Security Assignment

Bradley J. Mackey

for 4th December 2017

1 Vulnerability

Path traversal vulnerability on the FileServer (running on port 8080).

1.1 Exploit

A malicious user is able to access any file on the entire server (including files not owned by `user` from `root` and the `backup` user) by traversing up and down to any file, by using url encoding in the link.

1.2 Mitigation

Improved **url input sanitation** on the FileServer would prevent url encoding that allows such traversal. This could be achieved by only allowing alpha-numeric characters to be entered in a url, followed by a dot and then a short file extension. If the url is detected not to match this pattern, the request can be rejected.

2 Vulnerability

SQL injection vulnerability on 'shop' web server (running on port 80).

2.1 Exploit

When a user signs up or logs in to the shop system, a maliciously crafted username or password can execute arbitrary SQL queries on the database (e.g. `';DROP DATABASE shop;`).

2.2 Mitigation

Improved input sanitation, by **escaping** user input using `real_escape_string` in PHP will prevent user data being interpreted as SQL commands.

3 Vulnerability

XSS vulnerability on 'shop' web server (running on port 80).

3.1 Exploit

If a user enters a JavaScript snippet for a username (or password) when signing up, the browser interprets this as code rather than data, meaning arbitrary code could execute in the browser of anyone that views this user's username. This could, for example, perform a malicious act such as cookie theft on another client's computer.

3.2 Mitigation

Similarly to the SQL injection vulnerability, this can be mitigated by **improved input sanitisation**, such as only allowing a short alpha-numeric username and only allowing certain characters within the password and using `real_escape_string` on the PHP server to prevent the data being interpreted as code.

4 Vulnerability

User permissions. User accounts on the system (such as `user`) have permission to view other user's home directories, including `/root`.

4.1 Exploit

Any user on the system has permission to navigate to any directory of any other user on the system without barriers. This includes root directories and files such as `/var/www/html/` containing `register.php`, which happens to be a file that contains the username and password for `root` (password `letmein`) (I won't mention the terrible password $^{(oops)}$) in plaintext, used for connecting to the `shop` database.

4.2 Mitigation

A **increased level of security** for individual user's home directories, i.e. users' should only be able to read/read-write their own directory. This will have an additional dampening effect on vulnerability #1, as path traversals *couldn't* access other user's files.

5 Vulnerability

Passwords stored in plaintext in SQL server for 'shop'.

5.1 Exploit

If a malicious user (including a malicious database administrator) gained access to the database, they are able to view a list of usernames and passwords for the shop in plaintext, inside the `users` table in the `shop` database. This would mean they could impersonate any user in the database and spend their credits.

5.2 Mitigation

Instead of storing plaintext passwords in the database, storing a **salted hash** for each user's password would mean that even if the database was compromised, user's accounts will not be. This is because the salted hash is only used to *check* if a user's password is correct by using a one-way hash function. The additional layer of salt (that should be a randomised string for each user) will ensure that even if 2 user's passwords are the same, they will not have the same hash and also means using password cracking facilities such as rainbow tables are much less feasible due to this additional layer of randomness. Of course, user's should also use a strong password (which it appears the current user's of the shop do not have) in order to avoid having their password easily brute-forced or dictionary brute-forced.

6 Vulnerability

6.1 Exploit

6.2 Mitigation