SQL Injection (SQLi)

SQL Injection (SQLi) is a code injection technique that exploits a vulnerability in an application's software by inserting malicious SQL code into the input fields that are passed to a backend database. This allows the attacker to gain unauthorized access to or manipulate the database.

How SQL Injection Works

When an application directly incorporates user input into SQL queries without properly validating or escaping it, an attacker can craft malicious input to alter the query's behavior. This results in unauthorized actions such as data retrieval, modification, or deletion.

For example, consider a login page where the application performs the following SQL query:

sql
Copy code

```sql
SELECT * FROM users WHERE username = 'input' AND password = 'input';
```

If the application does not sanitize the user input, an attacker could enter:

sql
Copy code

```sql
' OR 1=1 --
```

This would modify the query to:

sql
Copy code

```sql
SELECT * FROM users WHERE username = '' OR 1=1 --' AND password = '';
```

This condition (`1=1`) is always true, which allows the attacker to bypass authentication.

Types of SQL Injection Attacks

1. **Classic SQL Injection**
    - ○ An attacker injects malicious SQL code to manipulate the query and retrieve data from the database.

Example:
sql
Copy code

```
SELECT * FROM users WHERE username = '' OR 1=1 --;
```

- ○
- ○ This would log the attacker in as the first user in the database.

2. **Blind SQL Injection**
    - ○ In blind SQL injection, the attacker does not directly see the error message but can infer information about the database through the application's behavior.
    - ○ **Boolean-based blind SQL injection:** The attacker sends a query that returns a different result based on the truth value of a condition.
        - ■ Example: `' AND 1=1 --` (returns valid results) vs. `' AND 1=2 --` (returns no results).
    - ○ **Time-based blind SQL injection:** The attacker uses time delays to infer if a query is true or false.
        - ■ Example: `' AND IF(1=1, SLEEP(5), 0) --` (delays the response if the condition is true).
3. **Union-based SQL Injection**
    - ○ The attacker uses the `UNION` SQL operator to combine the results of the original query with the results of another SELECT query.

Example:
sql
Copy code

```
' UNION SELECT null, null, username, password FROM users --;
```

- ○
- ○ This can reveal sensitive data like usernames and passwords.

4. **Out-of-Band SQL Injection**
    - ○ This type of attack occurs when the attacker cannot retrieve data through the same channel as the injected query but instead forces the database to send data to a different location (like a remote server).
    - ○ Example: Using DNS queries to extract information from the database.

Common SQL Injection Attack Scenarios

1. **Authentication Bypass**
    - An attacker bypasses login forms by injecting SQL code that always returns a true value, allowing them to log in without valid credentials.
2. **Data Extraction**
    - Attackers use SQL injection to extract sensitive data, such as usernames, passwords, credit card numbers, etc., from the database.
3. **Database Modification**
    - Attackers can modify the database by altering data, deleting records, or inserting malicious content.
4. **Remote Code Execution**
    - In rare cases, attackers can execute system-level commands on the server via SQL injection, leading to full system compromise.

Example Attack: SQL Injection on a Login Page

Suppose the website's login form executes the following SQL query:

sql
Copy code

```sql
SELECT * FROM users WHERE username = '[user_input]' AND
password = '[password_input]';
```

An attacker might input the following:

- **Username:** `admin' OR '1'='1`
- **Password:** `anything`

This results in the following SQL query:

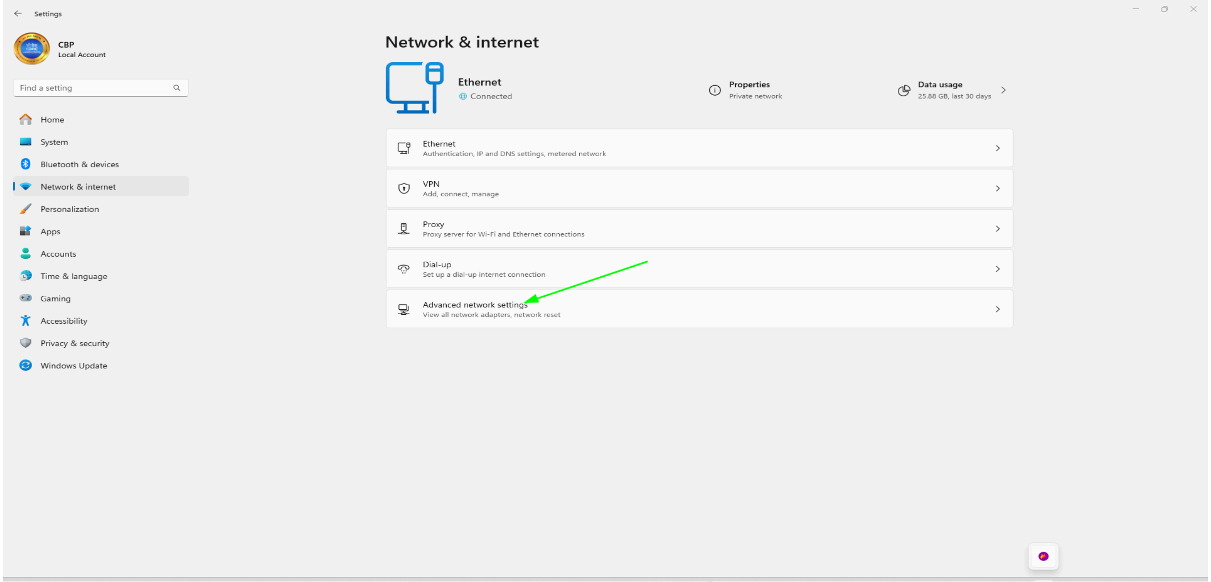
sql
Copy code

```sql
SELECT * FROM users WHERE username = 'admin' OR '1'='1' AND
password = 'anything';
```

Since `'1'='1'` is always true, the query will return valid results, bypassing the password check and logging the attacker in as the first user found (e.g., `admin`).

Prevention of SQL Injection

1. **Use Parameterized Queries / Prepared Statements**
   - Parameterized queries ensure that user input is treated as data, not executable code. This approach is language-specific and involves separating SQL logic from data.



Network & internet

Ethernet
Connected

Properties
Private network

Data usage
25.88 GB, last 30 days

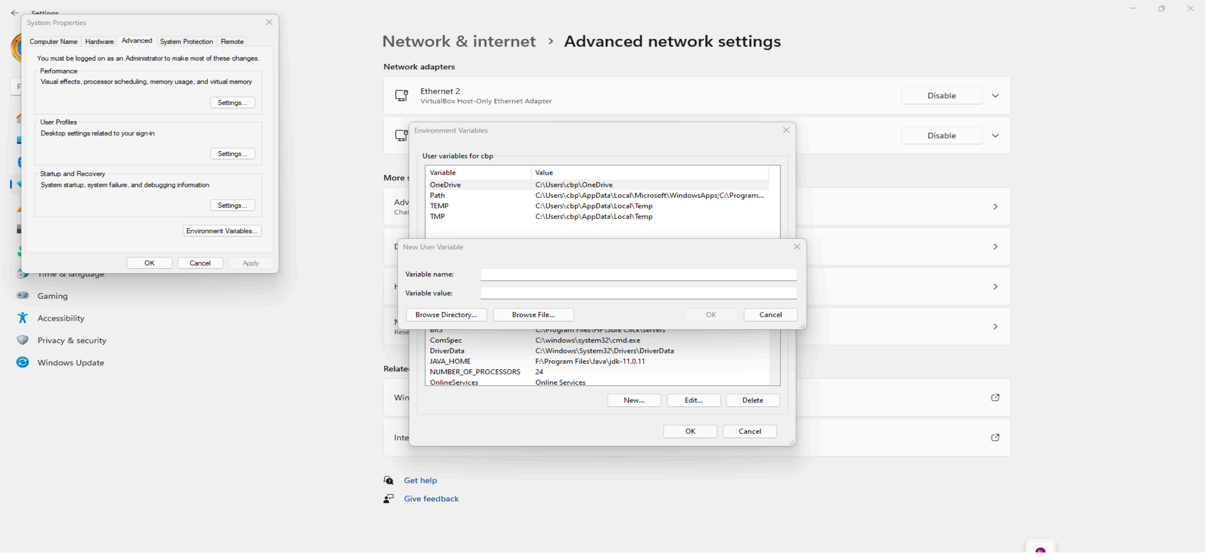
Ethernet
Authentication, IP and DNS settings, metered network

VPN
Add, connect, manage

Proxy
Proxy server for Wi-Fi and Ethernet connections

Dial-up
Set up a dial-up internet connection

Advanced network settings
View all network adapters, network reset



You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles
Desktop settings related to your sign-in

Startup and Recovery
System startup, system failure, and debugging information

Network & internet > Advanced network settings

Network adapters

Ethernet 2
VirtualBox Host-Only Ethernet Adapter

Environment Variables

User variables for cbp

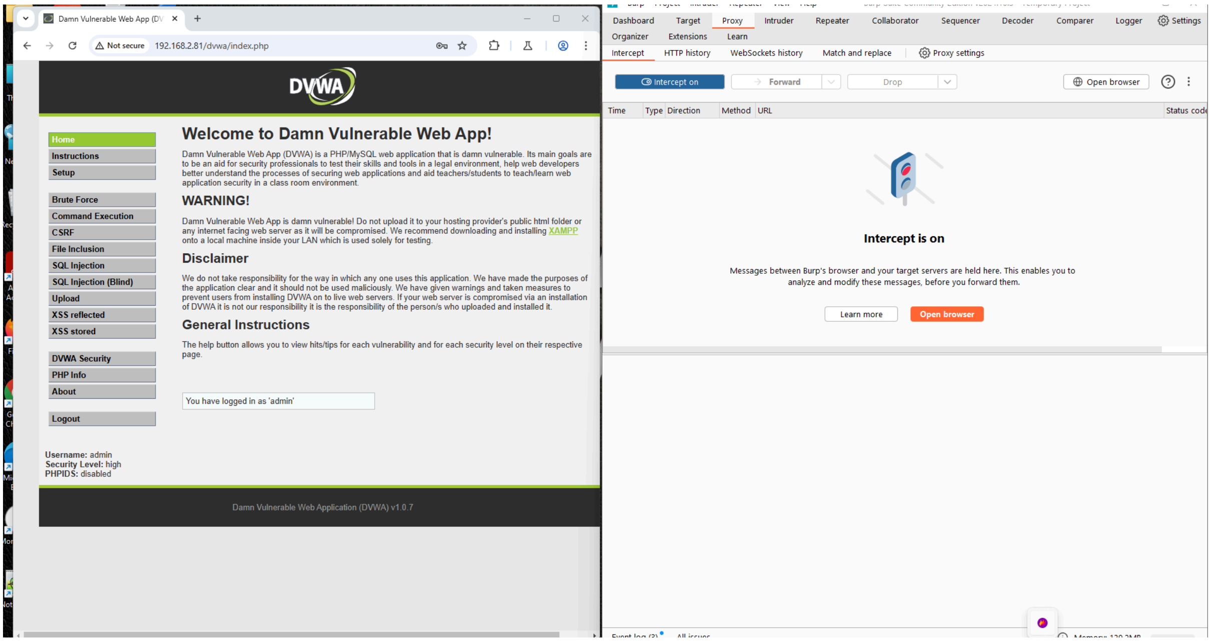
Variable	Value
OneDrive	C:\Users\cbp\OneDrive
Path	C:\Users\cbp\AppData\Local\Microsoft\WindowsApps;C:\Program...
TEMP	C:\Users\cbp\AppData\Local\Temp
TMP	C:\Users\cbp\AppData\Local\Temp

New User Variable

Variable name:

Variable value:

BRS	C:\Program Files\HP\Sure Click\servers
ComSpec	C:\windows\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
JAVA_HOME	F:\Program Files\Java\jdk-11.0.11
NUMBER_OF_PROCESSORS	24
OnlineServices	Online Services



Welcome to Damn Vulnerable Web App!

Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a class room environment.

WARNING!

Damn Vulnerable Web App is damn vulnerable! Do not upload it to your hosting provider's public html folder or any internet facing web server as it will be compromised. We recommend downloading and installing XAMPP onto a local machine inside your LAN which is used solely for testing.

Disclaimer

We do not take responsibility for the way in which any one uses this application. We have made the purposes of the application clear and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your web server is compromised via an installation of DVWA it is not our responsibility it is the responsibility of the person/s who uploaded and installed it.

General Instructions

The help button allows you to view hits/tips for each vulnerability and for each security level on their respective page.

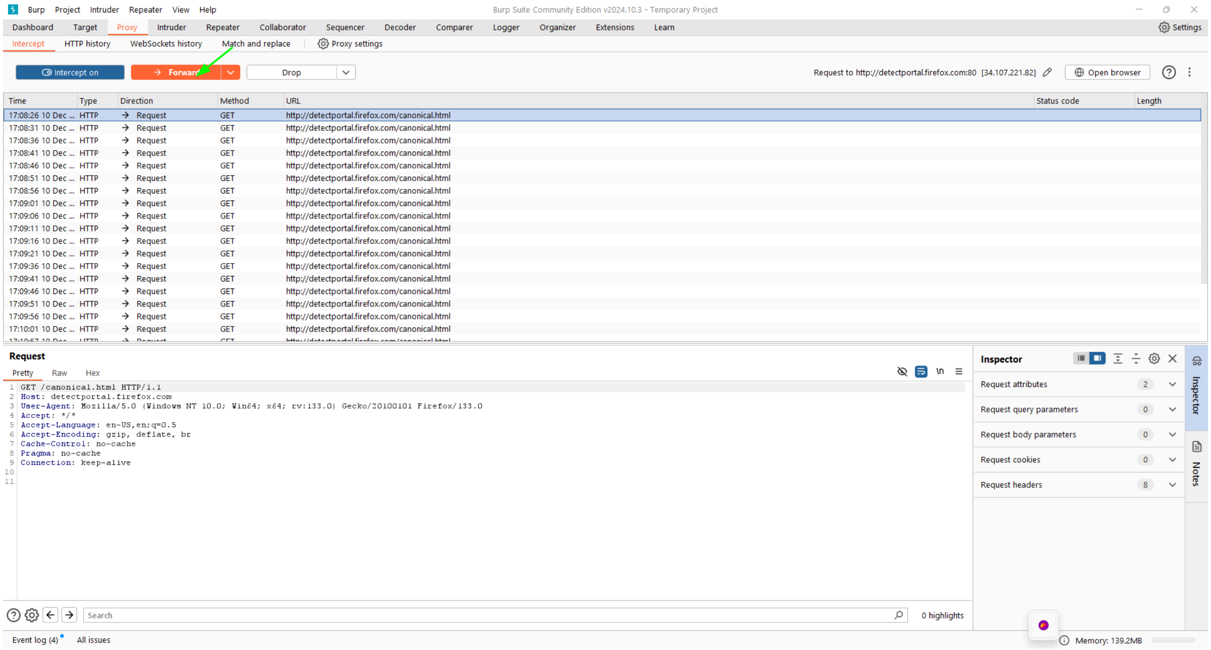
You have logged in as 'admin'

Username: admin
Security Level: high
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.0.7

Intercept is on

Messages between Burp's browser and your target servers are held here. This enables you to analyze and modify these messages, before you forward them.



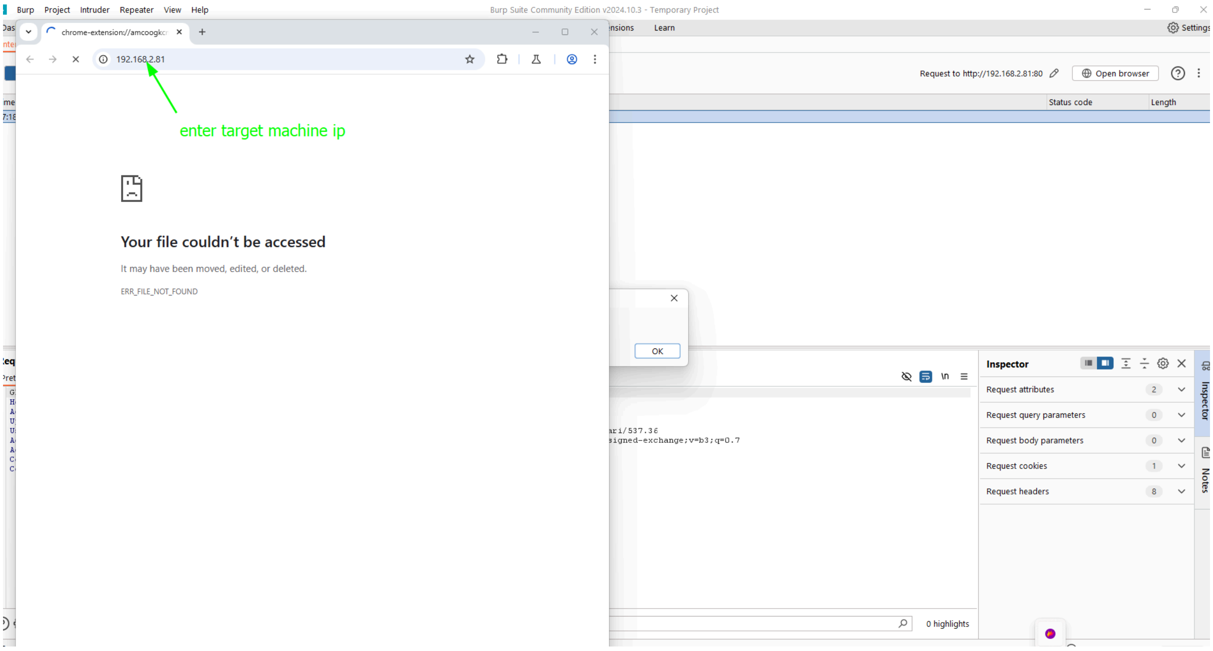
Request to http://detectportal.firefox.com:80 [34.107.221.82]

Time	Type	Direction	Method	URL	Status code	Length
17:08:26 10 Dec ...	HTTP	→ Request	GET	http://detectportal.firefox.com/canonical.html		
17:08:31 10 Dec ...	HTTP	→ Request	GET	http://detectportal.firefox.com/canonical.html		
17:08:36 10 Dec ...	HTTP	→ Request	GET	http://detectportal.firefox.com/canonical.html		
17:08:41 10 Dec ...	HTTP	→ Request	GET	http://detectportal.firefox.com/canonical.html		
17:08:46 10 Dec ...	HTTP	→ Request	GET	http://detectportal.firefox.com/canonical.html		
17:08:51 10 Dec ...	HTTP	→ Request	GET	http://detectportal.firefox.com/canonical.html		
17:08:56 10 Dec ...	HTTP	→ Request	GET	http://detectportal.firefox.com/canonical.html		
17:09:01 10 Dec ...	HTTP	→ Request	GET	http://detectportal.firefox.com/canonical.html		
17:09:06 10 Dec ...	HTTP	→ Request	GET	http://detectportal.firefox.com/canonical.html		
17:09:11 10 Dec ...	HTTP	→ Request	GET	http://detectportal.firefox.com/canonical.html		
17:09:16 10 Dec ...	HTTP	→ Request	GET	http://detectportal.firefox.com/canonical.html		
17:09:21 10 Dec ...	HTTP	→ Request	GET	http://detectportal.firefox.com/canonical.html		
17:09:36 10 Dec ...	HTTP	→ Request	GET	http://detectportal.firefox.com/canonical.html		
17:09:41 10 Dec ...	HTTP	→ Request	GET	http://detectportal.firefox.com/canonical.html		
17:09:46 10 Dec ...	HTTP	→ Request	GET	http://detectportal.firefox.com/canonical.html		
17:09:51 10 Dec ...	HTTP	→ Request	GET	http://detectportal.firefox.com/canonical.html		
17:09:56 10 Dec ...	HTTP	→ Request	GET	http://detectportal.firefox.com/canonical.html		
17:10:01 10 Dec ...	HTTP	→ Request	GET	http://detectportal.firefox.com/canonical.html		

Request

```
GET /canonical.html HTTP/1.1
Host: detectportal.firefox.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:133.0) Gecko/20100101 Firefox/133.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Cache-Control: no-cache
Pragma: no-cache
Connection: keep-alive


```

Inspector	
Request attributes	2
Request query parameters	0
Request body parameters	0
Request cookies	0
Request headers	8



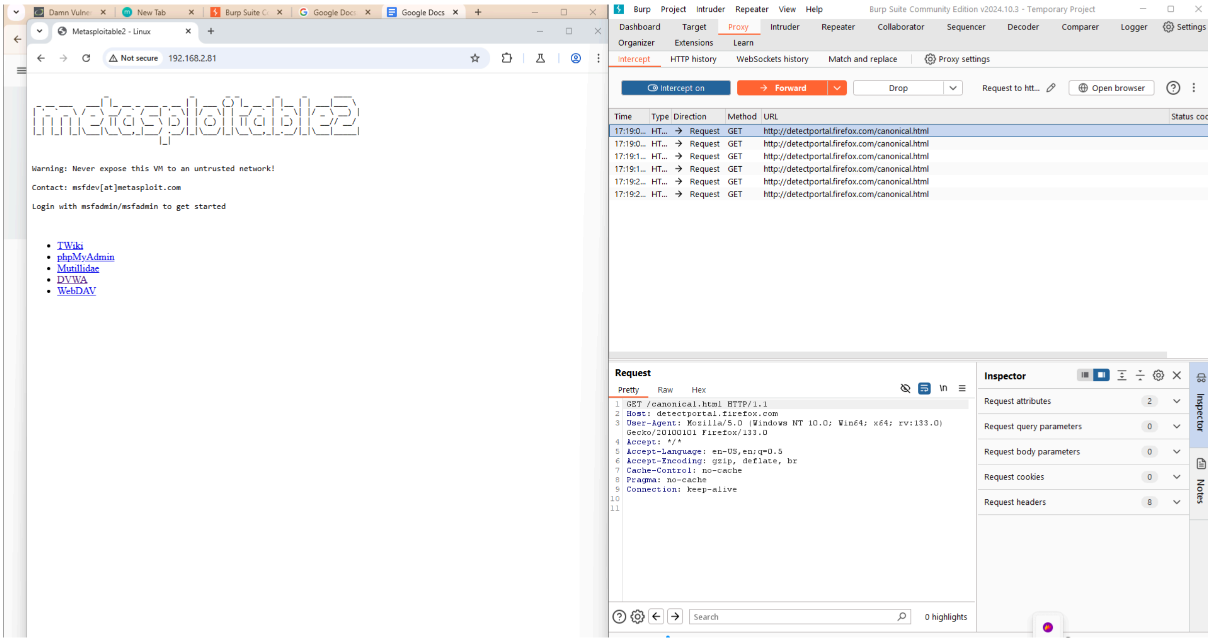
Burp Suite Community Edition v2024.10.3 - Temporary Project
192.168.2.81
Request to http://192.168.2.81:80
Open browser
OK

enter target machine ip

Your file couldn't be accessed

It may have been moved, edited, or deleted.

ERR_FILE_NOT_FOUND



Warning: Never expose this VM to an untrusted network!

Contact: msfdev[at]metasploit.com

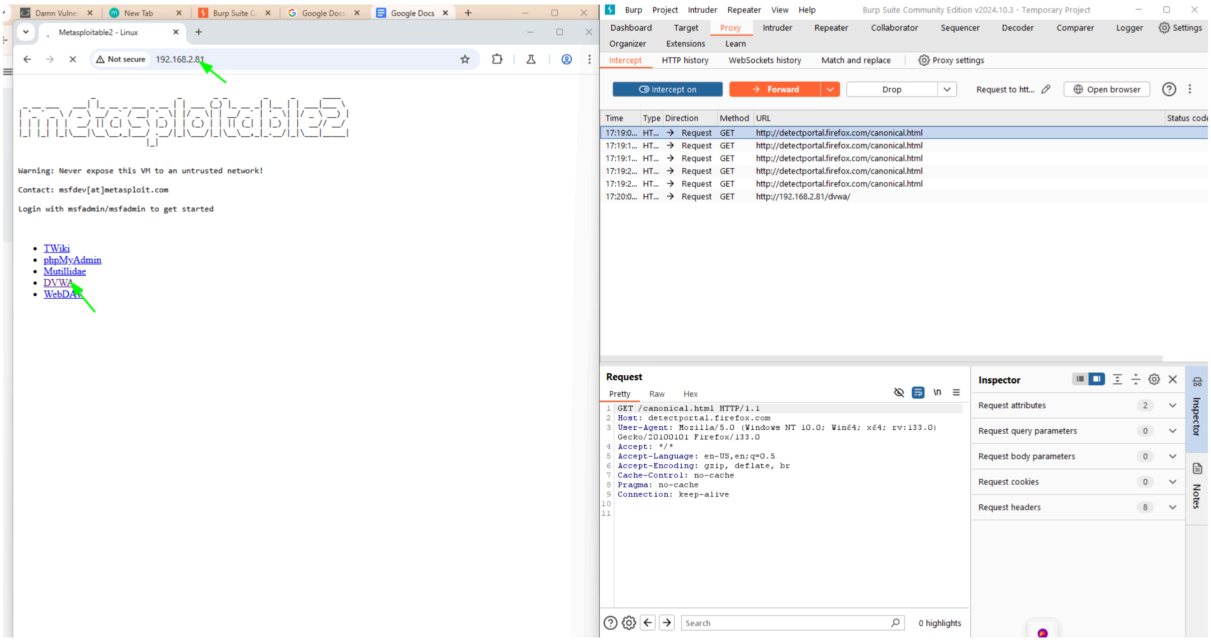
Login with msfadmin/msfadmin to get started

- TWiki
- phpMyAdmin
- Mutillidae
- DVWA
- WebDAV

Time	Type	Direction	Method	URL	Status coc
17:19:0...	HT...	→ Request	GET	http://detectportal.firefox.com/canonical.html	
17:19:0...	HT...	→ Request	GET	http://detectportal.firefox.com/canonical.html	
17:19:1...	HT...	→ Request	GET	http://detectportal.firefox.com/canonical.html	
17:19:1...	HT...	→ Request	GET	http://detectportal.firefox.com/canonical.html	
17:19:2...	HT...	→ Request	GET	http://detectportal.firefox.com/canonical.html	
17:19:2...	HT...	→ Request	GET	http://detectportal.firefox.com/canonical.html	

Request

```
GET /canonical.html HTTP/1.1
Host: detectportal.firefox.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:133.0)
Gecko/20100101 Firefox/133.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Cache-Control: no-cache
Pragma: no-cache
Connection: keep-alive


```



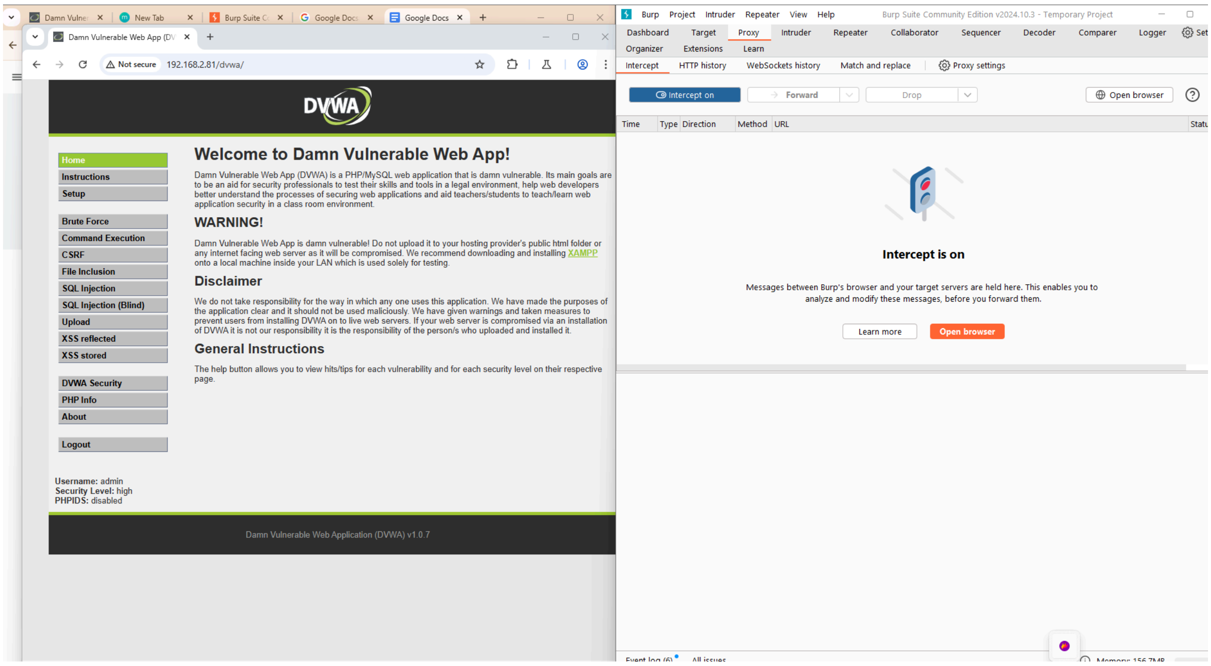
```
metasploitable2

Warning: Never expose this VM to an untrusted network!

Contact: msfdev[at]metasploit.com

Login with msfadmin/msfadmin to get started
```

- TWiki
- phpMyAdmin
- Mutillidae
- DVWA
- WebDAV

Time	Type	Direction	Method	URL	Status code
17:19:0...	HT...	→ Request	GET	http://detectportal.firefox.com/canonical.html	
17:19:1...	HT...	→ Request	GET	http://detectportal.firefox.com/canonical.html	
17:19:1...	HT...	→ Request	GET	http://detectportal.firefox.com/canonical.html	
17:19:2...	HT...	→ Request	GET	http://detectportal.firefox.com/canonical.html	
17:19:2...	HT...	→ Request	GET	http://detectportal.firefox.com/canonical.html	
17:20:0...	HT...	→ Request	GET	http://192.168.2.81/dvwa/	

```
GET /canonical.html HTTP/1.1
Host: detectportal.firefox.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:133.0)
Gecko/20100101 Firefox/133.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Cache-Control: no-cache
Pragma: no-cache
Connection: keep-alive
```

Inspector	
Request attributes	2
Request query parameters	0
Request body parameters	0
Request cookies	0
Request headers	8



Welcome to Damn Vulnerable Web App!

Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a class room environment.

WARNING!

Damn Vulnerable Web App is damn vulnerable! Do not upload it to your hosting provider's public html folder or any internet facing web server as it will be compromised. We recommend downloading and installing XAMPP onto a local machine inside your LAN which is used solely for testing.

Disclaimer

We do not take responsibility for the way in which any one uses this application. We have made the purposes of the application clear and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your web server is compromised via an installation of DVWA it is not our responsibility it is the responsibility of the person/s who uploaded and installed it.

General Instructions

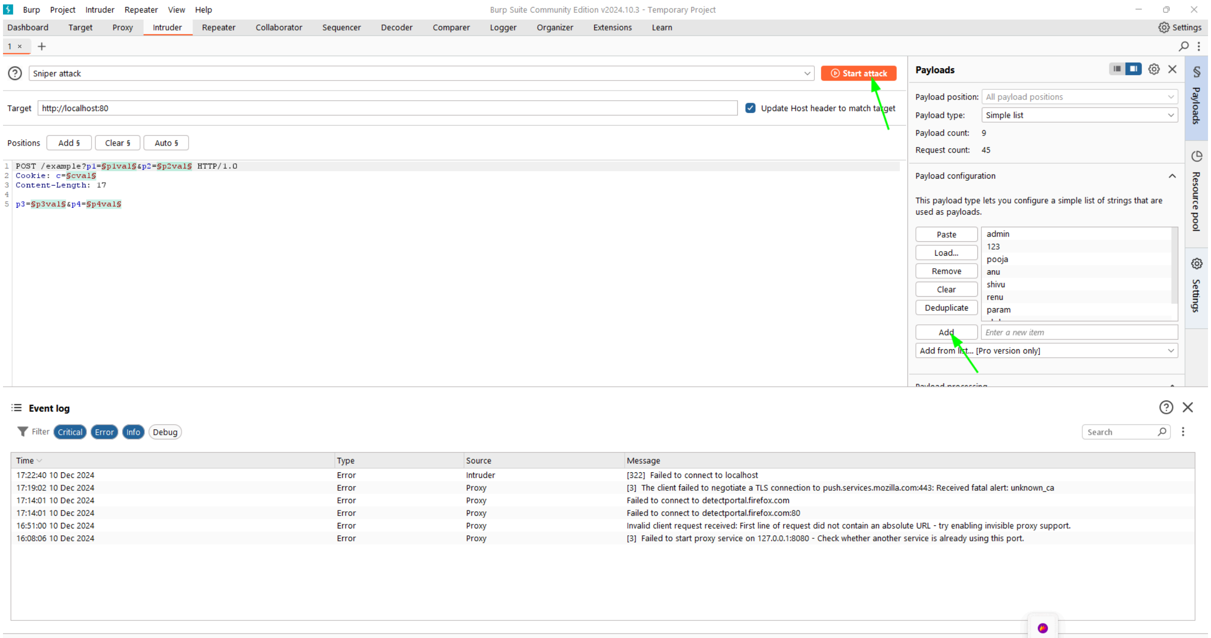
The help button allows you to view hits/tips for each vulnerability and for each security level on their respective page.

Username: admin
Security Level: high
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.0.7

Intercept is on

Messages between Burp's browser and your target servers are held here. This enables you to analyze and modify these messages, before you forward them.



Burp Suite Community Edition v2024.10.3 - Temporary Project
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Settings
Start attack
Positions Add § Clear § Auto §
Paste
Load...
Remove
Clear
Deduplicate
Add
Enter a new item
Add from list... [Pro version only]
Event log
Filter Critical Error Info Debug
Search

Sniper attack

Target http://localhost:80 ☑ Update Host header to match target

```
POST /example?p1=§p1val§&p2=§p2val§ HTTP/1.0
Cookie: c=§cval§
Content-Length: 17

p3=§p3val§&p4=§p4val§
```

Payloads

Payload position: All payload positions

Payload type: Simple list

Payload count: 9

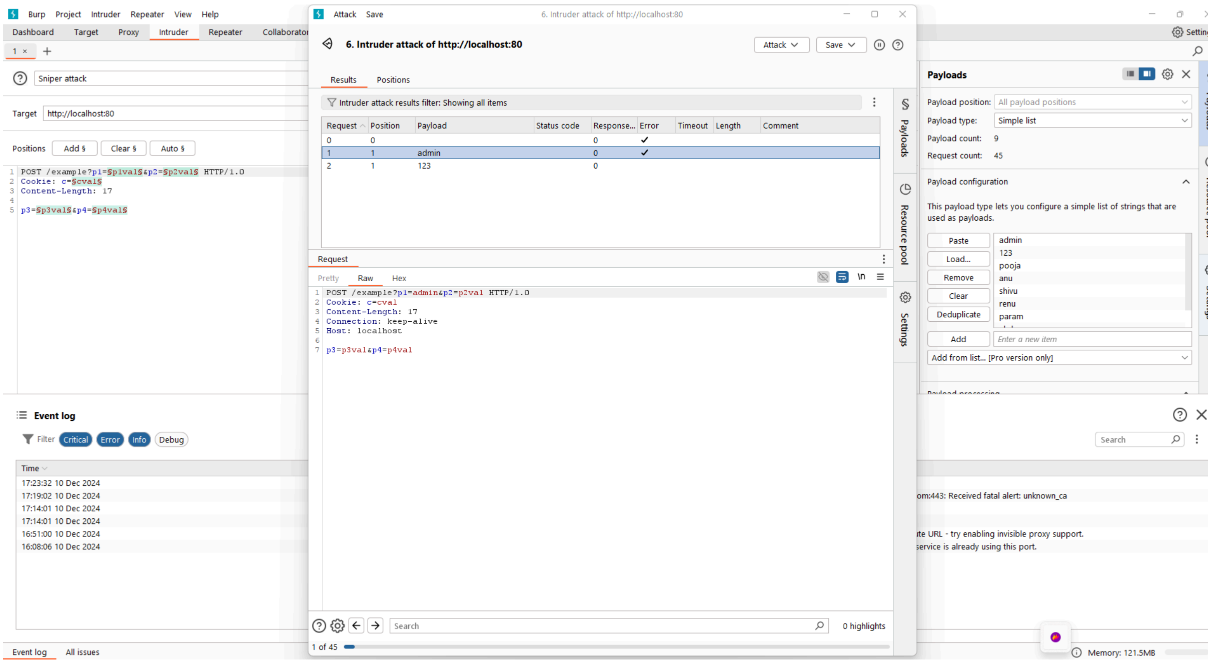
Request count: 45

Payload configuration

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

admin
123
pooja
anu
shivu
renu
param

Time	Type	Source	Message
17:22:40 10 Dec 2024	Error	Intruder	[322] Failed to connect to localhost
17:19:02 10 Dec 2024	Error	Proxy	[3] The client failed to negotiate a TLS connection to push.services.mozilla.com:443: Received fatal alert: unknown_ca
17:14:01 10 Dec 2024	Error	Proxy	Failed to connect to detectportal.firefox.com
17:14:01 10 Dec 2024	Error	Proxy	Failed to connect to detectportal.firefox.com:80
16:51:00 10 Dec 2024	Error	Proxy	Invalid client request received: First line of request did not contain an absolute URL - try enabling invisible proxy support.
16:08:06 10 Dec 2024	Error	Proxy	[3] Failed to start proxy service on 127.0.0.1:8080 - Check whether another service is already using this port.



6. Intruder attack of http://localhost:80

Intruder attack results filter: Showing all items

Request	Position	Payload	Status code	Response...	Error	Timeout	Length	Comment
0	0			0	✓			
1	1	admin		0	✓			
2	1	123		0				

```
POST /example?p1=admin&p2=p2val HTTP/1.0
Cookie: c=cval
Content-Length: 17
Connection: keep-alive
Host: localhost

p3=p3val&p4=p4val
```

```
cursor.execute("SELECT * FROM users WHERE username = ? AND
password = ?", (username, password))
```

○

2. **Use ORM (Object-Relational Mapping) Libraries**
   - ORM libraries like Django, Hibernate, or SQLAlchemy help abstract SQL code and reduce the risk of SQL injection by safely handling user inputs.
3. **Input Validation**
   - Validate all user inputs to ensure they match expected patterns (e.g., only alphanumeric characters for usernames). Reject inputs that

contain special characters (like `'`, `"`, `--`, etc.) unless they are necessary.

4. **Escaping User Input**
    - If dynamic SQL is absolutely necessary, ensure that user input is properly escaped to prevent it from altering the SQL query.
5. **Error Handling**
    - Avoid exposing database error messages to users. Instead, log them securely and display generic error messages to the user.
6. **Least Privilege Principle**
    - Limit the permissions of database accounts used by the application. Ensure that these accounts only have the necessary permissions (e.g., read-only access if no data modification is required).
7. **Web Application Firewalls (WAFs)**
    - WAFs can help detect and block SQL injection attacks in real-time by filtering out malicious requests.
8. **Regular Security Audits**
    - Regularly test your applications for SQL injection vulnerabilities through code reviews, penetration testing, or automated vulnerability scanners.

SQL injection is a powerful and dangerous attack vector, but it can be effectively mitigated using secure coding practices. By using parameterized queries, validating inputs, and following security best practices, developers can minimize the risk of SQL injection and protect sensitive data.