CSE 484: Lab #2, Autumn 2024

Web Attack Lab

Due: Friday November 15th, 11:59pm
Turn in: Gradescope assignments, see Deliverables
Individual or group: Solo or partners
Points: 32 total (23 for sploits, 9 for writeups) plus some extra credit (see below)

Before you start

- Please use the sign up form here: https://forms.gle/CJamNmJqk8S594TM7 (Use your **@cs.washington.edu** Google account, *not your UW Google account.*)
    - If you are working in a pair, you only need to submit the form once, with both of your UW NetIDs included.
- We will grant access approximately once every 10 minutes, once we open up the lab.
- **Notice that there is an FAQ at the end of this document! You may want to revisit it periodically as you get stuck.**

Server Address:
https://cse484.cs.washington.edu/lab2

Deliverables

In addition to successfully completing the exploits on our server, please submit an individual writeup to Gradescope that, for each exploit:

- Provide the payload you found (i.e., what you entered into the text box on our server)
- Provide a short description for why it works (~1-4 sentences)
- Any accompanying files you created (e.g., your PHP script)

Please note that there is an opportunity for partial credit here: if you can't get an exploit to work, please still submit what you tried and why you thought it might work.

Goal

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.

Points

The following is a breakdown of the points for each problem. **Each writeup is worth an additional point, plus a point for uploading any supplemental files (e.g., php, html) you created.**

Scenario #1

Problem #

1. 2 points
2. 2 points
3. 3 points
4. 3 points
5. 3 points
6. 1 points (**optional, extra credit**)
7. 2 points (**optional, extra credit**)
8. 2 points (**optional, extra credit**)

Scenario #2

Problem #

1. 2 points
2. 3 points
3. 2 points (**optional, extra credit**)

Scenario #3

Problem #

1. 5 points

(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…

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

```
https://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.

1. Host your webpage at `homes.cs.washington.edu`, follow this link to read the FAQs.
2. 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. (Hint: This should not take more than 10 lines of code). Here are some hints on what you will need:
    a. PHP get variables
    b. PHP tutorial
    c. Using PHP to write to a file (useful for saving cookies)

    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 :)
3. Now, you can try out your cookie receiver by using your browser to navigate to `homes.cs.washington.edu/~<username>/cookieEater.php?cookie=secr`

`etCookieValue`, assuming your php file is named `cookieEater.php`. If your php script records the value `secretCookieValue` correctly, you can get started!

4. We strongly recommend having your php script record *every* visit it gets, not just the cookie value. This will make debugging your xss much easier.

If your script does not work, you might want to check whether your PHP script can be run by the Apache server -- in particular, you might need to set the file permissions to be world-readable. We recommend **chmod o+r,o-w for your php script** (world-readable but not world-writable) and **chmod o+w,o-r for the file to which you're writing cookies** (world-writable but not world-readable, so other students cannot read your stolen cookies, but Apache can write to it).

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: link

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

Scenario #2: Jailbreak (SQL Injection)

For this scenario, you will need to perform a SQL Injection attack.

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

Scenario #3: Hack your 4.0 (CSRF)

For this scenario, you will need to perform a Cross-Site Request Forgery attack. Some helpful links:

- https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)
- What is CSRF: http://talks.php.net/show/xss-csrf-apachecon2003/13

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.
  - Make sure your code is not creating a popup / new window. We have popups blocked in our browser! Who still has popups enabled these days? ;)
  - Make sure your attack is on `homes.cs.washington.edu`.
  - Make sure you are using https not http for your URLs.
  - Make sure " is used instead of ” for all quotations
  - 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!