CS6542 - Graduate –Computer Network Defense – Spring 2017

Lab 15-1 page 391
Due 21st March 2017, 6:00 pm.
100 points

Policy on collaboration: All examinations, papers, and other graded work products and assignments are to be completed in conformance with The George Washington University Code of Academic Integrity. Each student is expected to write his or her own HW out independently; you may not copy one another's assignments, even in part. You may not collaborate with others on the test and final.
You are expected to cite all your sources in any written work that is not closed book: papers, books, web sites, discussions with others - faculty, friends, students. For example, if, in a group, one student has a major idea that leads to a solution to a HW problem, all other students in the group should cite this student.
You may not refer to solutions to previous years' problem sets, or ask for help students from previous years. Any violations will be treated as violations of the Code of Academic Integrity.

Please work on each lab and capture screenshot of tasks along with your words and analysis of each slide. PLEASE submit all Labs on Blackboard only. Name your files:

PLEASE submit all Project on Blackboard only

Late submission
Please note that, there is a %10 penalty for late submission until next project due date, and also there is NO grade for project submission after the next project due date.

1. Lab 15-1 Injecting the Hash

In this Lab, an MD5 hash is used to try to protect the users' passwords; however, there are same flaws in the implementation that can allow an attacker to perform SQL injection to bypass the authentication.

1.1. System Requirements:

1.1.1. Download VMware Image:

Download Backtrack 5 R3 VMware Image from (file name is BT5R3-GNOME-32-VM):

http://www.backtrack-linux.org/backtrack/backtrack-5-r3-released/

Or
http://www.wirelesshack.org/backtrack-5-download

BackTrack 5 R3 focuses on bug-fixes as well as the addition of over 60 new tools – several of which were released in BlackHat and Defcon 2012. A whole new tool category was populated – "Physical Exploitation", which now includes tools such as the Arduino IDE and libraries, as well as the Kautilya Teensy payload collection.

1.1.2. Install the BackTrack VMware:

You need to install BackTrack VMware image as a operating system like Windows 7, Kali, and the other operating systems that you have installed in your VMware software on your host machine.

1.1.3. Login to BackTrack VM

Login with to the BackTrack R3 VM with the following authentication information:

- Username: root
- Password: toor

1.1.4. Run GNOME

If you would like to run GUI interface please run the following command:

```
root@bt:~#startx
```

1.1.5. Configure the database

Log in to Mysql with password "toor" (or whatever password you set) and load the db_schema.sql file from this lab:

```
root@bt:~# mysql –u root –p
```

You can recover a MySQL database server password with the following five easy steps:

Step # 1 : Stop the MySQL service:

```
# /etc/init.d/mysql stop
```

Output:
Stopping MySQL database server: mysqld.

Step # 2: Start the MySQL server w/o password:

```
# mysqld_safe --skip-grant-tables &
```

Output:
[1] 5988
Starting mysqld daemon with databases from /var/lib/mysql
mysqld_safe[6025]: started

Step # 3: Connect to the MySQL server using the MySQL client:

```
# mysql –u root
```

Output:

Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 4.1.15-Debian_1-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

```
mysql>
```

Step # 4: Set a new MySQL root user password:

```
mysql> use mysql;
mysql> update user set password=PASSWORD("NEW-ROOT-
PASSWORD") where User='root';
mysql> flush privileges;
mysql> quit
```

Step # 5: Stop the MySQL server:

```
# /etc/init.d/mysql stop
```

Output:
Stopping MySQL database server: mysqld
STOPPING server from pid file /var/run/mysqld/mysqld.pid
mysqld_safe[6186]: ended

[1]+ Done mysqld_safe --skip-grant-tables
Start the MySQL server and test it:
/etc/init.d/mysql start

mysql -u root –p

```
mysql> source /path_to_schema/db_schema.sql
```

like for my backtrack is:
mysql> source /root/Lab_15_1/db_schema.sql

1.1.6. Set up the web environment:

- Go to your apache web root directory (usually /var/www/) and create a folder name "GH4".
- Unzip the Lab_15-1.zip file and copy it to the GH4 web folder.
- The final structure should be something like this:

```
root@bt:/var/www/GH4/Lab_15_1# ls -la
total 28
drwxr-xr-x 2 root root 4096 2014-11-12 10:43 .
drwxr-xr-x 3 root root 4096 2016-02-27 11:01 ..
-rwxr-xr-x 1 root root  683 2014-09-26 09:43 access.html
-rwxr-xr-x 1 root root  142 2014-09-26 09:43 brute.php
-rwxr-xr-x 1 root root  370 2014-09-26 09:43 db_schema.sql
-rwxr-xr-x 1 root root   63 2014-09-26 09:43 hash.php
```

```
-rwxr-xr-x 1 root root  658 2014-09-26 09:43 login.php
```

1.2. Tasks:

1) Navigate to your /var/www/GH4/Lab_1 and open login.php script

```
root@bt:/var/www/GH4/Lab_15_1# cat login.php
```

2) Run hash.php, which is located in the same web root folder

```
root@bt:/var/www/GH4/Lab_15_1# php -f hash.php
```

You can see that the output generated some nonprintable characters, a double quote, a colon, and so on. Therefore, we need to find a combination of chars that can generate MD5 raw output with our injection string embedded that's able to bypass the login check.
So, what combination of chars can we use for injection? Here, the first rule is that the string should be as small as possible so it can be generated by the MD5 raw output relatively quickly; otherwise, it could take hours or even months to find a match.
One of the smaller injection strings for bypassing authentication in MySQL is '=', which takes advantage of how type conversion during SQL expression evaluation works.

3) Login to mysql
4) Chose your database:

```
mysql> source /root/Lab_15_1/db_schema.sql
```

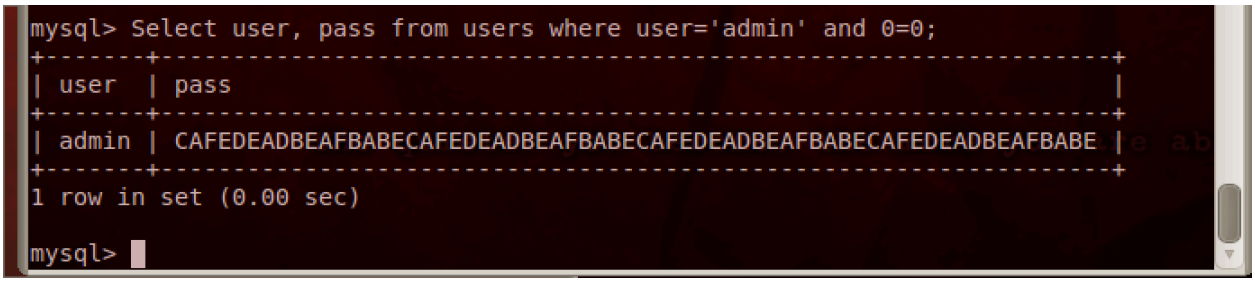
5) You'll be surprised at the end of this exercise when you see the weird results MySQL can produce when the type conversion feature is used. For this exercise, let's assume we know the username (admin) but do not know the password (of course). Therefore, if we execute the following query with the nonexistent password string1, we get no results:

```
mysql> Select user, pass from users where user='admin' and
pass='string1';
```

```
mysql> Select user, pass from users where user='admin' and pass='string1';
Empty set (0.01 sec)
```

NOTE MySQL does not have a proper Boolean type; instead, TRUE is equal to 1 and FALSE is equal to 0. What we need in order to bypass authentication is to force MySQL to return 1 instead of 0 when evaluating the password. The following query will suffice for our purposes because 0=0 is TRUE and therefore would return 1, thus giving us the admin password:

```
mysql> Select user, pass from users where user='admin' and
0=0;
```



6) So, how can we force MySQL to evaluate 0=0? Here is where type conversion comes into play. The following query will help us to achieve our requirement:

```
mysql> Select user, pass from users where user='admin' and
pass='string1'='string2';
```

Here, **string1** is a sequence of arbitrary characters (for example, X_1 X_2 …X_n) and **string2** is also a sequence of arbitrary characters (for example, Y_1 Y_2 …Y_n).

7) It is now time to brute-force MD5 raw output until it contains our injection string '='. We can do this by running brute.php, which is found in the repository. Check you can get the "esvk" as the password.

8) SQL Injection

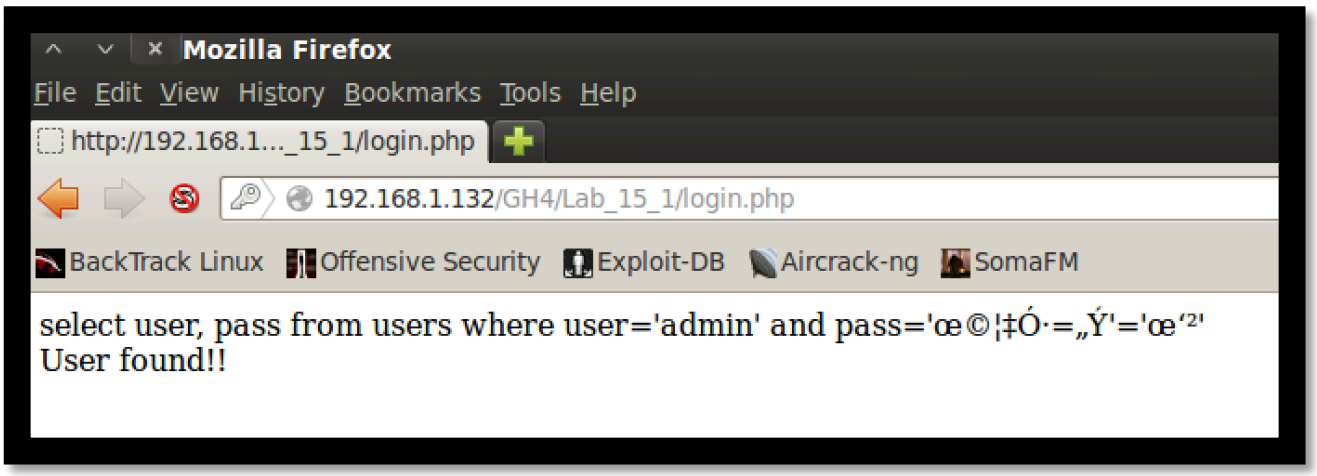
Now that the injection string "esvh" needed to bypass authentication has been identified, let's test it:

a) Ensure that your apache2 service is running

```
root@bt:~# /etc/init.d/apache2 start
```

b) Go to http://<your_ip>/GH4/Lab_15_1/access.html.

c) 2. Enter user admin and password esvh and then click Submit to send the data to login.php, as shown here:



Because the password is alphabetic, it won't be filtered by mysql_escape_

string()❷ in the code listing for login.php. The string “esvh” is converted into raw output and pasted into the SQL query, allowing us to bypass authentication, as shown here:
You can see the message “User found!!” here, which confirms we were able to bypass the authentication. The content of the raw output was intentionally printed out to show the full injection; string1 and string2 represent the left- and right-side portions of the query, respectively.

We can see in this Lab that the security controls were in place to prevent a SQL injection attack; however, the design of the MD5 hashing algorithm introduced a vulnerability to the authentication module. Actually, any of the 42 or so hashing algorithms supported by PHP (MD5, SHA256, crc32, and so on) can be exploited in the same way in a similar scenario.
NOTE The key point to keep in mind when hunting SQL injections is to analyze the input validation controls, trying to find a potential weakness.
Even when other, more secure technologies such as cryptography are used, if the implementation is wrong, input validation can be bypassed easily. One example is when implementing AES-128 with CBC (Cipher Block Chaining)1 without a ciphertext integrity check. From a developer’s point of view, make sure you use parameterized SQL queries (see the “For Further Reading” section) when creating queries based on user input.