CIS 4930 - Software Security - Spring 2020

Mini-Project 1 Report -- Salt/Pepper Hash and Web Exploit Prevention

Members:

Stalim Rivero

Amanda Gonzalez

Rafael Gonzalez

Tyler Allan

Our mini-project represents a login/registration scheme for an online banking system that exemplifies a basic user-system interaction. The vulnerable version will be subject to 4 types of attacks, namely SQL injection, IDOR/URL manipulation, CSRF/Session attacks, and XSS which all violate several security policies of the website. On our hardened version, these vulnerabilities will be mitigated. The goal to the hacker to initiate the exploit is to find ways to acquire account information in order to have access to the account services. All 4 of the exploits that we will cover in the vulnerable version, and mitigate in the hardened version, will be ways of acquiring this information.

Types of Vulnerabilities

Based on the STRIDE Threat types and what this diagram shows, some potential vulnerabilities come to light. The hacker can obtain account information and pose as a legitimate user (Spoofing). After gaining this information, he/she can then modify that account information, say mailing address, phone number, or apply for any of the provided bank services (Tampering). The attacker can also, in order to criminalize the account, deposit fraudulent checks and raise suspicion about the legitimacy of the account and deny any involvement (Repudiation). After gaining information about the account, it can easily be disclosed to any other unauthorized user (Information Disclosure). After changing the account information, legitimate users could potentially be denied their service when the administrator attempts to verify sensitive information in order to process a request (Denial of Service).

Exploits and Mitigations

URL manipulation is a potential exploit in out vulnerable version, given that in the URL you can see and id number, which if you change it will redirect you to another account's information and their secret word. As mitigation, instead of sending the user_id through the url, we created a global variable within the application that stores the username, password, and user_id of the logged in user only.

On step further, for **XSS,** after the URL manipulation you can input <script>alert(document.cookie)</script> and the page will respond with a cookie containing username and password for the current logged user. As mitigation, we took out the "| safe" keyword after the input. It's a feature that comes built in with Django, once you take it off, it takes in any input as string, so it does recognize things such as <script>

For **CSRF**, the attacker can hide POST request in hidden iframe that is processed because he has user authentication after logging in as a user, money will be transferred from currently logged in user account to hacker account and there will be no visual confirmation of this. This is mitigated by using a key that is submitted with the form to verify it's from the correct place. A hidden field was created with the form on the create.html page. That hidden field has a variable with the key name hidden directly on the page. The blog.py function checks if the key is present so the attack no longer works from another page.

Finally, as a **SQL** Injection, first you register a new user, switch to admin, send user money, switch to user, send someone money, go to search transactions page on user, do a normal search. Only the user's transactions will be shown then Input ' OR '1'='1'; -- and all transactions shown. To mitigate the attack, we used functions in flask that escape the sequence before sending it to the database.

Password hashing & encrypting, Salting and Salt + peppering passwords

The most important aspect of a user account system is how user passwords are protected. User account databases are hacked frequently, so you absolutely must do something to protect your users' passwords if your website is ever breached. Securing user information begins with a proper understanding of security controls and the protection of user passwords using modern hashing and encrypting algorithms. Though both methods provide a layer of security for the majority of situations, hashing has one feature that stands out, which is that is not a reversible operation, once the value is hashed it cannot go back to its original state (password). On the other hand, symmetric encryption provides a security vulnerability, the value can be reversed with access to the key that was used to encrypt, decrypting the value and obtaining the password.

In cryptography, a salt is random data that is used as an additional input to a one-way function that hashes data, a password or passphrase. It is added to the hashing process to force their uniqueness, increase their complexity without increasing user requirements, and to mitigate password attack. Pepper works in a similar way to salt in that it is data that is also appended to data prior to being hashed. However, the main difference is that while salt is stored with the hashed value, the pepper value is hidden away from the hashed value. This adds the additional benefit that the pepper is not known to the attacker.

Because hashes are a one-way function, the rotation of the pepper key creates an additional problem. A password hashed with a pepper relies on that original pepper in order to be validated. This precludes the rotation of the pepper key, a big security flaw.