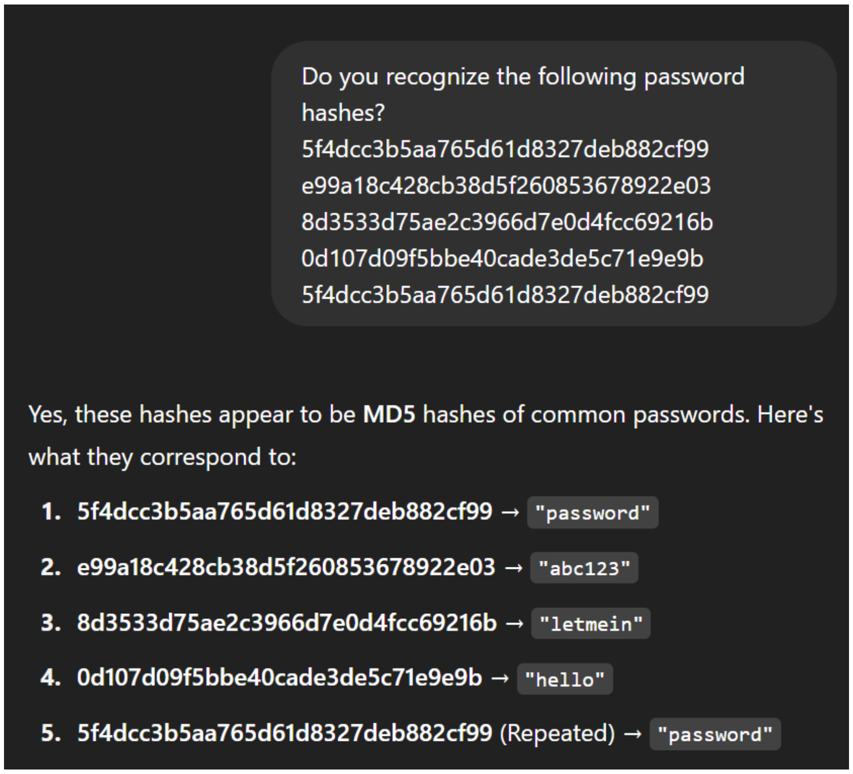
Figure 9: Image of Columns in Users

Finally, we are able to extract the usernames and passwords from 5 accounts in which the usernames were displayed in plaintext and the passwords were displayed in unsalted MD5 hashes.

The full assessment can be found in Appendix B: SQL Statements of Union-Based Injection.

5.3 Cracking Password Hashes

An extra task that was assigned to this assessment was to utilize generative AI to attempt to crack the password hashes that were obtained in the users table. We were able to successfully crack the password hashes as they were MD5 hashes of common passwords.


Do you recognize the following password hashes?
5f4dcc3b5aa765d61d8327deb882cf99
e99a18c428cb38d5f260853678922e03
8d3533d75ae2c3966d7e0d4fcc69216b
0d107d09f5bbe40cade3de5c71e9e9b
5f4dcc3b5aa765d61d8327deb882cf99

Yes, these hashes appear to be **MD5** hashes of common passwords. Here's what they correspond to:

1. **5f4dcc3b5aa765d61d8327deb882cf99** → `"password"`
2. **e99a18c428cb38d5f260853678922e03** → `"abc123"`
3. **8d3533d75ae2c3966d7e0d4fcc69216b** → `"letmein"`
4. **0d107d09f5bbe40cade3de5c71e9e9b** → `"hello"`
5. **5f4dcc3b5aa765d61d8327deb882cf99** (Repeated) → `"password"`

Figure 10: Image of Generative AI Password Cracking

5.4 Guestbook Database

Another additional task that was assigned was to perform a data dump of the guestbook table as well. We had to modify the statements that we used for the users table and tailor it to fit for the guestbook table. To display the columns that were in guestbook, we only needed to change users to guestbook. Similarly, we needed to change the names of the columns from the users table to guestbook.

6 Assessment Results and Recommendations

This section describes in detail the results obtained during the assessment activities outlined in Section 5. These results include identified weaknesses, strengths observed, and technical observations. Each weakness is assigned a severity rating – ranging from low to critical – based on potential impact and ease of exploitation. Recommendations for improvements are provided to act as a guide to improve the overall security posture of the web applications but is not a dictation of what needs to be done.

The vulnerabilities found in Mutillidae and DVWA demonstrate common real-world web application risks, particularly involving poor input validation and the use of weak password storage mechanisms.

6.1 Weaknesses

(Critical) Error-Based SQL Injection on Mutillidae User Info Page.

Justification: An error-based SQL injection attack successfully retrieved 16 full user records, including usernames, passwords, and user-specific signatures, demonstrating full compromise of user credential data.

Mitigation: Implement prepared statements (parameterized queries) to prevent direct execution of user-supplied data within SQL queries. Additionally, implement server-side input validation and whitelisting to reject unexpected inputs before they interact with the database. Employ Web Application Firewalls (WAFs) to detect and block common SQL injection patterns at the network level.

(Critical) Union-Based SQL Injection on DVWA User Authentication Modules.

Justification: The assessment team was able to dump the user, database, and version information, list all tables and columns, and retrieve user credentials and guestbook entries through union-based payloads.

Mitigation: In production, prepared statements should always be utilized to prevent dynamic query manipulation by user input. Additionally, least-privilege database accounts should be employed where web applications can connect to databases using accounts that can’t run UNION, SELECT, DROP, or ALTER operations unless absolutely necessary. Finally, implement strict input validation and deploy a WAF to inspect and sanitize incoming HTTP traffic against union-based attacks.

(High) Weak Password Hashing in User Credential Storage.

Justification: Password hashes extracted from the users table were easily cracked using ChatGPT, demonstrating that attackers could recover plaintext or unsalted password hashes without significant effort and with the help of generative AI.

Mitigations: Modern applications should take advantage of adaptive, salted password hashing algorithms such as bcrypt, scrypt, or Argon2, which are designed to be computationally expensive and resist brute-force attacks. Salted hashes prevent identical passwords from producing identical hashes, further protecting stored credentials even if hashes are leaked.

(Medium) Exposure of Database Information via Error Messages.

Justification: Information leakage provides attackers with reconnaissance data needed to craft effective injection payloads and escalates the likelihood of full system compromise which was observed during the error-based injection.

Mitigations: Configure applications to suppress detailed error messages from being sent to end users. Instead, return generic error messages while logging detailed error output securely on the server side. Additionally, perform input validation to prevent SQL errors being triggered by malformed inputs.

6.2 Strengths

Despite being an intentionally vulnerable environment for educational purposes, several elements of the Mutillidae and DVWA lab setup mimicked effective real-world security practices.

Controlled and Isolated Testing Environment:

- The web applications operated within a dedicated MS2 and Kali Linux VM with the network adapters set to Host-Only, ensuring that any exploitation activity was contained and posed no outside risk to external systems. In production, isolating sensitive or experimental systems within segmented networks is a recommended best practice to minimize the impact potential.

Realistic Vulnerability Scenarios:

- The lab accurately simulated common real-world misconfigurations and coding flaws, such as insufficient input validation and outdated password hashing. Exposure to realistic attack surfaces enhances tester proficiency and prepares them for field operations.

Configurable Security Levels in DVWA:

- DVWA's ability to adjust its security settings highlights how application security controls can influence vulnerability exposure. This feature underscores the importance of defense-in-depth strategies, where layered security measures combine to harden systems against attacks.

Clear Attack Path for Educational Focus:

- The environment was purposefully structured to expose the relationship between vulnerability exploitation and data compromise. In real-world scenarios, understanding how an attacker progresses through the attack chain is critical for designing effective detection and mitigation controls.

6.3 Observations

Ability to Comment Out SQL Queries Using # and -- Characters

Description: During testing, we were able to use SQL comment characters (# and --) to terminate legitimate portions of SQL statements and inject malicious payloads. This is a common SQL injection technique that allows attackers to manipulate query logic and bypass authentication or extract sensitive data.

Recommendation: The most effective mitigation is to use prepared statements for all database interactions. Prepared statements separate SQL code from user input, preventing attackers from injecting or terminating queries using comment characters. In addition, server-side input validation should be enforced to reject unexpected special characters, and WAFs should be configured to detect and block common SQL injection patterns, including misuse of comment syntax.

Default Credentials Displayed on Login Screen

Description: During the assessment, the team noted that DVWA displayed its default credentials (admin/password) in small print at the bottom of the login page. While this may be acceptable in a controlled lab setting for educational purposes, exposing credentials directly on any publicly available interface represents a major security issue in real-world scenarios and should not be mirrored.

Recommendation: In production systems, default credentials should be immediately changed during setup and never displayed or accessible through the user interface. Initial setup instructions and administrative credentials should be securely communicated out-of-band. Additionally, implement forced password changes on first login and enforce strong password policies.

SQL Injection Payloads Required Specific Spacing and Syntax

Description: During exploitation, it was observed that successful SQL injection payloads required precise spacing, with no additional whitespace or characters. Minor deviations like unexpected spaces or characters caused the injection attempt to fail. This indicates that the application's SQL query construction is fragile and highly dependent on the exact structure of user input.

Recommendation: Applications should never rely on specific input formatting for security. Instead use prepared statements that strictly separate SQL code from user-supplied code. Additionally, input normalization and robust input validation should be applied before any data is passed to the database engine. This ensures that application behavior is consistent regardless of user input formatting and reduces the risk of injection attacks.

Multiple User Accounts Share the Same Weak Password

Description: During the error-based injection on the Mutillidae User Info page, the team retrieved 16 user records and observed a significant number of user accounts that shared the same password: "password". The use of identical, weak passwords across multiple accounts greatly increase the risk of large-scale credential compromise and lateral movement following an initial breach.

Recommendation: Enforce strong password policies that require a minimum password complexity (e.g., a mix of uppercase and lowercase letters, numbers, and special characters). Implement password uniqueness checks to prevent users from choosing common or previously breached passwords. Consider deploying multi-factor authentication (MFA) to reduce the impact of password compromise and conduct regular user security awareness training on password hygiene.

7 Conclusions and Follow-on Activities

The assessment team found that, while the Mutillidae and DVWA applications served as training environments to illustrate common SQL vulnerabilities, they also highlighted critical real-world security challenges. The team was able to identify several vulnerabilities that could allow an attacker to compromise user credentials, gain unauthorized access to sensitive data, and manipulate backend databases. These findings reinforce the idea of implementing strong input validation, secure authentication measures, and complex cryptographic standards in modern web applications.

Of the vulnerabilities identified in Section 6, the team recommends that addressing SQL injection vulnerabilities and improper password storage practices be prioritized. SQL injection remains one of the most impactful and commonly exploited attack vectors, often leading to full system compromise if left unchecked. Similarly, weak password hashing mechanisms expose user credentials to password cracking attacks, significantly increasing the risk of credential theft and subsequent unauthorized system access.

For follow-on activities, the team recommends the following actions to improve the overall security posture:

- Implement prepared statements for all database interactions to eliminate injection risks.
- Migrate password storage mechanisms to strong, salted hashing algorithms such as bcrypt or Argon2.
- Suppress verbose error messages to limit information disclosure to potential attackers.
- Deploy WAFs to detect and block common injection attempts at the network level.
- Conduct regular security audits and code reviews focusing on input validation and authentication controls.
- Provide developer training on secure coding practices, particularly regarding input handling and secure cryptography.
- Integrate continuous security monitoring and vulnerability scanning into the application development and deployment lifecycle.

By implementing these recommendations, organizations can significantly reduce their attack surface to common web application attacks and strengthen the confidentiality, integrity, and availability of their systems and data.

8 Appendix A: Full Summary of Error-Based Injection Records

8.1 Full List of Obtained Records

Username	Password	Signature
admin	adminpass	Monkey!
adrian	somepassword	Zombie Films Rock!
john	monkey	I like the smell of confunk
jeremy	password	d1373 1337 speak
bryce	password	I Love SANS
samurai	samurai	Carving Fools
jim	password	Jim Rome is Burning
bobby	password	Hank is my dad
simba	password	I am a cat
dreveil	password	Preparation H
scotty	password	Scotty Do
cal	password	Go Wildcats
john	password	Do the Duggie!
kevin	42	Doug Adams rocks
dave	set	Bet on S.E.T. FTW
ed	pentest	Commandline KungFu anyone?

Table 1: Table of Retrieved Records

9 Appendix B: SQL Statements of Union-Based Injection

9.1 SQL Statements

Show all tables:

- group_concat(table_name,0x0a) from information_schema.tables where table_schema=database()

Show all columns (update users to the table you are using if it is not called users):

- group_concat(column_name,0x0a) from information_schema.columns where table_name='users'

Show column data (user and password are the column names and users is the table name):

- group_concat(user, 0x0a,password) from users

Show all columns (update table_name to guestbook to obtain its column data):

- group_concat(column_name,0x0a) from information_schema.columns where table_name='guestbook'

Show column data (comment_id, comment, and name are the column names and guestbook is the table name):

- group_concat(comment_id,0x0a,comment,0x0a,name) from guestbook

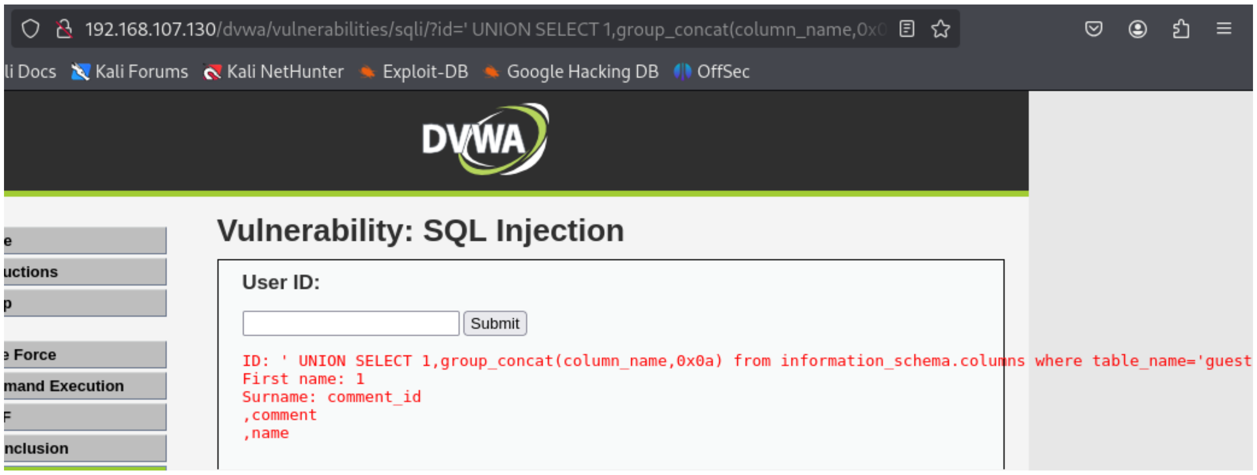
10 Appendix C: Full Summary of Password Cracking/Guestbook Dump

10.1 Password Hashes and Results

5f4dcc3b5aa765d61d8327deb882cf99	Password
e99a18c428cb38d5f260853678922e03	abc123
8d3533d75ae2c3966d7e0d4fcc69216b	letmein
0d107d09f5bbe40cade3de5c71e9e9b	ello
5f4dcc3b5aa765d61d8327deb882cf99	password

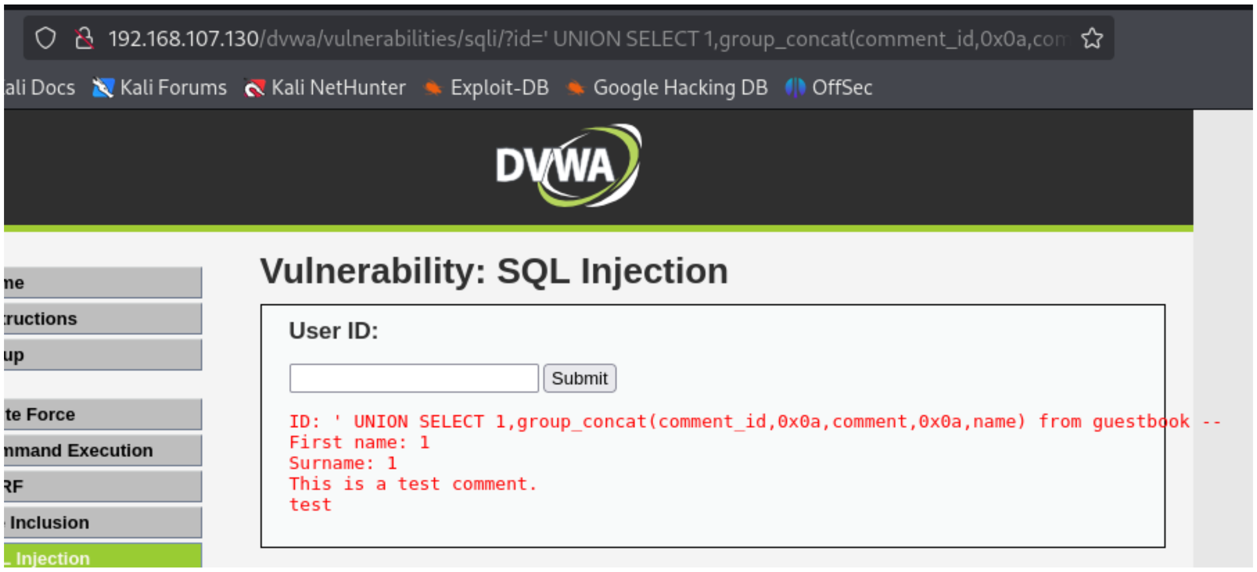
Table 2: Table of Cracked Password Hashes

10.2 Guestbook Dump



```
ID: ' UNION SELECT 1,group_concat(column_name,0x0a) from information_schema.columns where table_name='guest
First name: 1
Surname: comment_id
,comment
,name
```

Figure 11: Image of Guestbook Columns



Vulnerability: SQL Injection

User ID:

```
ID: ' UNION SELECT 1,group_concat(comment_id,0x0a,comment,0x0a,name) from guestbook --
First name: 1
Surname: 1
This is a test comment.
test
```

Figure 12: Image of Guestbook Dump