ITSC 302 – Web Application Security

Assignment 2

Submitted by Coleton Sanheim

Invalid locking/unlocking of user accounts

Code used:

'; UPDATE users SET locked = 1 WHERE username = "bob";#

This exploit allowed me to change the locked setting of user “bob” to 1, locking him from the web application.



Denial of Service (DOS)

Code used:

'; DROP TABLE users;#

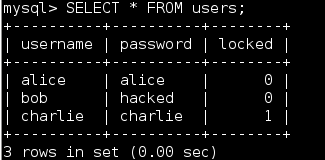
This exploit deleted the table “users” from the sql server and so any user wanting to log in will be denied access.



Code used:

'; UPDATE users SET password = "hacked" WHERE username = "bob";#

This exploit is similar to the unlocking/locking of user accounts but this time I changed the password of the user “bob”. This exploit effectively is a denial-of-service attack against one specific user to prevent them access to the application.



Invalid Access

Code used:

'; INSERT INTO users (username, password, locked) VALUES ("hacker", "hpassword", 0);#

This exploit has allowed me to create a new user account and set a password for it, allowing me access to the application as if I was an authenticated user.



Data Theft

Code used:

‘; SELECT \* FROM users INTO outfile “/root”;#

This exploit allows me to dump the contents of the user table, (usernames passwords and locked status) into a text file that I can then access and read. However, I had some issues getting this one to work, so I am unable to showcase a screenshot of this exploit working, sorry.

Security Recommendations

All of these exploits use a similar base syntax, they start with a ‘; which ends the previous sql command string as a command, with the semicolon. Then at the end of the injection there is a ;# the semicolon ends the command I have entered and the hash comments out the rest of the code in the backend.

The main recommendation I would give to harden this application is to filter inputs. Instead of passing strings with possible sql injections in them directly to the sql server, first parse the strings and either replace any common sql characters and commands (ex. ‘ character) with a replacement character, or simply not allow certain characters like that to be entered in the first place. Also, I would recommend a character limit in the input fields. I would also recommend that the database and tables use less common names than “users” and “password” so it is more difficult for an attacker to guess. Finally, it would be a good idea that any sensitive information such as passwords stored in this database be encrypted so even if an attacker gains access then the information is useless to them.