14-828/18-636: Homework 1

Released: Thursday, January 30, 2020
Due: Thursday, February 13, 2020 by 1:29 PM Eastern Time

Name:

Andrew ID:

Scores

Problem 1 (50 pts max):

Problem 2 (20 pts max):

Problem 3 (10 pts max):

Problem 4 (10 pts max):

Problem 5 (10 pts max):

Total (100 pts max):

Guidelines (Please read before starting!)

- Be neat and concise in your explanations.
- You must use at most one page for your explanation for each Problem (code you wrote may be on additional pages). Start each problem on a new page. You will need to map the sections of your PDF to problems in Gradescope.
- To access the CTF problems, create the SSH proxy as per the guide on Canvas, and register for an account using the following link:

  `http://192.168.2.82/`

- For CTF problems, **you must use the following format in your explanation**:
  - CTF Username
  - Flag
  - Explain the vulnerability in the program, and explain conceptually how that vulnerability can be exploited to get the flag.
  - How did you exploit the vulnerability? List the steps taken and the reasoning behind each step. The TA grading should be able to replicate the exploit following the steps. Feel free to make references to your code! **Note that *"use XYZ online solver"* is not sufficient - you must explain how the online solver derived the answer for full credit.**
  - Append your source code in the same writeup. Your source code should be readable from the writeup PDF itself. Note that this does not count towards the page count above.

  Omitting any of the above sections would result in points being deducted.
- Please check your English. You won't be penalized for using incorrect grammar, but you will get penalized if we can't understand what you are writing.

- Proofs (including mathematical proofs) get full credit. Statements without proof or argumentation get no credit.
- There is an old saying from one of my math teachers in college: "In math, anything partially right is totally wrong." While we are not as loathe to give partial credit, please check your derivations.
- **This is not a group assignment. Feel free to discuss the assignment in general terms with other people, but the answers must be your own.** Our academic integrity policy strictly follows the current INI Student Handbook `http://www.ini.cmu.edu/current_students/handbook/`, section IV-C.
- Write a report using your favorite editor. Note that **only PDF submissions will be graded.**
- Submit to Gradescope a PDF file containing your explanations and your code files before 1:29pm Eastern Standard Time on the due date. You can find the timing for EST here: `https://time.is/EST`. Late submissions incur penalties as described on the syllabus (first you use up grace credits, then you lose points).
- If you choose to use a late day, please inform the instructors via a post on Piazza.
- Post any clarifications or questions regarding this homework to Piazza.
- Good luck!

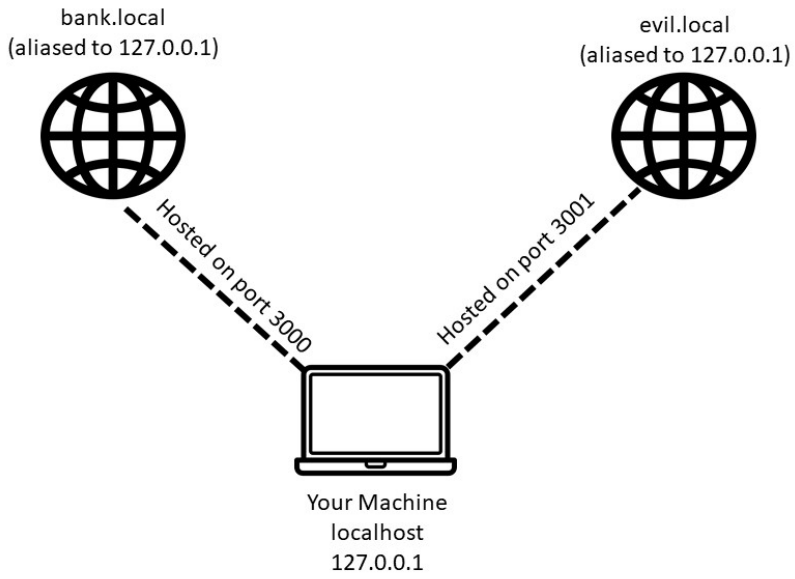
1 Of CORS It's Easy! (50 points)

The web is built on specifications, hence it is important for us to learn how to read, understand, and experiment with how specifications are implemented in the browser. To help you with this, we dedicate this section of the homework to walk you through cross origin resource sharing (CORS). CORS is described in depth in the fetch protocol for browsers - you can read the specifications in the living standards here: `https://fetch.spec.whatwg.org/`).

You may download the experimental set up on Canvas. The experimental code contains a barebone node.js server (under bank.local) as well as some boilerplate code on making XMLHttpRequest (XHR) requests via Javascript (under evil.local). We will be setting up the system on your local machine, so follow the instructions below carefully! Occasionally, for the sake of brevity of this homework writeup, you might be asked to look for resources online to set up the environment. If after some trying you still are unable to set-up the environment, please create a post on Piazza so your classmates and the TAs can help!

1.1 The Setup (10 Points)

In our setup, we will be simulating cross origin requests on our own machine via aliasing two different domains: **bank.local** and **evil.local** to **localhost (127.0.0.1)**. We will do this by modifying the hosts file on our operating system. Instructions vary across different operating systems (we trust that you would be able to find these instructions online!), and your final set up should look like the diagram shown below:



1. (2 points) Set up and run the node.js server for **bank.local**. You may have to install node.js on your system - instructions vary but they can be found here:

   `https://nodejs.org/en/download/package-manager/`.

   Install all relevant dependencies using the following command under the bank.local directory:

   ```
   npm install
   ```

   Run the server with the following command after the above is completed:

```
node index.js
```

Do the same for **evil.local**. Show that both servers have successfully started.

2. (2 points) Please read through all the code to get an idea of what is happening! We ask that you make a small modification - modify the homepage such that it shows "Hi welcome to *ANDREWID* bank", where *ANDREWID* is your Andrew ID. You may have to restart the server to see the changes.

   Show that you have successfully modified the homepage of the bank.local node.js server (under `http://127.0.0.1:3000`) by including a screenshot in the writeup.

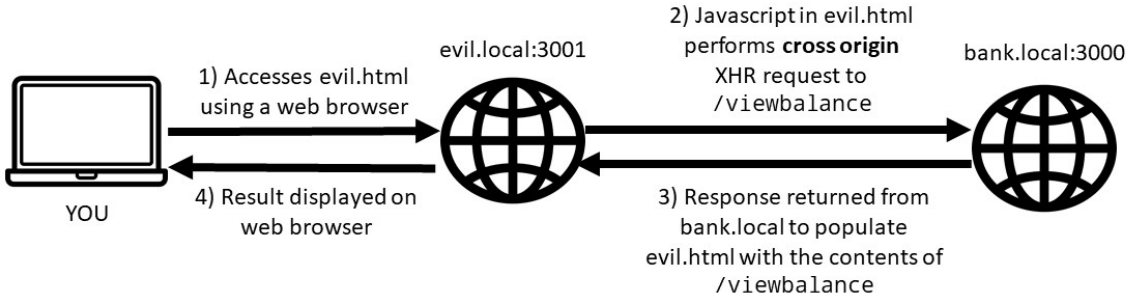
3. (3 points) Modify the hosts file such that `http://bank.local:3000` and `http://evil.local:3001` resolves to `http://127.0.0.1:3000` and `http://127.0.0.1:3001` respectively. Explain briefly the changes that you made and what it means on a network level.

4. (3 points) Does `http://bank.local:3000` have the same origin as `http://evil.local:3000`? Why or why not?

1.2 Simple and Not So Simple Fetches (30 points)

In this section, we will learn to distinguish between simple fetches and complex (pre-flighted) fetches for cross-origin requests. Please take some time to refresh your memory on the differences between the two kinds of fetches from the slides.

Before we continue, we need to understand how we are interacting with the service. This is illustrated in the diagram below:



Note that the endpoint `/viewbalance` is protected by a session cookie. Here, we are experimenting to see what headers in the CORS protocol would allow us to make cross origin requests which sends the session cookie over. In addition, we would also like to know which combination of CORS headers allow us to read the sensitive response returned from the server.

1. (2 points) **Please load the home page of `http://bank.local:3000` first to get the cookie.** Load evil.html as per the interaction diagram above. Is the cross origin request made to `http://bank.local:`

`3000` a simple fetch or a pre-flighted fetch? Provide evidence of your claim by showing and labelling the requests made in the Network tab.

You can find the tab in the developer tools on Chrome by pressing F12; other browsers should have similar facilities.

2. (3 points) Add a custom header by uncommenting the following line in evil.html:

   ```
   xhr.setRequestHeader('X-Custom-Header', '14828');
   ```

   Reload `http://evil.local:3001` by restarting the server. Is the cross origin request made to `http://bank.local:3000` a simple fetch or a pre-flighted fetch? Provide evidence of your claim by showing and labelling the requests made in the Network tab. With reference to the specifications for the fetch protocol, explain why the request was made in this mode.

3. (5 points) Experiment with different CORS headers together **with the custom header** above and try to reproduce the results of the table found here:

   `https://fetch.spec.whatwg.org/#cors-protocol-and-credentials`.

   Note that "omit" is identical to setting withCredentials to *false*, "include" is identical to setting withCredentials to *true*, and you should also replace "https://rabbit.invalid" to "http://evil.local:3001". Make sure you are able to do this before moving on to the next part!

   Suppose that the request credentials mode is *"include"*, and Access-Control-Allow-Origin is "`http://evil.local:3001`", but Access-Control-Allow-Credentials is *"false"*. Make sure that evil.html sends the request with the custom header. Was the request **sent with the cookie** and processed by the server? If so, can the response be read cross origin?

4. (5 points) Comment out the line with the custom header in evil.html and rerun for the scenario given above. Explain the similarities and differences in the request and response, justifying why they happened with reference to the specifications.

5. (5 points) With knowledge from the above experimentation, explain why CORS itself is not able to prevent Cross Site Request Forgery (CSRF). Suggest one mitigation for CSRF, and explain how it prevents both CSRF attacks for both simple and pre-flighted fetches.

6. (10 points) In your new job, you come to learn about different kinds of Cross Site Request Forgery (CSRF) protections. Your team has been tasked to suggest one CSRF protection for an upcoming project. Your colleague strongly believes that the use of a double-submit cookie can prevent such attacks.

   The double submit cookie implementation embeds a randomized CSRF token in the cookie AND in a custom header (e.g. X-ACME-Form-Key: [CSRF token]) of the request, where the CSRF token is generated randomly per session. Both tokens are then checked for correctness in the backend by checking if the CSRF token in the cookie is the same as the CSRF token in the custom header. You are asked to critique his design *(this actually happened in real life FYI)*.

   Name one advantage of using double-submit cookies over other CSRF protections, such as CSRF synchronization tokens. Show that the use of the double-submit cookie implementation above **can under certain assumptions** prevent CSRF attacks for both simple fetches and pre-flighted fetches. State what those assumptions are, and describe a scenario in which the assumptions can be broken.

1.3 My Headers Are Aching (10 points)

Over the course of using CORS, you might encounter issues and plenty of advice exists on the web to *help* resolve them. We will look at some suggestions and evaluate the security of the suggestions listed below. You may assume that the developer only wants to protect resources which are protected behind some credentials from cross-origin access, and that the credentials are cookies.

HINT: you may wish to use the test environment to experiment to see if any of these configurations would lead to unintended consequences.

1. (3 points) A StackExchange commenter suggests returning the following CORS header for all resources on the server:

   ```
   res.header("Access-Control-Allow-Origin", "*");
   ```

   Is this a good security practice or a bad one? Explain your case briefly.

2. (3 points) A different StackExchange commenter suggests returning the following CORS header for all resources on the server:

   ```
   res.header("Access-Control-Allow-Origin", req.get('origin'));
   ```

   Is this a good security practice or a bad one? Explain your case briefly.

3. (4 points) Bob wants to protect all his POST endpoints from cross-origin requests. To do so, he enforces that a custom header with a fixed value (e.g. X-NO-POST: "true") is always sent with every request at all his endpoints. If a request doesn't contain the custom header, the request is dropped on the server side. He also changes the CORS headers to prevent browsers from sending POST requests as follows:

   ```
   res.header('Access-Control-Allow-Methods', 'GET, OPTIONS');
   ```

   Is this a good security practice or a bad one? Explain your case briefly.

2 XSSessive (20 points)

Log into the 14828 CTF Server. Create an SSH proxy tunnel to 14741hw.ini.cmu.edu using the provided directions from Canvas.

Configure your browser to use this proxy, then create an account. You can pick any username you want (so your classmates don't know who you are), but please enter your correct name when you register (visible to instructors only). You must also let us know what your username is in your writeups. Register with the following url: `http://192.168.2.82/`

Do not use the webshell. Copy/pasting is hard. Follow the directions provided on Canvas to ssh to the server.

Note that it is trivial to get the flag through alternative means - we ask that you do the XSS to get the flag. We will grade you based on your writeup, and not whether you have the flag. If you give us the flag without the XSS, we will not give you any points.

1. (5 points) Solve the Reflected XSS problem by calling *alert*. Include an explanation of which field in the form is injectable. Submit a writeup containing your CTF username following the guidelines on the first page.
2. (5 points) Solve the Stored XSS problem by calling *alert*. Include an explanation of the difference between a reflected XSS and stored XSS. Submit a writeup containing your CTF username following the guidelines on the first page.
3. (5 points) Solve the DOM-XSS problem by calling *alert*. You may find that traditional payloads like the below don't work:

   ```
   <script>alert('1');</script>
   ```

   Include an explanation why the above payload won't work, as well as the differences between DOM-XSS and the other XSS types. Submit a writeup containing your CTF username following the guidelines on the first page.
4. (5 points) Suggest a fix (other than sanitization) that can prevent the attacks conducted above. State how the fix is able to prevent the attacks in each case.

3 Bad CSP (10 points)

CSP seems unbreakable, but what happens if you misconfigure the CSP? Solve the Bad CSP problem on the 14828 CTF Server. Submit a writeup containing your CTF username following the guidelines explained earlier. Also include in your writeup how the CSP policy can be fixed to prevent your exploit from working.

You might want to use the following site to help you: `https://csp-evaluator.withgoogle.com/`

4 Baby's First CSRF (10 points)

How can you say you know browser security if you have never done a CSRF attack before? Solve the Baby CSRF problem on the 14828 CTF Server. Submit a writeup containing your CTF username following the guidelines explained earlier.

5 I Am Root! (10 points)

The admin really love their site and would like to look at different profiles. Can you hijack their session? Solve the I Am Root problem on the 14828 CTF Server. Submit a writeup containing your CTF username following the guidelines explained earlier.