Flag Project

First handin due: 11:59 pm, Sunday February 25 (hard deadline)
Second handin due: 11:59 pm, Thursday March 8
Bob handin due: 11:59 pm, Thursday March 15
Note: No late days permitted for first handin, unless extension explicitly granted by instructor.

Contents

1 Introduction

Spectre 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.

2 Running the Web App

Each student gets their own instance of the web site to attack. Your copy is CURRENTLY running at an IP address which will be emailed you. A password is required to access the web site (this is not the same as the login scheme that the application itself implements, and is secure). That password will also be emailed to you along with your IP address. The username is "user".

3 First Handin: Vulnerabilities, Exploits, and Remediation

Due: 11:59pm, Sunday February 25

You should discover **5 distinct vulnerabilities**, 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.

For each vulnerability that you discover, you should explain (without implementation) how to fix the 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.

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.

- 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.

CS162

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

4 Second Handin: Fixes

Due: 11:59pm, Thursday March 8

Once the first handin deadline has passed, you will be given the code for the application you attacked. 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.

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.**

CS162

CS162 students are required to turn in fixes for each of their 6 exploits.

5 Grading/Extra Credit

This assignment will be graded out of 100 points, with each handin contributing 50 points to the total. If you submit an extra working exploit (that has a good description of the fix) before the first handin, you will be given 3 extra credit points. If you submit an extra patch (that is documented in the `README`) before the second handin, you will be given 2 extra credit points. These point values apply to both CS166 and CS162. The maximum score you can receive is 115/100.

Note that since the code for the website will be released after the first handin deadline, **THE FIRST HANDIN DEADLINE IS A HARD DEADLINE. We will not accept ANY late submissions. You may not use late days on the first handin.**

6 Handing In

6.1 First Handin

For this submission, you must submit a single text file.

You should put together a thorough description of your exploits along with any code you used and a description of how to perform the attack. Document your exploits in a `README` and hand it in using `cs166_handin flag_cs166_1`. A TA will evaluate your handin and tell you if the issues you've identified are sufficiently distinct and well-documented.

CS162

CS162 students should hand in using `cs162_handin flag_cs162_1`.

6.2 Second Handin

You will be given a Git repository with a copy of the website's code. This git 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 cd'd into 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.

Note: Do not put the `README` inside the Git repository, or Git will not be happy with you.

You should hand in your completed work with `cs166_handin flag_cs166_2`.

CS162

CS162 students should hand in using `cs162_handin flag_cs162_2`.

7 Hints, Tips, and Tricks

7.1 Accounts

Each student of the course has a login for FLAG. Your FLAG username is your CS username, and your FLAG password is iam<username>. E.g., if your CS username is alice, your FLAG username is alice and your FLAG password is iamalice.

Remember each of you have your own copy of the app, so feel free to log in as each other or yourself. Each member of the staff also has a login with their cs login, but the passwords are not known to you. They are, however, poorly chosen passwords. 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!).

7.2 Resetting

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.

7.3 Assumptions

Here are some hints:

- 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. Basically, you can send an email 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.

7.4 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 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. It may help, as with many puzzles, to take a break or go to sleep to let your idle mind think more creatively about the problem.

7.5 PHP-specific security knowledge

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:

- `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`

8 Tools and Rules

8.1 Tools

Ask the TAs before moving forward with any broad external tools. Authoring tools such as 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.

8.2 Burp Suite

Burp Suite can be a useful tool for this project, although it is not required that you 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.

To launch Burp Suite, run `/course/cs166/bin/burpsuite`, or just `burpsuite` if `/course/cs166/bin` is in your `PATH`. 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: `java -jar /course/cs166/bin/burpsuite_free_v1.6.jar`.

Here is the link to the documentation for Burp Proxy: `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.

9 Collaboration

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 OK and what would be unfair help, or even damage, to other student's work. Direct all questions to TAs via Piazza or at hours.

10 CS162 Extra Problem

In addition to the normal assignment, CS162 students are required to complete an extra problem, which is described here.

This problem is worth 20% of the credit for this assignment; the other two handins are each worth 40%.

10.1 Introduction

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.

10.2 Setup

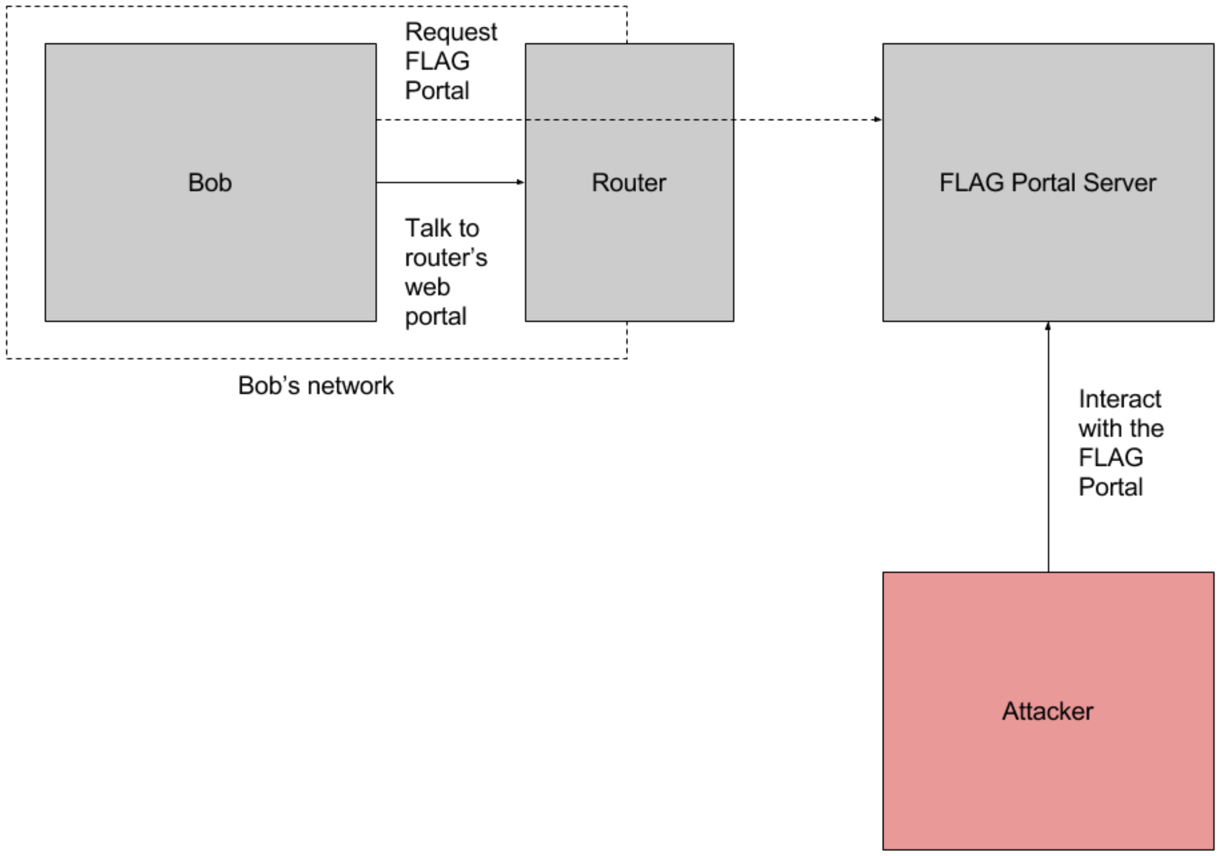
Bob, the second in command at Spectre, has confidential information on his home computer that could take down the Spectre organization. You decide to see if you can man-in-the-middle (MitM) Bob's internet traffic and retrieve this information!

You know that Bobs 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 doesnt accept any incoming connections - it only lets clients in the local network make outgoing connections. That means that in order to attack the router, youll need to somehow get one of the computers on the local network to make the connections for you.

Luckily for you, Bob is a little obsessed with Spectre University, and has an odd habit of refreshing the FLAG Portals 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. Now you have JavaScript executing in Bobs browser.
- Use this JavaScript to launch a CSRF attack against Bobs router, which, like many routers, has a web portal running on port 80. The routers IP on Bobs local network has been sent to you by email, which means that Bob can use this IP address to access the router, but to machines outside of Bobs 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 routers web portal.
- Once youve logged into the routers 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, heres a diagram of the various network connections and machines involved:



10.3 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 shouldnt 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).

10.4 Handing In

Due: 11:59 pm, Thursday March 15

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. And it should be detailed - dont 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) for your graders sanity :)

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`.

10.4.1 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.