University of Chicago
CMSC 23200/33250: Introduction to Computer Security, Winter 2020
Course Staff: David Cash, Alex Hoover, Rohan Kumar, Blase Ur, Valerie Zhao

Assignment 4: Web Security

Due at 11:59pm Thursday, February 6

Introduction

This assignment puts into practice a few different web security attacks we've learned about.

In this assignment, the flags you are aiming to capture differ by problem. In fact, for some problems, you are intending to cause an action on a server without capturing any flags.

Start early! There exist very short solutions to each problem, but you will need to learn new concepts and how to use new languages and techniques. Before getting to the problems, we discuss the rules for this assignment and how you can access the assignment infrastructure for your attacks.

Rules

Collaboration policy. Please respect the following collaboration policy: You may discuss problems with up to 3 other students in the class, *but you must write up your own responses and your own code. You should never see your collaborators' writing, code, flags, or the output of their code.* At the beginning of your submission write-up, you must indicate the names of your (1, 2, or 3) collaborators, if any. You may switch groups between assignments but not within the same assignment.

This should go without saying, but you should not capture anyone else's flags.

Sources. Cite any sources you use. You may Google liberally to learn helpful aspects of HTML/JavaScript, how web browsers work, or about web security techniques. Unlike some past assignments, you are welcome to perform additional Google searches on other topics related to this assignment, but be warned that these are unlikely to help you solve the problems on their own. You **must**, however, note at the end of your solution for each task any topics you ended up Googling to complete that task.

Campuswire. We encourage you to post questions on Campuswire, but do not include any of your code in the Campuswire posts. If you have a question that you believe will reveal secrets you have discovered while working on the assignment, post privately to just the instructors. If you have a question that you believe will be of general interest or clarifies the assignment, please post publicly. If you are uncertain, post privately; we will make public posts that we believe are of general interest.

Outside attacks. Your attacks for this assignment should be those discussed below. Do not attempt to compromise our server, sniff your classmates' network traffic (i.e., no Wireshark this time), or do other nefarious things. You will not receive credit for breaking into the server. In fact, you will lose credit for doing so.

Grading. Responses will be graded for correctness and clarity.

Assignment Tech Set-Up and Overview

This assignment involves visiting and interacting with pages on your favorite server: `http://insecurityclass.cs.uchicago.edu/`

Unlike in Assignment 2, you will *not* be using a Python script to make queries to our server. Instead, you will be visiting pages on our server in a web browser.

As in past assignments, you may implement your code in whatever programming language you choose. That said, not all of the problems require writing any substantial amount of code. In fact, many problems simply involve generating some very carefully crafted inputs or interactions with the server. You will be submitting these carefully crafted inputs or interactions alongside descriptions of how you generated them and how they work. However, in the process of crafting these inputs, you will likely have to experiment on your own. You should submit any test pages / minimal working examples you made for yourself in figuring out how to craft these inputs.

What and How to Submit

You will submit a set of files:

1. A file `<YOUR CNETID>-writeup.pdf/txt` that briefly describes how you solved each problem. This file is shared across all seven problems. You will include any flags you find (clearly delineated in bold) in this file, alongside the carefully crafted inputs or interactions you generated for Problems 1–7.

2. Individual files, clearly labeled by problem, that you generated in solving the problems. These are discussed for each individual problem below.

You will upload these files to Canvas. You may zip/tar/gzip/7zip/etc. them or submit them individually.

Final Notes and Hints

Please don't DDOS the server. It will support thousands of queries per student (which you will need), but not millions. If you are very sure that an input should be working (especially if it did earlier) but it no longer does, please let us know on Campuswire. There is some small chance that a large amount of traffic will cause our automated infrastructure problems.

This assignment was built last year by Blase fresh for this class and substantially expanded this year. It is the first time it has been deployed in any setting. Please let us know if you find bugs or need instructions fleshed out by posting on Campuswire.

Problem 1: Authentication By Cookie (10 points)

Following in the footsteps of successful cryptocurrencies like Bitcoin, Ethereum, Monero, Dogecoin, and Zcash, the world's newest, most exciting cryptocurrency — Dcash — is about to have its Initial Coin Offering (ICO). Unfortunately, since time was short, the mysterious, shadowy figure who created Dcash put together a website to keep track of Dcash in a very quick and slipshod manner. As a result, it has many basic security holes. In this problem, you will bypass a very simple security mechanism.

On `https://insecurityclass.cs.uchicago.edu/1/` the creator of Dcash has set up a portal for individuals to obtain a secret access code to begin trading in Dcash in advance of the ICO. People who send a specified amount of Bitcoin to the creator are assigned premium accounts, which enables them to generate their secret access code. Unfortunately, the way they keep track of, and verify, who is a premium member isn't very smart. Explore this by going to `https://insecurityclass.cs.uchicago.edu/1/index.php?username=<cnetid>`, replacing <cnetid> with your CNetID. Take a look at how persistent state is being kept. Leveraging what you learn, and without spending any Bitcoin, become a premium member and get yourself an access code to the Dcash ICO.

What to submit.

You are not required to write any code to solve this problem.

In your shared write-up file, `<YOUR CNETID>-writeup.pdf/txt`, describe your approach to solving this problem and **include your secret access code in bold text**. Also briefly describe why the way the Dcash creator tried to handle authentication is fundamentally incorrect. What should they have done differently?

Problem 2: Easy CSRF (15 points)

The founder of Dcash has decided to give everyone in the class 1000 Dcash in advance of the ICO. On `https://insecurityclass.cs.uchicago.edu/2/` they created a tracker to know who has particular amounts of Dcash. Why might some people have a value different than 1000 when you look? That's a great question. The founder decided to let early holders of Dcash transfer some of their Dcash to other users. Of course, some authentication is necessary. Thus, transfers are only valid when presented alongside the secret access token discovered in Problem 1. That is, the sender's CNetID and that sender's secret access token must be presented in conjunction for the transfer to be accepted. This means that you can give other people your Dcash, but you can't take theirs. Or can you?

The founder of Dcash wanted to be able to transfer Dcash arbitrarily between users, so the secretive founder has configured the site such that when they are logged in and making the transfer request, they do not need to provide an access token.

The founder is also kind of a creeper and likes to look up the webpages of people who hold Dcash. Can you make a homepage that uses some clever CSRF to transfer Dcash from one of the instructors ("blase" or "davidcash") to yourself?

Follow these instructions to set up your own homepage on the CS department servers. (The following steps assume you have a CS account; please let us know if you don't have one). Log into `linux.cs.uchicago.edu` (e.g., by running `ssh <username>@linux.cs.uchicago.edu`) and follow the instructions on `https://howto.cs.uchicago.edu/techstaff:personal_homepage` to create a homepage if you don't have one already. Carefully follow the steps outlined on Techstaff's webpage to set permissions on the different directories and on **each** file you create, including those we ask you to create to solve different problems. (If you don't set the permissions properly, the file will not be accessible to the founder, or to anyone.)

Create your exploit file as assignment4-2.html so that, for example Blase's page would be accessible at `https://people.cs.uchicago.edu/~blase/assignment4-2.html` if he hadn't already deleted it. Seriously, don't forget to set the permissions on this file or else it can't be opened.

Every few minutes, the founder visits your page (over HTTPS). If your CSRF works, then you will notice your transaction reflected in the amounts of Dcash your sender and recipient have within a few minutes.

After you get the exploit working, please delete assignment4-2.html from your web directory. It is a violation of the university academic integrity policy to try looking at other students' HTML pages.

For both student privacy and to facilitate your testing, all Dcash amounts reset to 1000 every hour on the hour. That is, don't be shocked when your ill-gotten gains disappear hourly.

What to submit.

Upload your assignment4-2.html file to Canvas.

Furthermore, in your shared write-up file, `<YOUR CNETID>-writeup.pdf/txt`, describe your approach to solving this problem. Also briefly describe the primary reasons the Dcash creator was vulnerable to a CSRF attack on the page they made.

Problem 3: Difficult CSRF – Overcoming CSRF Tokens (15 points)

Feeling a bit embarrassed by falling victim to CSRF, the founder changes their system. The new version can be found on `https://insecurityclass.cs.uchicago.edu/3/` . Note that this new version's transfer page works quite a bit differently than the previous problem's.

Notably, they added a CSRF token to the page and now use POST requests, rather than GET requests. To compute the CSRF tokens, the founder is using a JavaScript crypto library. Unfortunately, the founder made some critical errors in designing the CSRF token, enabling you to still steal Dcash despite their attempt at security!

The founder continues to be a creeper. Every few minutes, the founder visits this new page you created (over HTTPS). If your CSRF works, then you will notice your transaction reflected in the amounts of Dcash your sender and recipient have within a few minutes. **Create your exploit file as assignment4-3.html** in your directory on `people.cs.uchicago.edu` and again delete assignment4-3.html from your web directory after you get the exploit working.

Hint: Learn how to make AJAX requests. Note that AJAX requests do not send cookies by default, which would prevent the founder's cookies being sent by default when you make an AJAX request on their behalf. However, if you set `AJAXobject.withCredentials = true;` while making the request (replacing *AJAXobject* with the variable in which you stored your XMLHttpRequest object), the cookies will be sent.

For both student privacy and to facilitate your testing, all Dcash amounts reset to 1000 every hour on the hour. That is, don't be shocked when your ill-gotten gains disappear hourly.

What to submit.

Upload your assignment4-3.html file to Canvas.

Furthermore, in your shared write-up file, `<YOUR CNETID>-writeup.pdf/txt`, describe your approach to solving this problem. Also briefly describe (i) what the Dcash creator did wrong in designing their CSRF token and (ii) what the Dcash creator could have configured on the server to further protect themselves from this problem. (Note that we talked about that last part in class.)

Problem 4: CSRF+XSS in Spite of Sanitization (15 points)

After falling victim to CSRF two problems in a row, the founder learned not to be a creeper. However, the founder is also narcissistic and still likes to get messages from their devoted followers. It's fair to assume that they read these messages in their web browser – with HTML and JavaScript enabled – and that they always stay logged into their site in that browser.

The new version of the system can be found on `https://insecurityclass.cs.uchicago.edu/4/` .

Send the founder a friendly message. Put a lot of thought into it. And in doing so, figure out how to transfer lots of Dcash to yourself from others. Give the founder a little bit of time to read the message, though note that they read all of their messages within a few minutes both night and day.

However, the founder got a little smarter, so they wrote some regular expressions to filter out certain types of HTML from the messages you send. Unfortunately, they didn't get smart enough to write the regular expressions correctly. Think about their most likely rookie regex mistakes and try exploiting them.

For this problem, come up with **two different ways** of evading the founder's attempt at filtering and give yourself more Dcash through messages you send the founder.

Note that this problem shares its database with problem 2, so solutions to either problem could be changing values. As before, all Dcash amounts reset to 1000 every hour on the hour. That is, don't be shocked when your ill-gotten gains disappear hourly.

Note also that this problem is sort of a combination of CSRF and XSS in the way you evade filters and then use your code to forge a request.

What to submit.

In your shared write-up file, `<YOUR CNETID>-writeup.pdf/txt`, describe your approach to solving this problem and include the **two different messages you sent the founder in bold text**. Also briefly describe where you think the founder went wrong in their attempts to sanitize input.

You do not need to submit any code for this problem if you didn't use any. However, if you created any sample/test files (e.g., HTML files) or wrote any code to help craft your input, please submit those in their native format.

Problem 5: XSS-based Deanonymization (15 points)

At this point, you are probably curious who the founder of Dcash really is. In this problem, you will unmask them. You notice that when you visit your message page at `http://insecurityclass.cs.uchicago.edu/5/` it takes your username from your browser cookie (from Problem 1) and fills it into the page. Well, since you can send messages to the founder and they presumably will be reading your messages on a page with the same layout, you have an opportunity to figure out their name!

In this problem, you'll send the founder another message, this time using XSS to exfiltrate their name. The founder is still using the same (buggy) HTML sanitization filters from the last problem.

Of course, after your message extracts their name from the page, it is still running locally on the founder's browser, so you'll need this data somewhere.

To that end, we've provided a sample PHP script for your linux.cs machine that saves data you send it in a file (outside your web directory) called exfiltrated.txt.

As a first step, replacing <username> with your CNetID, log into the linux.cs machines: `ssh <username>@linux.cs.uchicago.edu`

This puts you in your home directory on linux.cs. Put the file `sink.php` (posted alongside the assignment on Canvas) in your `/home/<username>/html` directory, either through scp or by simply copying and pasting the contents of the file. Note that the vi and emacs text editors are both installed on linux.cs, so pick your poison.

Make sure to edit the `sink.php` file and replace <username> with your username in the file path!

Then, run the following commands to make a text file and ensure that www-data (the user under which Apache runs) will be able to access the files, as well as to ensure that this text file is readable only by you:

```
chmod 711 /home/<username>
touch /home/<username>/exfiltrated.txt
chmod 602 /home/<username>/exfiltrated.txt
chmod 711 /home/<username>/html
chmod 755 /home/<username>/html/sink.php
```

To test that everything is working, just go to `https://people.cs.uchicago.edu/~<username>/sink.php?id=1&data=foo` and then look at the contents of the text file, which appends all requests it gets. For example, I would run this as: `https://people.cs.uchicago.edu/~blase/sink.php?id=1&data=foo`

Yes, this means that you have the capability of writing data to your classmates' home directories on linux.cs. You absolutely MAY NOT do so. Once you complete this problem, be sure to delete both sink.php and exfiltrated.txt from your linux.cs account. Otherwise, anyone can fill up your home directory in perpetuity!

What to submit.

In your shared write-up file, `<YOUR CNETID>-writeup.pdf/txt`, describe your approach to solving this problem and include the **exact message you sent the founder in bold text**. Also include the **full name of the founder in bold text**. Note that if you only have a two-word name for the founder, you're not quite done. Also briefly describe how the founder could have protected against this attack.

You do not need to submit any code for this problem if you didn't use any. However, if you created any sample/test files (e.g., HTML files) or wrote any code to help craft your input, please submit those in their native format.

Problem 6: Bobby Tables Joins The Dcash Ecosystem (15 points)

The mysterious and secretive, but also careless, founder of Dcash is getting ready for the ICO and has put together a finalized version of the site to track how much Dcash each person has. You might notice they are making Dcash rather centralized, but we can ignore that for now. In any case, you've realized that you can go to `https://insecurityclass.cs.uchicago.edu/6/` and yet again see how much Dcash a particular user has. Don't be alarmed that your Dcash from previous problems has disappeared. According to this new database, everyone starts off with no Dcash.

Because we don't want to leave a vulnerable database open to the web, please use username: "cs232" and password: "ImissGCSSPalready" to access this directory.

The founder lets you query for a particular username to see how much Dcash they have. The things you type into this field are used for a MySQL query. Given the founder's track record, this is bad news for security. Use your best SQL Injection tricks to (separately) do the following two things:

1. Determine the full list of users in the database.
2. Give yourself some free Dcash.

If you weren't previously familiar with MySQL, you will probably have to read up on how queries are structured to solve this problem.

Note that, because the odds of one of you accidentally blowing up the database for this problem are high, it automatically resets every 5 minutes (on :00, :05, :10…).

What to submit.

In your shared write-up file, `<YOUR CNETID>-writeup.pdf/txt`, describe your approach to solving this problem and include (in bold) each value you entered into the name field to solve each of the two parts of this problem. Also include in this shared write-up file a list of the people identified in sub-part 1. Finally, briefly describe how the founder could have protected against this attack.

You do not need to submit any code for this problem if you didn't use any. However, if you created any sample/test files (e.g., HTML files) or wrote any code to help craft your input, please submit those in their native format.

Problem 7: Reflected XSS (15 points)

The founder has now made a login page at `http://insecurityclass.cs.uchicago.edu/7/` to access the system. Because the founder wants to make people feel at home, they have added a URL parameter for greetings, such as `http://insecurityclass.cs.uchicago.edu/7/index.php?greeting=cs232student` . Noticing that user-provided input is shown on a page, you might be wondering whether you can conduct a reflected XSS attack to steal the founder's password. Indeed you can, and that's what you will do here.

Experiment with these greeting messages and try to understand how you can craft a greeting that redirects form submissions to a page you control. You will probably want to create some sort of new sink file on your `people.cs` page that will serve to handle the redirected form submissions. The easiest approach would just be to modify the sink.php file we provided for an earlier problem (requiring you to learn just a little PHP), but it's ultimately up to you.

Trolling everyone at this point, the founder has decided to adopt the username "blase&david". Furthermore, the founder will only enterr their password and submit the form if they see "blase&david" autofill on the page and feel confident that everything is working as intended.

Note that the founder still has not learned to sanitize their inputs; they are still using the buggy sanitization from earlier problems. You will probably also want to read up about how to URL encode your greeting. Note that you can use JavaScript in the browser console to URL encode a string. The output should turn special characters into tokens beginning with percent signs.

When your specially crafted message is ready, go to `https://insecurityclass.cs.uchicago.edu/7/message.html` and enter only the URL you want the founder to visit. If you crafted it to correctly redirect the form output to the page you control and autofill the username "blase&david", you can expect the founder to load the page and type in their password within a few minutes.

And that's how you get the password!

What to submit.

Upload the file you wrote as your sink to Canvas with whatever filename you used. (The message you send the founder should make obvious to us what this file name will be.)

Furthermore, in your shared write-up file, `<YOUR CNETID>-writeup.pdf/txt`, include **in bold the exact message you send the founder on message.html**. In addition, briefly describe your approach to solving this problem and also briefly point out some of the key security mistakes the founder made that enabled this attack.