CSE 484 / M584 Lab 3:
Web Application Exploitation

1 Structure

Due Dates

- **All components**: Wednesday May 14th, 11:59pm

Handin

- Complete components weblab website: https://cse484.cs.washington.edu/lab2
- Writeups (individual) on Gradescope, see Deliverables.

Groups

- **Exploits on website:** Partners or individual
- **Writeups:** Individual

Grading

All writeups MUST be done individually: do not collaborate on the writeups.

Points: 34 + 5 EC

- 20 points total for exploits (XSS: 1–4, SQL: 1–2, CSRF)
  - 2 points XSS 1,2, SQL 1
  - 3 points XSS 3,4, SQL 2
  - 5 points CSRF
- 2 points per writeup (14 total).
- 1 point per EC exploit/writeup (XSS: 5–8, SQL: 3) (5 total)

2 Formatting Notes for this Handout

- Programs, files, or tools are stylized like `this`.
- Commands for programs are stylized like `this`.
- Code, assembly, and strings are stylized like `this`.

3 Before you start

- Sign up for an account here: https://forms.gle/LWV8EYgh4FusZ29v8 (You MUST use your @cs.washington.edu Google account, not your UW Google account.)
- **Your group name will be both net id's (or one, if individual) combined together alphabetically. The password is what you specify in the form.**
  - If your netids are: "btoren" and "amianaai", your group name will be: "amianaaibtoren". Note the alphabetical ordering.
- If you are working in a pair, only submit the form from one of your accounts (with both netids listed.)
- Ensure to not submit this form twice with the same username if you decide to work with a partner later / need a change to the form, please reach out to us
- Accounts take up to ten minutes to be created.
- The FAQ and external links will be necessary to read to complete the lab.

4 The Server

https://cse484.cs.washington.edu/lab2 hosts the web application you'll be attacking.

There are not secret problems or extra components that are not listed. The only problems are the ones explicitly shown on the page once logged in.

5 Overview

- The goal of this assignment is to familiarize yourself with basic web application vulnerabilities.
- This includes: XSS, SQL injection, and CSRF.
- A secondary goal is to expose you explicitly to how trust between clients and servers on the web works.
- There are 4 XSS, 2 SQL, and 1 CSRF required problems, and several extra credit.
- There are required short writeups for each problem.

The goal of this lab is to gain hands-on experience with penetration testing of web applications.

For this part of the lab, you are presented with three different scenarios. Each scenario asks you to perform a task you would not otherwise be able to complete as a regular, benign user. You'll have to figure out what vulnerability exists in each challenge, apply what you've learned in class, and craft a special payload to achieve your goal.

5.1 Deliverables

Required:

- Have the weblab show you completed: XSS 1–4, SQL 1–2, CSRF
- Writeups on Gradescope containing:
  - Your payload (what your input was to the application to exploit it)

  - A short description for why it works (~1–4 sentences)
  - Any files you used (php scripts, extra files you uploaded, etc.)

Extra Credit:

- The same as above, but for problems: XSS 5–8, SQL 3

6 (Optional, fun only) Back Story

Scenario #1: Pikachu, Meowth, and Cookies

Everyone likes cookies, and Pikachu and Meowth are no exception. As Team Rocket's 4294967296th evil plan, Meowth is going to purchase all the cookies within Pikachu's reach so Pikachu would eventually surrender and give himself in, but of course Team Rocket cannot win.

Having eavesdropped on their conversation, you learned that Team Rocket keeps the cookies they bought in 8 different safes and stores the combinations to each of the safes in 8 different cookies Meowth carries with him. You also learned that Meowth set up a website to facilitate communications with his fans (if any). With these in mind, you want to find a way to get Pikachu some cookies back before he faints from a lack of cookies... but how?

Scenario #2: Jailbreak

You have been put in jail due to a wrongful conviction. You have no one to depend on, and the only way you can eat that University Teriyaki again is to jailbreak. Physical locks are for the weak; as a former Jedi, you can easily break them with the Force. What bothers you are the digital locks that are connected to a central database. But then, some material you've learned from CSE 484 flashes before you...

Scenario #3: Hack your 4.0

Having joined CSE 484, you realized a sad truth: there's no way you can get a 4.0 for the class. You've learned that the seemingly nice and friendly CSE 484 TA has no mercy and routinely fails students as a hobby, and that the only way to get a good grade is to surreptitiously hack into the gradebook and change your own grade.

However, your CSE 484 TA is like no other; there's no way their website can be vulnerable to any attacks, or so they say...

7 Getting Started

Now that you have read the motivational backstory, let's get started!

Helpful tools and setup:

Browser

During the course of this lab, we recommend that you use Firefox. The server uses Firefox and your exploit might exhibit different behavior with another browser like Chrome (i.e., your code might work on Chrome but not on Firefox).

Also, disable extensions that may change how your browser handles cookies like ad blockers. If you use Firefox as your daily browser, and don't want to disable your extensions, you can install Firefox Developer Edition to have a separate, "clean" installation.

Note that protection tools that are built into some browsers may interfere with this assignment. You might try turning off tools like Chrome's XSSAuditor and IE's XSS Filter.

Setting up your webpage

When doing XSS attacks, you will need to exfiltrate the cookie from the victim's browser to a location where you can retrieve the cookie. One easy way to do this is to set up a webpage that takes `GET` requests with parameters. The goal is to have a page that when you navigate to:
`homes.cs.washington.edu/~<username>/cookieEater.php?cookie=secretCookieValue`
your page will record `secretCookieValue` so you can read it later. We will go through the steps to help you get set up.

- Host your webpage at homes.cs.washington.edu, instructions at https://internal.cs.washington.edu/lab/web/homepages/.
- Once you have figured out where to host your page, you will need to write some PHP (or any other server side programming language) that will retrieve `GET` variables.
- Some useful PHP links:
  - PHP GET Variables
  - PHP Tutorial
  - Writing to a file in PHP
- You will need to mark the correct permissions on your PHP Script, as well as on the file it logs to.
- We provide a very basic starting script below, but encourage you to modify and improve it for you use. (E.g. Adding logging a timestamp is very useful!)
- Note: this part of the lab is not intended to be hard. If you are struggling, please reach out so we can help you move past it to the interesting stuff.

- Now, you can try out your cookie receiver by using your browser to navigate to `homes.cs.washington.edu/~<username>/cookieEater.php?cookie=secretCookieValue`, assuming your php file is named `cookieEater.php`. If your php script records the value `secretCookieValue` correctly, you can get started!

Permissions

Your script must be world-readable (so that the Apache webserver process can read it). You definitely don't want it to be writeable by other students though! Your log file that you write to must be world-writeable, but you don't want it to be readable by anyone but you! (Otherwise other students can steal your solutions!)

If you get confused here, I highly recommend reading the manual page for `chmod`, and the 391 lecture notes.

We recommend `chmod u=rw,g=,o=r` for your php script (world-readable but not world-writable, group non-readable or writable) and `chmod u=rw,g=,o=w` for the log file to which you're writing cookies (world-writable but not world-readable, group non-readable or writeable.)

Starter PHP Script We are supplying a sample starter script. You will need to rename it (to a `php` file so Apache knows to interpret it as PHP), as well as setting the permissions correctly. We strongly encourage making modifications to the script to help with debugging.

8 Scenario #1: Pikachu, Meowth, and Cookies (XSS)

In this scenario, you will mount a cross-site scripting attack against all versions of the link sharing website, stealing the bot's (Meowth) login cookies, and using it to unlock the next level.

Helpful links for XSS:

- Javascript: http://www.w3schools.com/js/js_intro.asp
- XSS intro: https://owasp.org/www-community/attacks/xss/
- Tutorial on XSS: http://excess-xss.com/
- XSS Filter Evasion Cheat Sheet: https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

As you progress through the problems, the filtering will get more challenging and you will have to think of more creative ways to evade the filters.

List of filters:

1. No filter
2. Filters "script"
3. Filters "document", "(", ")", "<", ">"

4. Filters "script", "style", "on", and " " (space) (think carefully about what these filters do!)
5. Filters "<", ">"
6. Filters using `s/[()<>+]//g` (that's a regular expression that removes all the characters in the square brackets from the input string)
7. Filters using `s/[bcdfihzjrst<>]//ig` (similar to above, but ignores case for letters as well)
8. Filters using `s/[0-9a-z]//gi` (removes all numbers and all letters from input string)

XSS attack process:

1. Login to the server, select a problem.
2. Start by typing JavaScript into the "Send me an image link!" box
3. See if you can get your browser to execute JavaScript (see if you can evade filters)
4. Craft JavaScript to steal your own cookie and send it to a server (the one we set up previously)
5. Enter your attack JavaScript into "Send me an image link!" box
6. Your browser will load the confirmation/error page (the page will complain about the URL being invalid)
7. Copy the URL of the error page the address bar (Hint: if your attack involved redirecting away from the error page, check out the tips at the very end of the lab description for getting that error page URL)
8. Go back to the URL entry page
9. Paste the URL into "Send me an image link!"
10. Wait ~30 seconds for the bot to visit your link
11. If successful, your server will record the value of the bot's cookies (`authenticated=<something>`)
12. Copy the value, and use the Storage Inspector in Firefox's developer tools to create the same cookie (with `name=authenticated` and `value=<something>`) for yourself
13. Click "Open Safe #" on the top right corner, if you got the right cookie, the page will say "Congratulations! ..."
14. The button for the corresponding problem should turn green when you solve it
15. Repeat for the rest...

9 Scenario #2: Jailbreak (SQL Injection)

For this scenario, you will need to perform a SQL Injection attack. For scenarios 2 and 3, you may need to go read up on various aspects of SQL syntax, restrictions, and quirks. We're using mySQL for this problem.

Note: Some attempted SQL injection attacks may be blocked by CSE's web application firewall (WAF). If it takes a long time to load the page, you probably hit the WAF. You can try adding spaces or similar characters where possible, and when in serious doubt, check with the course staff if you're on the right track. The correct solution can bypass the WAF.

Some helpful links:

- SQL injection: https://www.owasp.org/index.php/SQL_Injection
- SQL (and SQL injection): http://www.w3schools.com/sql/default.asp

10 Scenario #3: Hack your 4.0 (CSRF)

For this scenario, you will need to perform a Cross-Site Request Forgery attack. You will need to investigate the different parts of the grading system to understand how a CSRF might be possible and what likely form submission endpoints would be useful to an attacker.

Some helpful links:

- CSRF Guides https://owasp.org/www-community/attacks/csrf
- What is CSRF (2003) http://talks.php.net/show/xss-csrf-apachecon2003/13

11 FAQs and Hints

Issue: My php script isn't working!

- Your script must run on homes.cs.washington.edu! You'll need to use one of your group members' CSE accounts to host on homes.cs.washington.edu. More details for how to access and edit your files there can be found here: https://homes.cs.washington.edu/FAQ.html.
- Double check that you have the file permissions correct (see above).
- If you haven't manually visited your script (i.e., just loaded it in your own browser and give it some value for the cookie) and seen it save your cookie, do that first.

Issue: Is the clicking bot working?

- Probably :) Check using a previously working script for a different problem, contact the course staff otherwise.

Issue: My XSS exploit is not working!

- First things first: Have you made sure that you can steal your own cookie successfully, by visiting the attack URL yourself in your own browser? If that does not work, then it won't work when the clicking bot clicks on it either.
- Don't create a popup / new window. We have popups blocked in our browser! Who still has popups enabled these days?
- You will need to host your page on homes.cs.washington.edu.
- Use https not http for your URLs.
- Check that your quote characters are plain `"` (smart quotes aren't the same character!)
- Double-check what the filter for the problem is and whether you've missed something.
- Are you trying to use `fetch()` to make a request? This will not always work (though we are not sure why). Try another way of loading a URL or redirecting the page instead.
- The XSS cheat sheet has examples like `<IMG SRC="javascript:alert('XSS');">`. It notes, but you might miss, that this actually does not work on modern browsers. But there are other tags where you can apply similar principles, i.e., tags that take a src attribute. There's even one in the cheat sheet...
- Are you trying to cleverly encode an HTML tag or attribute? This won't work. You can only encode things inside strings to evade a filter, not HTML tags/attributes themselves. So for example, if `on` is filtered, you won't be able to find a way to use an "onload" attribute no matter what you try. You'll have to find another way.

Issue: My SQL exploit is not working!

- (For SQL2:) Getting a SQL error that looks something like "Mysql Error: You can't specify target table 'sql2' for update in FROM clause"? You're probably on the right track! To deal with this, you'll want to select in a subquery. This might help: https://web.archive.org/web/20210731114958/https://www.xaprb.com/blog/2006/06/23/how-to-select-from-an-update-target-in-mysql/. (Note: This hint is not designed to help you figure out the solution if you're lost, so ignore it for now if it doesn't make any sense.)

Other tips:

- Always `view-source` of the resulting error page to check for bugs in how your intended HTML might have been interpreted.
- Another reason to view-source: Not all error pages are the same, so your strategy might not always be the same.

- Some approaches to solve the XSS problems cause the page to immediately redirect upon submission away from the page that complains about the URL being invalid. This is problematic / annoying because you need to copy the URL from that page to try resubmitting it! Here are a couple ways you can get around this:
- Modify the tag that you're passing in so that it does not render when you submit the link. For example, changing `<body …>` to `<bdy …>`. Then you can fix it back to body when you resubmit the link.
- In Firefox, you can click History on the top bar to view the URLs you have visited recently. Your result page URL should be here and you can right click and copy it!