Contemporary Computer Science - Security

Tom Robson - hzwr87

1 Passwords Stored in Plaintext

In the directory /var/www/html/, there is a file called query.php. This file is used to connect to the sql server and enable users to log in to the web store. When the mysql server is set up, the root username and password for the system are written in plaintext in the line of code below. As the user account given as vulnerability 0 has read access to this file, we can open this in a text editor such as nano and view the unencrypted root password:

```
mysql_connect(localhost,root,letmein) or die(mysql_error());
```

This vulnerability is also present in the outputDB.c file in the user's home directory, with the root password being written in plaintext in the following line:

```
char cmd[4096] = "mysqldump -pletmein --all-databases > ";
```

Even if we did not have access to the source file for this, we would still be able to use the 'strings' command in linux to view the strings within the compiled file, as shown below.

In order to mitigate this vulnerability in both of these files, these credentials must not be stored in plaintext. A way of ensuring this would be to store this information in a different file, such as a config file, which would be located elsewhere in the system and have stricter permissions applied to it. This should be done using a separate account to host and run the web service.

2 index.html is SQL Injectable

The index.html webpage contains a simple username and password form. This form is vulnerable to SQL injection, as we proved by using ' or 1=1– as both the username and password. This logs us in as the first user in the system. From here, we can experiment with more useful commands. If we know the username of any of the users of the system then we can log in as this person without needing their password and spend their credits. We can also use this to inject commands to give a user more balance, for example.

```
[user@dhcp-10-245-128-130 ~]$ strings outputDB
/lib/ld-linux.so.2
__gmon_start__
libc.so.6
_IO_stdin_used
setuid
memset
strcat
system
__libc_start_main
GLIBC_2.0
PTRh0
QVh4
[^_]
mysqldump -pletmein --all-databases >
```

Figure 1: Using the strings command

Another important aspect of this vulnerability is that it permits an attacker to access the file system of the host machine, and as the sql system runs as root, accessing the system through this method gives the attacker root permissions. Once we have found out the number of columns in the database through experimentation, we know where to inject commands to get data out. For example, this command shows us what user we are:

user = 1
password = 'UNION ALL SELECT 1,2,user()#
output: user 1 is logged in successfully. Your credit is root@localhost

This command shows us that we can extract the password file, only one of the passwords is included here:

user = 1
password = 'UNION ALL SELECT 1,2,LOAD_FILE('etc/passwd/')#
output: user 1 is logged in successfully. Your credit is
root:$1$8WSL8Sld$gcAwfXJ7.GzA/K131SGKV.:0:0:root:/root:/bin/bash.....

In order to mitigate this vulnerability, all user inputs must be sanitised using a whitelist before passing to SQL. Each query must be analysed to see what it will do before it is executed. We must also configure the database to ensure the queries are performed by a low privileged user so significant changes to the database cannot be made. Do not embed input into queries, use just as values not commands.

3 Full Database is available to all users

When signed in as user (the account given as vulnerability 0), we can use the mysql command to get access to the mysql shell, and therefore full database. Using the sequence of commands 'show databases;', 'use shop;', 'show tables;', 'select * from users;' shows us all the information stored about each user with an account for this online shop, including their usernames, passwords and balances. This would enable anyone with access to this machine to access all of the user accounts for the shop and spend their credits.

```
mysql> use shop;
Database changed
mysql> show tables;
+----------------+
| Tables_in_shop |
+----------------+
| customers      |
| goods          |
| users          |
+----------------+
3 rows in set (0.00 sec)

mysql> SELECT * FROM users;
+-------+----------+--------+
| user  | password | credit |
+-------+----------+--------+
| sally | password |   1000 |
| ian   | 12345    |    100 |
| simon | winner   |   -100 |
+-------+----------+--------+
3 rows in set (0.00 sec)
```

Figure 2:

In order to mitigate this vulnerability, these passwords must be stored in an encrypted form, using a hashing algorithm such as SHA-512.

Also, it is recommended that access to this database through mysql is restricted to the root user, or a different user account by that requires a password to access it.

4 Inappropriate permissions for .bashrc

When logged in as the user account that is vulnerability 0, we can access the .bashrc file in the /root/ directory. All users have read and write permissions for the files in this directory, so we can exploit this. In the .bashrc file, we can then add a malicious command, such as 'rm -rf *', to delete everything on the system. The next time the root user logs in or opens a shell, this command will be executed and all the data on the system will be deleted. We could also use this vulnerability to execute other commands that require root permissions, such as creating more users with root permissions, or setting the userid of existing users to 0.

In order to mitigate this vulnerability, the permissions on this file, and the root folder in general, must be changed so it is only accessible by root, using the command chmod 700. This means that only root can read, write and execute it, and other users can not do anything. There are other occurrences of inappropriate permissions present within this system, but this problem with the .bashrc file is the most severe vulnerability that has not been addressed elsewhere in this document.

5 Command Injection in outputDB

The executable outputDB, compiled from outputDB.c, requires a string input from the command line to create a dump of the current database. This filename is concatenated to the command to dump the database, which is stored along with the root password, and executed with user id zero to enable it to run privileged commands. However, because this can run any string as a command with root permissions, this is vulnerable to command injection. For example, by inputting the following command, we can make this program output the database to the file "dump.txt", but also give us a bash shell with root permissions:

```
./outputDB "dump.txt && bash"
```

To mitigate this vulnerability, the execve() command should be used instead of system(), as the system() command is unsafe. This is called using execve(cmd,args); Anything the right hand side of the &&, or a similar symbol, will be treated as a text argument not run as a command, which prevents the command injection from occurring.

6 Passwords are weak and easily crackable

In the /etc directory, the passwd file contains the hashed passwords for the root, toor and backup accounts, among other things. They are stored alongside the corresponding salt from their encryption. This means that these are vulnerable to dictionary attacks. These passwords are very weak, so can be broken very quickly using the John the Ripper password cracking tool, as they are very weak (password, letmein, 1234567), so will be broken very quickly by a dictionary attack.

To mitigate this vulnerability, the passwd file should not be readable by all users, it should be restricted to root. Also, this data should be stored in the shadow file rather than the passwd file in the same directory. This can be done using the command pwconv, which uses the passwd file to create a corresponding shadow file.

Also, change permissions on the backup password files (passwd- and passwd.OLD) so that they are only readable by root, using the command chmod 700.

These passwords are also encrypted using the MD5 system. This is not a good idea, as this system is old and is known to have been compromised and exploited many times in the past. A superior, alternative hash function should be used for encryption, such as SHA-512.

7 Stored XXS in Database

The database is vulnerable to stored Cross Site Scripting, or XSS. By either using SQL injection or through direct access to the mysql shell, we can use the following command to change the type of the credit column to allow commands to be placed there.

```
alter table users modify column credit varchar(500);
```

From here, we can use the following command to insert a script element in this column.

```
update users set credit = "<script>alert('Hacked!')</script>";
```

This is a simple alert command, however our ability to insert such commands makes it clear that we can use this vulnerability for more malicious means as desired.

This can be mitigated using a whitelist of commands that are allowed to be run on this database by certain users, so that we do not have the ability to change the type of the column, for example. Also, all commands must be checked and sanitised before they are run so that we can escape from <script> tags and similar.

8 Race Conditions in backupOnHour.bash

The file backupOnHour.bash creates a copy of the database and uses java to send this copy of the database to the backup account on this computer. After it has been sent, the file is deleted from the user system. Assuming we do not have write access to this bash file, we need to write another file that constantly checks the system to see if a file called tmp.db exists in this directory, and if it does make a copy of it before and save it or send it to another networked device. From here, the attacker would have access to a full copy of the database, including the user's credentials and information.

To mitigate this vulnerability, we could use a less predictable filename, such as a random filename, to thwart our searching file. Also, we could store the temporary file in another directory that will have permissions such that it will not be searchable by our file. This way, we will ensure that the attacker is not able to obtain a copy of our database, and ensure the data stored within is not accessible.