# CS392 – Endsem Test

Name: **M.Maheeth Reddy**

Roll: **1801CS31**

**A1**:

My answer is "No". SQL Injection Attack is still possible because code and data are getting mixed.

Consider there is a user with user id as uid1

If an attacker is targeting this particular account, he can enter the following string in the user id field: uid1',256) OR 1=1 #

And he can enter any random text in the password field.

So in the backend, the mysql query would effectively become:

$sql = "SELECT * FROM employee

WHERE eid=SHA2('uid1', 256) OR 1=1 #, 256) and password=SHA2($passwd, 256)";

In this way we can bypass the requirement of password and gain access to the whole database.

**A2**:

In my opinion, P1 and P2 both don't have race condition vulnerability. However, P2 is vulnerable as password file can be opened if the program is a set-uid program owned by root user.

P1 won't have Race Condition vulnerability because "/etc/password" is a protected file so normal users does not have any access. Then !access("/etc/passwd", W_OK) is evaluated to false, and the program is never going to pass this if condition and open /tmp/X file.

For P2, if the user has access to /tmp/X, !access("/tmp/X", W_OK) is true and this condition is passed. In the next step, there is a direct attempt to open /etc/passwd. Only If the program is root owned set uid then file can be opened as the euid in this case will be 0 (root). If the program is not root owned set-uid, it will fail

**A3**:

Laksha is wrong. HTTPS can do nothing by itself to prevent CSRF attack. HTTPS only encrypts the traffic between the communicating parties i.e., client and server computers.

With respect to given site http://www.iitp.ac.in/samrat/CS392 SSD,

Site (a) https://www.iitp.ac.in/samrat/CS392_SSD and

Site (b) http://www.iitp.ac.in/samrat/CS392_SSD/submission/

are different sites according to the same origin policy. Origin is basically a combination of URI scheme, host name, and port number. For two sites to have same origin, these three fields must match. Sites (a) and (b) are of different scheme and with respect to the given site, an attempt to access (a) from (b) would fail. Also, the given site and (a) are different since the scheme is not same. But the given site and (b) are the same sites, because the scheme is same (http).

**A4**:

Given,

Length of password = 8 characters

Also, each character can take 64 different values

(a)

Since each character can take 8 different values and there are 8 characters in one password, 64^8 = 2^48 different passwords are possible

(b)

Let us consider the average half the dictionary, before the matching password is found, would take Tamal password tests.

In half of the cases, Tamal would have succeeded cracking the password just by using a dictionary. So, this requires 2^30/2 = 2^29 trials on average.

In the remaining half of the cases, Tamal would have to use bruteforce which requires 64^8/2 = 2^47 trials on average.

So, the total number of tests is (2^29 + 2^47)/2 = 2^46 approximately.

The amount of time it would take Tamal's program to crack the password, would be 2^46 / 2^6 = 2^40 seconds.

(c) When we use salt, it will increase the difficulty of cracking the password in the given situation. Because when salt is not used, we can precompute the hashes of all passwords in the dictionary file. But when salt is used, the hashes of each password must be recomputed for every trial. But the time taken would be of the same order as in case (b).

**A5**:

The given code snippet "WILL NOT BE HELPFUL" in defending against clickjacking attack.

getDom() function returns the domain of the given URL. The given code snippet is only checking whether any of the strings provided is a substring of the domain. We can infer that any domain that contains the substrings usbank or usbnet will be allowed to frame the page.

For example, if we assume a website www.qwertyusbank.com, this website can be used to frame the website of USBank because it contains the substring usbank in the domain.

**A6**:

Among the attacks we have learnt in this CS392 course, the only attack which can spread by itself is cross-site scripting attack (XSS).

Self Propogation XSS worm is where the victim's account is not only effected, but it would further act as a medium of spreading the attack elsewhere.

Such an attack happens when a self propagating script is copied to the victims' profiles whenever the victim opens the attacker's profile. This would cause the code to spread to all those users who shall view the effected accounts. And thus self propagation happens.

Such an attack could be done in two ways:

DOM approach: JavaScript code can get a copy of itself directly from DOM via DOM APIs

Link approach: JavaScript code can be included in a web page via a link using the src attribute of the script tag.

Now, this is in essence an XSS attack, where the ability to copy code into other profiles is present as an extra feature. And thus, the root of the problem is still mixing code and data.

Thus the obvious countermeasure is to seperate them. The following two approaches coould be used

Filter Approach: We remove the code from users input by identifying the <script> tags or other code embedding methods. We could use open source code parsers to filter out code

(jsoup)

Encoding Approach: We could encode the characters with special functions using indicators.

For eg:

Converts <script> alert('XSS') </script> to

"&lt;script&gt;alert('XSS')&lt;/script&gt;"

This would just display the code and not execute this