CS3226

WEB PROGRAMMING AND APPLICATIONS

LECTURE 09 - SECURITY

DR STEVEN HALIM

STEVENHALIM@GMAIL.COM

Are you a (white) hacker?
Try this https://xss-game.appspot.com/

OUTLINE

WHY DO WE DISCUSS SECURITY? (1)

Analogy: Writing a web application and hosting it in a 24/7 publicly accessible web server is like opening the doors of your home for *both* visitors and intruders (side story)

Visitors are legitimate users
You want them to visit and use your web application

Intruders are users with intention of harming your web application and/or its database
You do not want them to attack your web application

WHY DO WE DISCUSS SECURITY? (2)

Web security is basically a collection of growing techniques to let visitors in and keep intruders out

You cannot say that your web application is 100% secure and then becomes complacent

This is an ongoing battle of good and evil

A1 AND A2

Two most important concepts in web application security are Authentication (AuthN, A1) and Authorization (AuthZ, A2)

AUTHENTICATION (AUTHN, A1)

Authentication is simply a way to tell a web application about who you are

Usually by means of (strong) password (side story: a simple brute force dictionary attack experiment in a recent CS3233 contest)

In more sensitive web applications (e.g. Internet banking), possibly we also use additional token for 2FA*

PS: We are not yet in the era where biometric* authentication system is useful yet

AUTHORIZATION (AUTHZ, A2)

Appropriate authorization level is granted by the web application **after** successful authentication

Example authorization levels @ VisuAlgo Online Quiz:

1. Admin (can do everything)
2. Lecturer (can set up, start, analyze, close online test)
3. Student (can join online test only)

REVIEW: HTTP BASIC A1 (1)

Pros:

1. It is simple to use (see Lab1)
2. It 'works*' for our labs as it is harder for your peers to steal your (client-side) source code before the lab is due*

REVIEW: HTTP BASIC A1 (2)

Cons:

1. It is very inflexible, 0/1 authorization levels*
   0 = no access to those who do not know the password
   1 = full access for those who know the password
2. We are actually on shared Virtual Machine...
   There is a ______ (to be mentioned verbally)
3. Poor user interface (uses browser default)
4. User has to be notified of their password via other means
5. User cannot (or can but too troublesome to) ask the admin (web application owner) to change their password

BETTER A1+A2 SYSTEM

Use PHP server-side scripting + MySQL database to have a better system

Here are the key ingredients:

1. A user table in our (supposedly more secure*) database, typically with the following schema: user_id, hashed* password (use `crypt` function in PHP*), and authorization level, something like this

2. A sign in page `login.php` with typically two input boxes, a user_id and a password
   (with masked input like this)

   This script performs check against our database if such user_id exists and his/her hashed password (using the same one-way hashing algorithm, note about 'salt' if using PHP crypt function) matches the one stored in our database, typically we return a new session ID and let this user_id access resources according to his/her authorization level; If the sign in fails, we redirect the user back to login page

3. A sign up page `signup.php` that allows initially **total stranger** to use your web application by specifying their preferred user_id and password (entered twice*, preferably with password strength indicator), and optional email field for password reset functionality (no 4 below)

4. For convenience: A forget password page `forgetpassword.php` with at least one input: either user_id or useremail and we will send email with newly generated password to user's registered email address

   Be careful of not opening security loophole here

5. For convenience: A change password page `changepassword.php` with typically four input: user_id, old password, new password (entered twice, preferably with password strength indicator)

   Similarly, be careful of not opening security loophole here

6. For extra piece of mind: A logout page `logout.php` that basically destroys all session information, cookies (if any), and basically tries to 'forget' who you are (that is, the reverse of A1)

   Side discussion: Do we always need to logout from a web application?

BASIC WEB SECURITY PRINCIPLES

This is my current compilation so far and should grow over time

1. *NEVER TRUST THE USER*

   Sanitize everything that comes from user (e.g. escape all quotes from string that will form (part of) SQL queries to minimize SQL injection), never trust client-side validation (it may have been bypassed by attackers who know his/her stuffs — example with Lab5, etc), whitelist* which users are allowed access and what those users can do — example with recursion @ VisuAlgo...

2. *DO NOT HARM OTHER (GOOD) USERS*

   In case bad stuffs ever enter your web application database, do not let it affects your other (good) users

3. Web Browser (Client-side) Same Origin* Policy

   The main reason why popular web browsers typically employ Same Origin Policy is for privacy issue (you don't want a random website to be able to check your personal details while you happens to have your Facebook account open in another browser tab) and to minimize Cross (Xross) Site Scripting (XSS, not CSS) attacks
   For now, I assume that you will* put your server-side (PHP) scripts in the same origin as your client-side (JS) scripts

4. Setup Correct File/Directory Permissions

Summary of permissions for "others" in UNIX file system:

Directories	Execute only, no read access
HTML/CSS/JS (client-side) files	Read only, so that the client web browser can read them
PHP/SQL files	Read only, sensitive credentials outside public_html

5. Use HTTP POST instead of HTTP GET to avoid exposing sensitive user data in the URL (minimize shoulder surfing attack)

6. Use HTTPS (HTTP over SSL) so that our HTTP requests/responses are encrypted

   But you need to setup SSL (Secure Socket Layer) certificate which is not cheap (and I have not try)...

7. Use (re)-CAPTCHA to differentiate human versus machine

   If necessary, you can use application-specific CAPTCHA, like this (but not maintained though...)

NOTES (1)

By knowing more about (web) security, you will realize that your current web application is actually not that secure... Try reading this OWASP (Open Web Application Security Project) Top 10 list

The more popular/important your web application is, the greater the need of making it as secure as possible...

NOTES (2)

This lecture is short and should be expanded with updated details for future AYs

If this is not yet your final semester and you are "weak" at this topic, I recommend you to take security modules in SoC CS5331 - Web Security is a very relevant level 5 module taught by my colleague Dr Prateek Saxena

Finally, I will close this topic by giving you this link: https://xss-game.appspot.com/; At least you should be able to clear the first level by yourself (it has been discussed in this lecture)

THE END

Try https://xss-game.appspot.com/ if you have not done so