Flag

Part 1 Due: Thursday, February 28, 2019 @ 11:59pm
Part 2 Due: Monday, March 11, 2019 @ 11:59pm
Bob's Router (CS162 Only) Due: Monday, March 11, 2019 @ 11:59pm

Contents

1 Introduction

Minion University's Department of Computer Science, like most, has implemented their own in-house course management web application, the FLAG (Fast Lightweight Administrative Grades) Portal, and the creators have asked you to test its security.

The application is presented to you as a black box. You are given the usernames and passwords of regular, unprivileged users to see how the site works. Discover how it's supposed to work, and then figure out how to break it.

1.1 Running the Web App

Each student gets their own unique instance of the web application to attack.

The necessary credentials to access the website are located in the `/course/cs1660/student/<your-login>/flag/credentials` directory. (Please keep your credentials private from other students.)

Your personal instance is located at the unique IP address found in the `IP` file and is protected via a username and password which can be found in the `LOGIN` and `PASSWORD` files respectively. Note that this is not the same as the login scheme that the application itself implements and you can assume this credential system is outside the scope of potential vulnerabilities in the website.

1.2 Grading

This project consists of two parts (colloquially referred to as "Flag 1" and "Flag 2"). For CS166 students, each part is worth 50 points, for a total of 100 points overall. CS162 students must also complete an extra component ("Bob's Router"). For CS162 students, Flag 1 and Flag 2 are each worth 40 points and Bob's Router is worth 20 points.

There are extra credit opportunities in this project for all students—see each part of the handout for more details. The maximum score you can receive on the whole project is 115/100.

2 Late Handins

The late day policy for this assignment is different from previous projects, so please pay close attention to the following rules.

- For Flag 1, *you must declare the number of late days you wish to use prior to the assignment deadline* using this form: `https://goo.gl/forms/TCk8YOc1f9R2pf8N2`. You may update your response to the form multiple times before the deadline and the latest form response *prior to the deadline* will be considered.

If you declare late days for Flag 1, you will be unable to start Flag 2 of the assignment until after the number of late days you declare. If you declare some number of late days prior to the deadline but finish the assignment after the deadline earlier than expected, please email the HTA list to let us know that you'd like to start working on Flag 2 and we will restore all unused late days.

Finally, late days on Flag 1 only apply to the Flag 1 deadline (that is, they do not extend the Flag 2 deadline as well).

We *will not* accept any late handins if you do not declare your late days prior to the deadline.

- For Flag 2, late days will be applied automatically.

Please consult the syllabus for more information about late handins.

If there are extenuating circumstances preventing you from completing an assignment on time (e.g., illness), please contact the instructor before the assignment is due via the "Extension Requests" form on the `http://cs.brown.edu/courses/cs166/resources/` page.

3 Collaboration Reminder

Large parts of this assignment can be completed easily once an idea is known. Therefore, it is once again important to *not* discuss anything about this assignment with other students. Please do not discuss particular tools, particular parts of the app that you have attacked, or generally anything about your progress. It's simpler to say nothing than to try and tiptoe around what is okay and what would be unfair help, or even damage, to other student's work. Direct all questions to TAs via Piazza or at hours.

Part I

Vulnerabilities, Exploits, and Remediation

4 Assignment

You should discover **5 distinct vulnerabilities** in the web application, and for each, submit an exploit. Each exploit must allow you to perform an action in the web application that you are not supposed to be able to do. For example, changing a grade, logging in as an admin user, deleting a particular user's data, or accessing another student's grades would all count as successful exploits. Social engineering, phishing, and denial of service do not count as exploits. Finally, the scope of this project is the website itself. Attacks that rely on any networking (for example, eavesdropping on unencrypted connections) do not count.

4.1 Distinct Vulnerabilities

In order to make clear what makes two vulnerabilities count as distinct, we have compiled a list of every vulnerability we could possibly imagine coming up in a project like this. To decide whether two vulnerabilities count as distinct, figure out what categories they belong under from this list. If they both belong to the same category, then they are not distinct; otherwise, they are.[1]

- CSRF
- Cross-Site Data Access
- Stored XSS
- Reflected XSS
- DOM-Based XSS
- UI Redress
- Cookie Injection
- Client-Side HTTP Parameter Pollution
- SQL Injection
- OS Command Injection
- File Upload
- File Inclusion
- Path Sanitation Bypass
- Client-Only Input Validation
- Client-Hidden Sensitive Data
- Insecure Direct Object Reference
- Parameter-Based Access Control
- Referrer-Based Access Control
- Multi-Stage Functions with only some pages authenticated
- Session Cookie Prediction
- Session Fixation
- Bad Password Hashing

Note: Business logic vulnerabilities are considered on a case-by-case basis. If you find two or more business logic vulnerabilities, please consult the TAs to see if they count as distinct.

4.2 CS162

CS162 students are required to find and exploit **6 distinct vulnerabilities**.

[1] While we've discussed some of these vulnerability categories in lecture, many of the vulnerabilities in this project will either be new to you, or will at the very least not be exactly the same as you have seen before. Much of security is learning about previously unknown systems, so, for this project, we expect that you may need to teach yourself more than has been covered in lecture. If you find yourself at a point where you feel that you haven't been taught how to do something, that's okay!—you should feel confident that you can do it if you set your mind to it.

4.3 Extra Credit

You may find and exploit additional, distinct vulnerabilities for extra credit. In Flag 1, each additional exploit is worth 3 points.

5 Setup

5.1 Accounts

Each student in the course has a login for FLAG. Your FLAG username is your CS login, and your FLAG password is iam<username>. For example, if your CS login is alice, your FLAG username is alice and your FLAG password is iamalice. Each student has their own copy of the web app, so feel free to log in as each other or yourself.

Each member of the course staff also has a FLAG account under their CS login, but the passwords are not known to you. They are, however, poorly chosen passwords (though they are not simply random, short, alphanumeric passwords). They are more realistic, but certainly not passwords any of us would use in real life (especially after taking this course!).

5.2 Assumptions

- Assume the site is active. Users often log in, leave comments, visit profiles, and generally use all of the features of the site.
- When thinking about how you would exploit a vulnerability, imagine you can get any particular person to click on a link via some kind of trickery. In other words, you can assume you can email a link to any other student or staff member and we will click on it.
- Most parts of the application have been (intentionally) implemented poorly. That is, many vulnerabilities exist.
- All grades, comments, and handins are randomly generated, so don't feel bad if "your" login has bad grades on it. (Obviously, this reflects nothing about you!)

5.3 Resetting the Application

If you ever find that you have broken the website and would like to refresh it to its original state, you can do so by requesting the path `/reset.php`. This will delete the website and recreate it. It will also redirect your browser to `/setup`, which will generate all of the content for the website. If you request `/reset.php`, and are not redirected to `/setup` (for example, if you are using a command-line tool such as `curl`), you will need to request `/setup` before you'll be able to log in.

If you find that you have have broken the website to the point where the `/reset.php` endpoint no longer works, please email the HTA list and we'll reset the application for you.

6 Hints, Tips, and Tricks

6.1 Where to Start

It's vital that you first thoroughly see what the features of the application are. Figure out what you can do and as much about how it works as you can. Click on every link, check every box, and visit every page. If

you are having problems thinking of more vulnerabilities, first look at the lectures for ideas. You may want to get familiar with basic HTML and investigating the source code of particularly interesting pages. Get to know web developer tools like Firebug and Inspect Element. Think about what power you as a user have to modify the application beyond the ways that the developers expected you to.

6.2 Tools (and Restrictions)

Ask the TAs before moving forward with any broad external tools. Authoring tools such as Sublime Text or Eclipse are of course acceptable, as are applications that proxy web traffic and allow you to inspect requests/responses and modify/replay requests (like Burp Suite as specified below). More fully-featured web application security analysis tools such as Metasploit or sqlmap, however, should not be used since the purpose of the project is for you to discover vulnerabilities by yourself with your knowledge and skill as CS166 students. Specific tools such as Firebug or the "Inspect Element" tool in Chrome and Firefox are perfectly fine. If you have any questions about other tools, please ask TAs before using them.

6.3 Burp Suite

Burp Suite can be a useful tool for this project, although you are not required to use it. Burp Suite is a tool for analyzing the the security of web applications. A feature especially useful for this project is the ability to view and intercept HTTP requests and responses. If a request gets intercepted, Burp Suite allows you to edit it before it gets sent off to the server. This feature is called Burp Proxy.

You can launch Burp Suite from a department machine using the `cs166_burpsuite` command. Burp Suite is written in Java. If you have any issues running our script (which uses `/usr/bin/java`), you can run the `.jar` file directly using the following command:

`java -jar /course/cs166/bin/burpsuite_free_v1.6.jar`.

The documentation for Burp Proxy can be found here: `https://support.portswigger.net/customer/portal/topics/720233-burp-proxy/articles`. You should first go through "Getting Started with Burp Proxy," and complete the prerequisites. In particular, be sure to configure your browser to work with the program.

7 Handing In

Your handin should consist of a `README` (text file) along with any code you used in completing this assignment. Your `README` should contain *thorough* descriptions of each your exploits and descriptions of how to perform each attack.

Your `README` should also explain (without implementation) how to fix each vulnerability. Try to come up with a fix that would have a low technical and economic cost if possible. For example, hiring an experienced web application consultant to do it for you would not be a good fix. You may also argue that a vulnerability has no viable fix. Be sure to justify such an assertion.

You can hand in by running `cs166_handin flag_cs166_1` from a directory containing these files. CS162 students should instead run `cs166_handin flag_cs162_1`.

7.1 Interactive Grading

In security, attacks are only taken seriously when one can demonstrate that their attack actually allows one to perform unauthorized tasks in a clear and convincing manner. To give you the opportunity to exercise

your security presentation skills, there will be interactive grading sessions after the Flag 1 deadline in which you will meet with a TA and demonstrate all of your exploits.

- Each student will be given 20 minutes to demonstrate all of their exploits on a clean instance of the website. You are expected to prepare in advance to be able to present all your exploits within this time frame, as we will only give you full credit for exploits presented within the 20 minute window.
- In your presentation, you will be asked to walk through each of your exploits in a way that convinces us that your exploit allows you to perform an action in the web application that you are not supposed to be able to do. You should also briefly explain how you would fix each vulnerability as documented in your `README`.
- You will be asked to demonstrate all of your exploits via a department machine that contains a folder with all of the files you submitted in your handin. You will not be able to load any files not submitted in your handin onto this computer, so make sure everything you need to present is in your handin. You may, however, refer to your `README` or any other notes as a memory aid during your presentation.
- The TA may ask you questions that your presentation raises, so make sure to account for this when determining the length of your presentation.

As a reminder, please come prepared! The better prepared you are, the better your presentation will be, and if you clearly demonstrate how each of your exploits work your TAs will be able to give you advice as to how to start fixing the vulnerabilities you found in Flag 2.

Please also come on time—arriving late to your interactive grading session will result in lost presentation time. Additionally, if you do not show up within the first 10 minutes of your scheduled slot, you may be asked to reschedule your presentation and receive a 20 point deduction.

We will send out information on how to sign up for interactive grading sessions after the Flag 1 deadline.

Part II

Fixes

8 Assignment

Once the first handin deadline has passed, you will be given access to a Github repository containing the code for the application you attacked.[2]

For each of the exploits that you have discovered, you should make changes to the application so that it is no longer vulnerable to attack. If you submitted fewer than the required number of exploits in the first handin, you should attempt to find more vulnerabilities—and submit exploits for them—in order to receive full credit.

- Your fixes should be complete; it should not be possible for an attacker to bypass your patch and exploit the same vulnerability.
- You must patch *every* instance of your vulnerability, wherever it appears throughout the application. (In some cases, this might feel redundant—if this is the case, consider how you might improve your code so that it fixes the application in all places, but doesn't require redundant code).
- Your fix must not cause any loss of functionality, unless the previous functionality allowed the user to do malicious things. That is, functionality that was *intended* to be there must not be reduced or removed. Please consult the TAs if you are unsure if a potential fix reduces intended functionality.

If you described more than 5 exploits in your first handin, feel free to pick any 5 to patch. If you identify new vulnerabilities based on your code analysis, you may choose to document and patch them as part of your handin. **You must document your changes in a `README` for this handin, or you will not receive any credit for your changes.**

8.1 CS162

CS162 students are required to turn in fixes for each of their 6 exploits. If you described more than 6 exploits in your first handin, feel free to pick any 6 to patch.

8.2 Extra Credit

You may patch additional vulnerabilities for extra credit. In Flag 2, each additional, working patch is worth 2 points. As a reminder, every patch must be documented in your `README` or you will not recieve any credit for your changes.

9 Hints, Tips, and Tricks

9.1 PHP-specific Security

For the vast majority of you, this project will be the first time you are analyzing and trying to fix an insecure PHP application. If you find a function and don't know what it does, feel free to look it up in the PHP documentation (`php.net`). To assist you in getting started, here are some useful PHP functions / classes related to security:

[2] If you declare that you are using late days on Flag 1, you will receive your website's code after all of your declared late days have elapsed.

- `http://www.php.net/manual/en/function.htmlspecialchars.php`
- `http://www.php.net/manual/en/function.htmlentities.php`
- `http://www.php.net/manual/en/class.pdo.php`
- `http://www.php.net/manual/en/function.basename.php`
- `http://www.php.net/manual/en/function.crypt.php`

10 Handing In

You will be given access to a Github repository containing a copy of the website's code. This repository will contain a number of branches, each named `fix_x` (e.g., `fix_1`, `fix_2`, etc). **Each fix should be committed to its own branch.** For example, in order to commit your patch for your first vulnerability, you can execute the following commands while inside of the Git repository:

```
$ git checkout fix_1
$ # make your fixes...
$ git commit -a # commit all changes
```

Your handin should contain this Git repository in its own subdirectory, along with a `README` file documenting the following:

- If you identified and patched any vulnerabilities which you did not discover as part of the first handin, please document them here (include a thorough description of the vulnerability along with any code you used and a description of how to perform the attack, just as in the first handin).
- For each vulnerability, what changes you made to the code.

Your `README` should not be included in the same folder as your Git repository. To make this clear, the directory from which you hand in should look like this (where `my-handin-directory` is the directory that you are running the `cs166_handin` command in):

```
/my-handin-directory
|_ README
|_ /source-code  <-- this is the folder containing your git repository
   |_ hostsite.sh
   |_ /webroot
      |_ ...
```

You should hand in your completed work with `cs166_handin flag_cs166_2`. CS162 students should instead run `cs166_handin flag_cs162_2`.

Part III

Bob's Router

In addition to the normal assignment, CS162 students are required to complete an extra problem, which is described here. For CS162 students, this problem is worth 20% of the credit for this assignment; the other two handins are each worth 40%.

11 Assignment

In this problem, you will explore how attacking a network can be a multi-step process. You will use your access to the FLAG Portal as a starting point from which to launch a further attack on another client of the FLAG Portal.

11.1 Setup

Bob has confidential information on his home computer that could take down Gru and the rest of his organization. You decide to see if you can man-in-the-middle (MitM) Bob's internet traffic and retrieve this information!

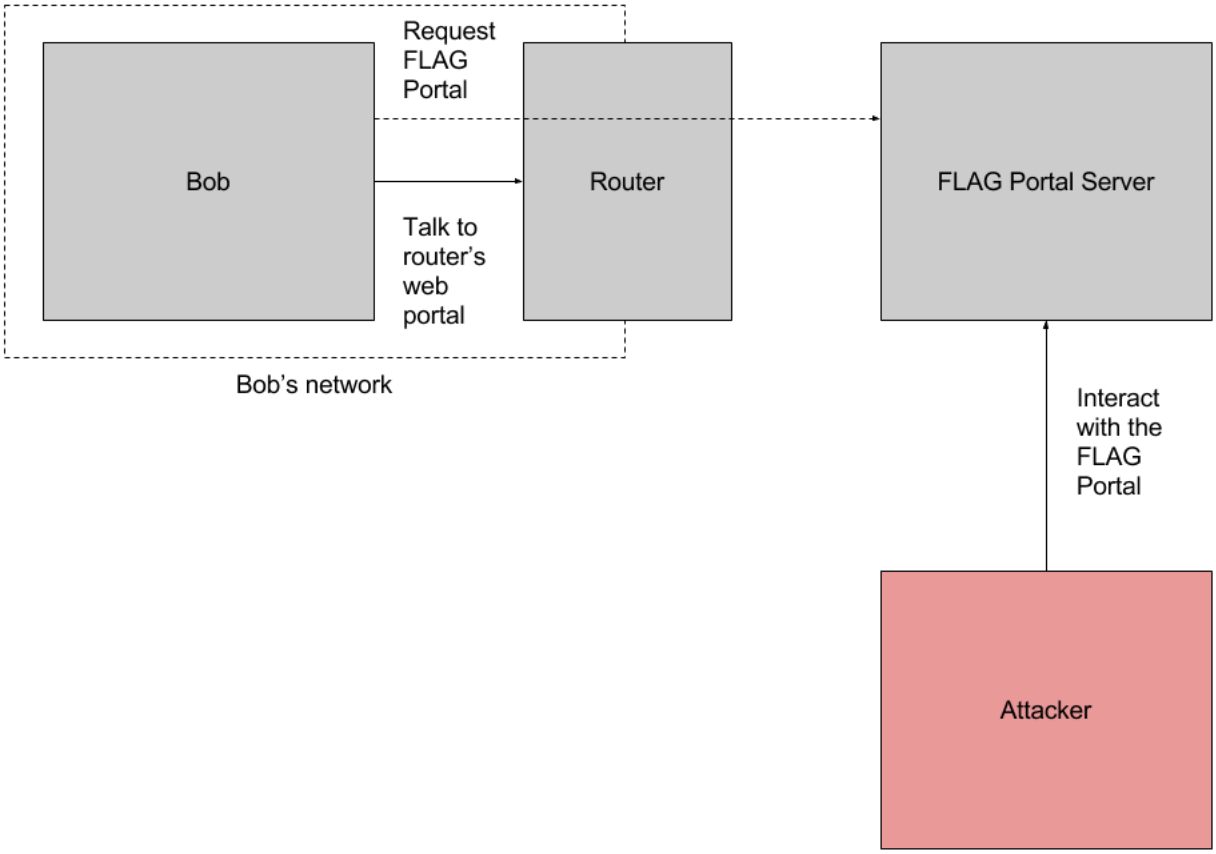
You know that Bob's home router is old and undoubtedly has lots of vulnerabilities (for example, the router uses short numerical PINs instead of passwords), and a router would be the perfect place from which to launch your MitM attacks. However, his router doesn't accept any incoming connections—it only lets clients in the local network make outgoing connections. That means that in order to attack the router, you'll need to somehow get one of the computers on the local network to make the connections for you.

11.2 The Attack

Luckily for you, Bob is a little obsessed with Minion University, and has an odd habit of refreshing the FLAG Portal's landing page every 15 seconds. Thus, your task is as follows:

- Use one of the attacks you developed for the first handin to somehow inject JavaScript into the FLAG Portal landing page. This allows you to execute JavaScript in Bob's browser.
- Use this JavaScript to launch a CSRF attack against Bob's router, which, like many routers, has a web portal running on port 80. The router's domain name on Bob's local network is `router.local`. Bob can use this domain name to access the router in his browser, but to machines outside of Bob's local network, this IP address will not mean anything.
- Discover as much as you can about the router and its attack surface. Eventually, you should be able to log into the router's web portal.
- Once you've logged into the router's web portal, poke around for further vulnerabilities. From here, you should be able to get remote code execution (RCE) on the router itself (which is just a Linux machine).
- Once you have RCE on the router, find the flag. This is what will prove to us that you have successfully hacked the router. What a "flag" is will make sense once you have RCE and can poke around :)

To make this clear, here's a diagram of the various network connections and machines involved:



11.3 Extra Credit

For 10% extra credit (making this problem worth 30% as opposed to 20%), you can additionally implement a reverse shell. A reverse shell is a program that uses your access to the router to run commands on the router, send the stdin of the local process to the process running on the router, and get the stdout and stderr back. Think of it like SSH for hacked machines.

In particular, to get all of the extra credit, you should write a program that:

- Can be run from any internet-connected machine (that is, it shouldn't need to be run on any particular computer to work properly).
- Allows the user to run arbitrary commands on the router.
- While commands are running, copies the stdin of the shell program to the stdin of the program running on the router, and copies the stdout of the program on the router back to the stdout of the shell, and similarly for stderr. This should happen in real-time so that interactive programs work properly (that is, it should not simply collect all of the stdout and stderr and send them back after the process exits).

11.4 Resetting Bob

If you think Bob is stuck and isn't refreshing your FLAG Portal anymore, you can run the `cs166_reset_bob` command on department machines. This will restart Bob's internet browser and will resume his refreshing attempts on the FLAG Portal.

If you think that you have broken Bob or your router to the point where you think you need a hard reset, please email the HTA list and we'll create a new router for you.

12 Handing In

Your handin should include a `README.pdf` which documents the following:

- The value of the flag that you found.
- A *detailed* account of the steps that you took in order to find the flag. Note the emphasis on "detailed"—don't be surprised if you end up writing a few pages. :)

Due to the length, please format your `README.pdf` as a PDF (hence the `.pdf` extension). Additionally, you should submit any code, files, etc, that you used in carrying out your attack.

You can hand in by running `cs166_handin flag_bob`.

12.1 Extra Credit

If you completed the extra credit, then you should include in your handin a subdirectory containing the source code of your reverse shell and any related code, and also a `README` which documents how your reverse shell works. The TAs will arrange a time for you to meet and demonstrate your reverse shell.