Team thirstyboys Report

By: Gerrad Hardy, Dakota Rennemann, Logan Smith, Cameron White

Initiating Exploits

SQLi vulnerability in Login Page -- First, enter the email address of a known account, then enter `'OR'1'='1` for the password field. We're now in the user's dashboard page.
CSRF vulnerability in Transfer Page -- There is a sensitive data exposure in the transfer page, thus an attacker can send a forged GET request to `/transfer.php?email=arbitraryEmail@provider.com` in order to see all of a user's transactions
IDOR vulnerability in Transfer Page -- First, use burp to as MITM to grab the POST request from a legitimate transfer. Next, modify the parameters to have the sender be the account we'd like to take from and receiver be our account (amount can be changed at this point as well). Finally, we can send this modified request to the server. We've now taken money from another account.
XSS vulnerability in Transfer Page -- For this attack, you can initiate a transfer with the victim for an arbitrary amount. The key here is entering `'<SCRIPT>alert('XSS')</SCRIPT>` as a proof of concept into the memo.

Types of Vulnerabilities and Mitigations

SQLi vulnerability in Login Page -- Improper input sanitation allows us to bypass the password field with Boolean injection. Using prepared statements to escape the user input, we are able to prevent SQLi.
CSRF vulnerability in Transfer Page -- Poor session management allows us to send a forged request from a victim's account to view their transactions. By adding proper cookie management and hiding server operations from the client, we can help prevent stolen sessions and request forgeries.
IDOR vulnerability in Transfer Page -- Sending parameters in requests without some kind of validation server-side allows us to send money from another account to our own. Adding proper privilege checks and making IDs non-enumerable helps stop IDOR attacks from both finding fuzzable values and exploiting them.
XSS vulnerability in Transfer Page -- Improper escaping of user input makes the memo field of a transaction field vulnerable to us performing a stored xss attack that is triggered by the victim opening their dashboard. Again, using prepared statements, proper escaping, and input parsing, we can stop user input from being anything but read-only.

Password Hashing Vs. Encrypting

Both hashing and encryption to convert plaintext into an unrecognizable form to prevent unauthorized parties from easily reading the original text. The main difference between the two is encryption can be decrypted back to plain text while hashed values can not be unhashed.

Encryption uses an algorithm and a key to create a ciphertext (encrypt(plaintext, key) -> ciphertext). The ciphertext is reversible to plaintext by someone who knows the key and the algorithm used by reversing the algorithm (decrypt(ciphertext, key) -> plaintext).

Hashing, on the other hand, is one way as the hashes produced cannot be reverted to plaintext. Hashing doesn't use a key (hashingFunction(plaintext) -> hash) and someone can validate the hash by entering the

same input into the hashing function and comparing the output. Password hashing typically involves salting/peppering in order to make the hash harder to attack (see next section).

Salting Vs. Salt-and-Pepper Hashing (What it Enables)

Salting is a hash security measure when a set of random, unique data that is concatenated to the password before being hashed. Since it doesn't need to be memorized the size of it and therefore the hash table can be really large without any extra burden on the user. Salts are unique as well so if you put the same password through the hashing algorithm multiple times you will get a different hashed value because each instance salt was different.

Peppering is another hash security measure that can be used in combination with salting. It is akin to an encryption key in that it is required to check if the hashed value matched the input value. The main difference between salting and peppering is the pepper value is kept away from the password database in the case said database is compromised. Without knowing the pepper none of the hashes can be cracked. A concern that arises with peppers though is since they are stored away from the hashed value the pepper has to be passed around for validation purposes.

Salt and Peppering a hash enables us to have the hashing benefits of a one-way algorithm that cannot be reversed along with some defense from a rainbow table attack against our hashes.