Using Burp to Detect SQL Injection Flaws

SQL injection vulnerabilities arise when user-controllable data is incorporated into database SQL queries in an unsafe manner. An attacker can supply craftec input to break out of the data context in which their input appears and interfere with the structure of the surrounding query.

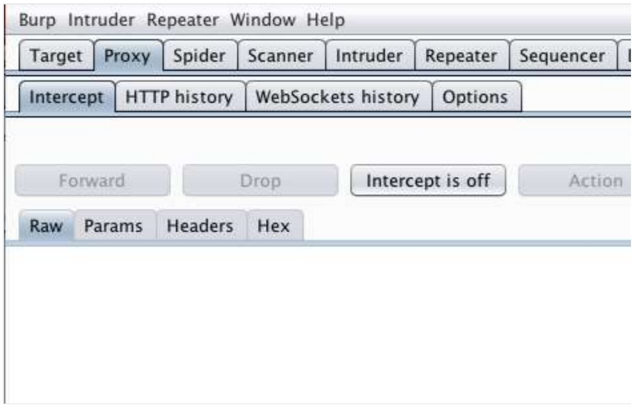
A wide range of damaging attacks can often be delivered via SQL injection, including reading or modifying critical application data, interfering with application logic, escalating privileges within the database and taking control of the database server.

In this example we will demonstrate how to detect SQL injection flaws using Burp Suite. This tutorial uses exercises from the "DVWA", "WebGoat" and "Mutillidae" training tools taken from OWASP's Broken Web Application Project. Find out how to download, install and use this project.

Scanning for SQL injection flaws

First, ensure that Burp is correctly configured with your browser.

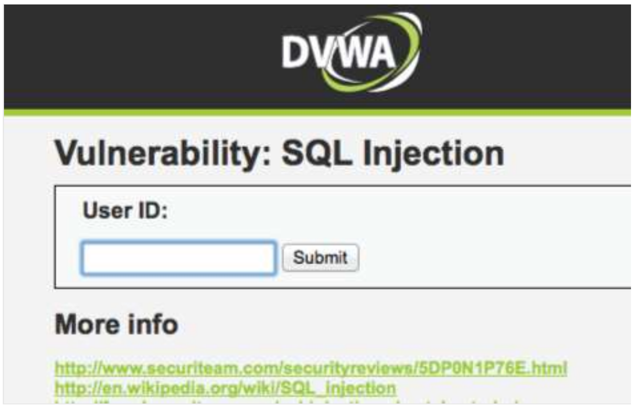
Ensure "Intercept is off" in the Proxy "Intercept" tab.



Visit the web page of the application that you are testing.

Return to Burp and ensure "Intercept is on" in the Proxy "Intercept" tab.

Now send a request to the server. In this example by clicking the "Submit" button.


DVWA
Vulnerability: SQL Injection
User ID:
Submit
More info
http://www.securiteam.com/securityreviews/5DP0N1P76E.html
http://en.wikipedia.org/wiki/SQL_injection

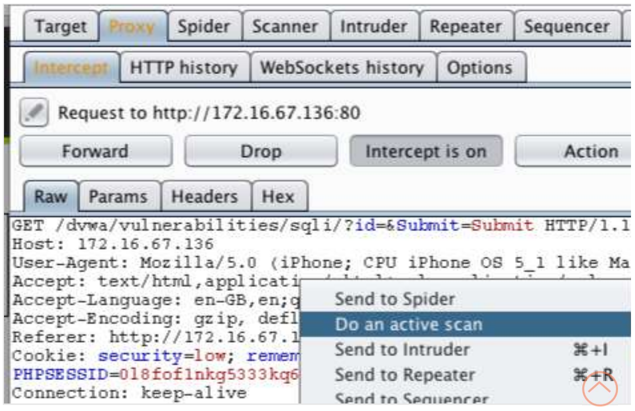
The request will be captured in the Proxy "Intercept" tab.

One way to test an application for SQL injection vulnerabilities is to send the request to Burp Scanner.

Right click anywhere on the request to bring up the context menu.

Click "Do an active scan".

Note: You can also send requests to the Scanner via the context menu in any location where HTTP requests are shown, such as the site map or Proxy history.



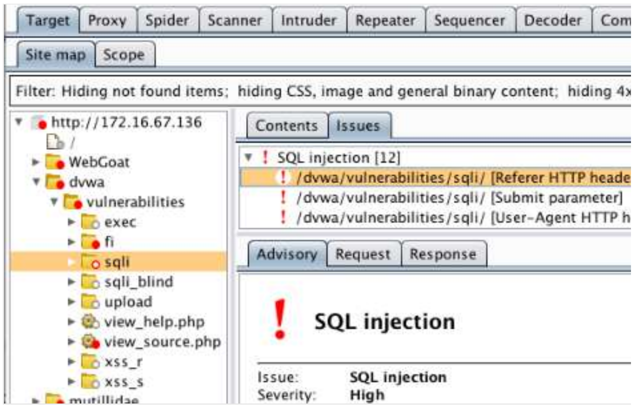
```
GET /dvwa/vulnerabilities/sqli/?id=&Submit=Submit HTTP/1.1
Host: 172.16.67.136
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 5_1 like Ma
Accept: text/html,applicati
Accept-Language: en-GB,en;q
Accept-Encoding: gzip, defl
Referer: http://172.16.67.1
Cookie: security=low; remem
PHPSESSID=018fof1nkg5333kq6
Connection: keep-alive
```

Once the scan is complete, go to the Target "Site map" tab.

In this example the Scanner has found a number of SQL injection issues.

Click on an individual issue to view the "Advisory" tab, which provides details about each specific vulnerability.

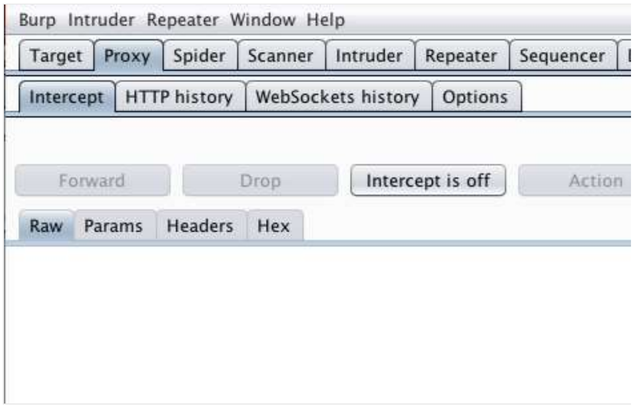
You can also view the requests and responses on the basis of which Burp has reported the issue.



Manual testing for SQL injection flaws

Alternatively, you can use Burp to manually test the application for injection vulnerabilities.

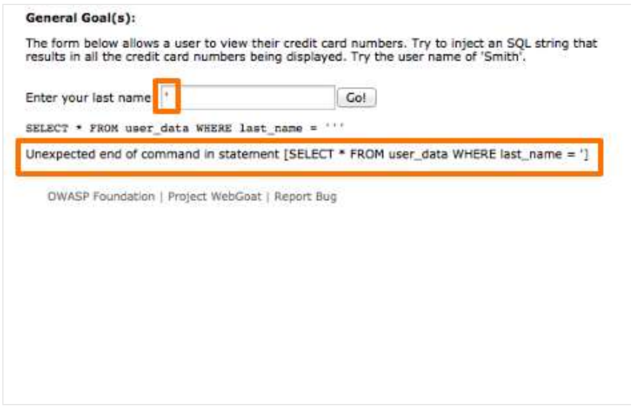
With intercept turned off in the Proxy "Intercept" tab, visit the web application you are testing in your browser.



Visit the web page you are testing.

You can often detect SQL injection by inputting certain characters in to the application's parameters.

In this example, submitting a ' (single quote) produces a custom error message.



General Goal(s):

The form below allows a user to view their credit card numbers. Try to inject an SQL string that results in all the credit card numbers being displayed. Try the user name of 'Smith'.

Enter your last name: ' Go!

```
SELECT * FROM user_data WHERE last_name = '''
```

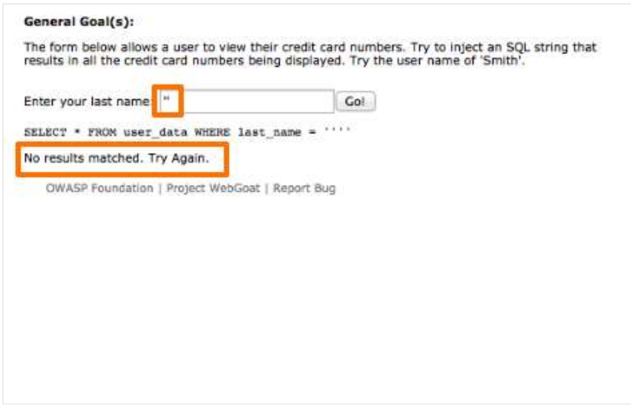
Unexpected end of command in statement [SELECT * FROM user_data WHERE last_name = ''']

OWASP Foundation | Project WebGoat | Report Bug

However, inputting '' (two single quotes) does not.

The reason for this difference is that SQL strings are contained within single quote delimiters. Submitting one quote breaks the representation of the string, and therefore the wider SQL statement. Two single quotes are an escape sequence representing a literal single quote. So submitting two quotes inside the string just modifies the value of the string and does not break the SQL statement,

If you are not already familiar with SQL and SQL Injection, you can learn more about it here and here.



General Goal(s):

The form below allows a user to view their credit card numbers. Try to inject an SQL string that results in all the credit card numbers being displayed. Try the user name of 'Smith'.

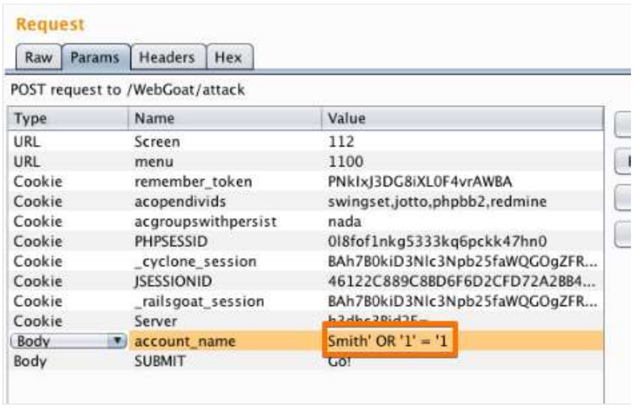
Enter your last name: '' Go!

```
SELECT * FROM user_data WHERE last_name = ''''
```

No results matched. Try Again.

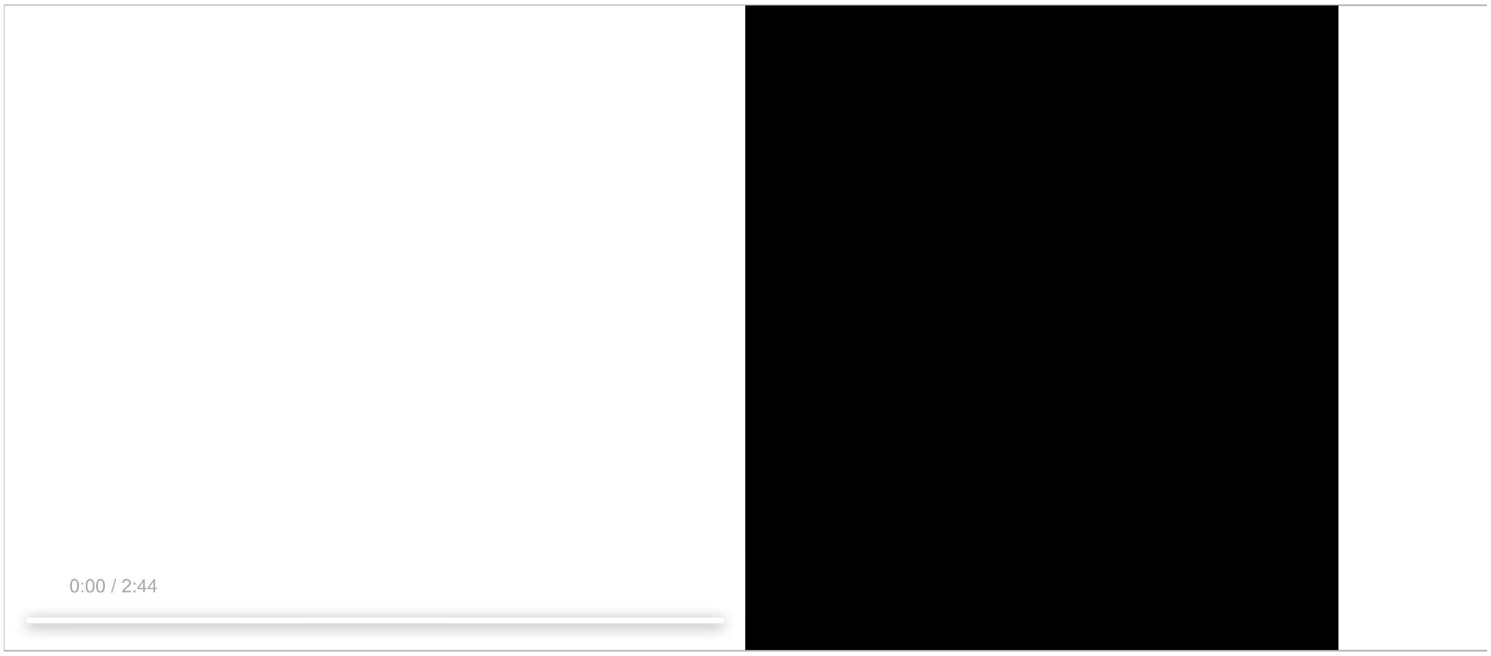
OWASP Foundation | Project WebGoat | Report Bug

Now that you have detected a potential SQL vulnerability you can use Burp to investigate the vulnerability further.



POST request to /WebGoat/attack

Type	Name	Value
URL	Screen	112
URL	menu	1100
Cookie	remember_token	PNkIxJ3DG8iXL0F4vrAWBA
Cookie	acopendivids	swingset,jotto,phpbb2,redmine
Cookie	acgroupswithpersist	nada
Cookie	PHPSESSID	0l8fof1nkg5333kq6pckk47hn0
Cookie	_cyclone_session	BAh7B0kiD3Nlc3Npb25faWQGOgZFR...
Cookie	JSESSIONID	46122C889C8BD6F6D2CFD72A2BB4...
Cookie	_railsgoat_session	BAh7B0kiD3Nlc3Npb25faWQGOgZFR...
Cookie	Server	[illegible]
Body	account_name	Smith' OR '1' = '1
Body	SUBMIT	Go!



Related articles:

- Getting started with Burp Proxy
- Getting started with Burp Scanner
- Using Burp to Investigate SQL Injection Flaws
- Using Burp to Exploit SQL Injection Vulnerabilities: The UNION Operator
- Using Burp to Detect SQL-specific Parameter Manipulation Flaws
- Using Burp to Detect Blind SQL Injection Bugs
- Using Burp to Exploit Bind SQL Injection Bugs

Burp Suite
Web vulnerability scanner
Burp Suite Editions
Release Notes

Vulnerabilities
Cross-site scripting (XSS)
SQL injection
Cross-site request forgery
XML external entity injection
Directory traversal
Server-side request forgery

Customers
Organizations
Testers
Developers

Company
About
PortSwigger News
Careers
Contact
Legal
Privacy Notice

Insights
Web Security Academy
Blog
Research
The Daily Swig

PortSwigger

Follow us