Project 3

CS 628/628A: Computer System Security

2016-2017 -- Semester II

Computer Science and Engineering Department

Indian Institute of Technology Kanpur

Due Date: This project has only a single part, and it is due on April 8, 2017 at 11:45 pm

Acknowledgement: This project is based on a homework used at MIT for MIT-6.858 course in 2014

Introduction

This lab will introduce you to browser-based attacks, as well as to how one might go about preventing them.

When working on the exercises, you may find the following hints and tools useful:

- Firefox Developer Tools which can be accessed by pressing Ctrl-Shift-I in the browser.
  - The Console tab contains the Javascript console, which lets you see which exceptions are being thrown and why. You can print debug information to the console using console.log() function.
  - The Inspector and Element Picker (arrow icon on the top-left in the developer toolbox) let you peek at the structure of the page and the properties and methods of each node it contains.
  - The Network tab lets you see the requests sent by the browser and the server responses. These include cookies, HTTP headers and form data.

In Firefox 44, these tabs should be enabled by default in the developer toolbox. If you don't see them, adjust the 'Default Firefox Developer Tools' and 'Available Toolbox Buttons' preferences in the developer toolbox options.

- You may need to use CSS to make your attacks invisible to the user. You should know what basic syntax like `<style>.warning{display:none}</style>` means, and you should feel free to use stealthy attributes like `style="display: none; visibility: hidden; height: 0; width: 0; position: absolute"` in the HTML of your attacks. Beware that frames and images may behave strangely with `display: none`, so you might want to use `visibility: hidden` instead. For instance, to create a hidden `iframe`, try `<iframe style="visibility: hidden" ...>`.

- If you need to encode certain special characters in URL parameters, such as newlines, percents, or ampersands, take a look at encodeURIComponent and decodeURIComponent.

Beware of Race Conditions: Depending on how you write your code, all four of these attacks could potentially have race conditions. Attacks that fail on the grader's browser during grading will receive less than full credit. To ensure that you receive full credit, you should wait after making an outbound network request rather than assuming that the request will be sent immediately. You may find the `load` event on the `iframe` element helpful.

Network setup

For this lab, you will be crafting attacks in your web browser that exploit vulnerabilities in the zoobar web application. To ensure that your exploits work on our machines when we grade your project, we need to agree on the URL that refers to the zoobar web site. For the purposes of this lab, your zoobar web site must be running on http://localhost:8080/. If you have been using your VM's IP address, such as http://192.168.177.128:8080/, it will not work in this lab.

Use VMware, and we will use ssh's port forwarding feature to expose your VM's port 8080 as http://localhost:8080/. First find your VM IP address. You can do this by going to your VM and typing ifconfig. (This is the same IP address you have been using for past labs.) Then configure SSH port forwarding as follows (which depends on your SSH client):

> For Mac and Linux users: open a terminal *on your machine* (not in your VM) and run
>
> ```
> $ ssh -L localhost:8080:localhost:8080 httpd@VM-IP-ADDRESS
> httpd@VM-IP-ADDRESS's password:
> ```
>
> For Windows users, this should be an option in your SSH client. In PuTTY, follow these instructions. Use 8080 for the source port and localhost:8080 for the remote port.

The forward will remain in effect as long as the SSH connection is open.

Setting up the web server

Download the provided .zip file, lab3.zip from course site project 3. Note that this lab 's source code is based on the initial web server from lab 1. It does not include privilege separation or Python profiles.

Now you can start the zookws web server, as follows.

```
httpd@vm-628:~$ ./zookld
```

Open your browser and go to the URL http://localhost:8080/. You should see the zoobar web application. If you don't, go back and double-check your steps. If you cannot get the web server to work, get in touch with course TAs before proceeding further.

Crafting attacks

You will craft a series of attacks against the zoobar web site you have been working on in previous labs. These attacks exploit vulnerabilities in the web application's design and implementation. Each attack presents a distinct scenario with unique goals and constraints, although in some cases you may be able to re-use parts of your code.

We will run your attacks after wiping clean the database of registered users (except the user named
"attacker"), so do not assume the presence of any other users in your submitted attacks.

You can run our tests with make check; this will execute your attacks against your server, and tell you whether your exploits seem to be working correctly or not. As in previous labs, keep in mind that the checks performed by make check are not exhaustive, especially with respect to race conditions.

Exercises 1, 3, and 4, as well as the challenge exercise, require that the displayed site look a certain way. The make check script is not quite smart enough to compare how the site looks like with and without your attack, so you will need to do that comparison yourself (and so will we, during grading). When make check runs, it generates reference images for what the attack page is *supposed* to look like (answer- XX.ref.png) and what your attack page actually shows (answer-XX.png), and places them in the lab3-tests/ directory. Make sure that your answer-XX.png screenshots look like the reference images in answer-XX.ref.png.

To view these images from lab3-tests/, either copy them to your local machine, or run python -m SimpleHTTPServer 8080 and view the images by visiting http://localhost:8080/lab3-tests/. *Note that SimpleHTTPServer caches responses,* so you should kill and restart it after a make check run.

We will grade your attacks with default settings using version 44 of Mozilla Firefox on Ubuntu 14.04 browser at the time the project is due. We chose this browser for grading because it is widely available and can run on a variety of operating systems. There are subtle quirks in the way HTML and JavaScript are handled by different browsers, and some attacks that work or do not work in Internet Explorer or Chrome (for example) may not work in Firefox. In particular, you should use the Mozilla way of adding listeners to events. We recommend that you test your code on Firefox before you submit, to ensure that you will receive credit for your work.

For exercises 1 and 3, you will need a mechanism to log the information captured by your client-side JavaScript. A simple way of doing this is to use the SimpleHTTPServer module in python which logs all the incoming requests. To start the server, run the following command in your VM

```
script -a -c "python -m SimpleHTTPServer 12345" /tmp/requests.log
```

This will start a listening server on your VM on port 12345 and log all requests to the file /tmp/requests.log. Your javascript code should send the captured information to this server as an HTTP GET request in the following format

http://<IP of your VM>:12345?payload=<payload>&roll=<your roll no>&random=<random number>&timestamp=<current time stamp>

The random number is required to prevent the browser from caching the responses.
Before submitting, you should change the VM's IP in your exploit to 192.168.56.4. This IP will be used by the TAs while grading.

By accessing the VM using its IP instead of port forwarding, we ensure that the browser does not treat the two servers, zoobar (localhost:8080) and SimpleHTTPServer (192.168.56.4:12345) as the same origin. This lets us simulate a cross-domain request using a single server. If the listening server is accessed as localhost:12345, the browser will automatically send the cookies with the request as an HTTP header.

> **Exercise 1: Auth Token Theft.** Construct an attack that will steal a victim's auth token from the cookie of the zoobar site and then logs the user out from the site when the victim's browser opens a URL of your choosing. (You do not need to do anything with the auth token after stealing it, for the purposes of this exercise, although in practice an attacker could use the token to impersonate the victim, and issue requests as if they came from the victim.)(Wait for your ctf ;))

Your solution is a URL starting with
http://localhost:8080/zoobar/index.cgi/users?
The grader will already be logged in to the zoobar site before loading your URL.
Your goal is to steal the document cookie and send it as the payload to the listening server on the VM and then log out the user from the site. Remember the grader script takes screenshot after some time, so your payload should not log the user out before the grading script takes the screenshot. The grader should see the same page as if without being logged out from the site.
Except for the browser address bar (which can be different), the grader should see a
page that looks *exactly* as it normally does when the grader visits
http://localhost:8080/zoobar/index.cgi/users. No changes to the site appearance or extraneous text should be visible. Avoiding the *red warning text* is an
important part of this attack. (It's ok if the page looks weird briefly before correcting
itself.)

Hint: You will need to find a cross-site scripting vulnerability in the /zoobar/index.cgi/users page, and then use it to inject Javascript code into the browser. What input parameters from the HTTP request does the resulting /zoobar/index.cgi/users page include? Which of them are not properly escaped?
Hint: To steal the cookie, read about how cookies are accessed from Javascript.

Please write your attack URL in a file named answer-1.txt. Your URL should be the only thing on the first line of the file.

For exercise 1, you will want the server to reflect back certain character strings to the victim's browser. However, the HTTP server performs URL decoding on your request before passing it on to the zoobar code. Thus, you'll need to make sure that your attack code is URL-encoded. For example, use + instead of

space and %2b instead of +. You can also use quoting functions in the python urllib module or the JavaScript encodeURIComponent function to URL encode strings.

Exercise 2: Cross-Site Request Forgery. Construct an attack that transfers zoobars from a victim to 2 different attackers, namely “attacker” and “attacker1”, when the victim's browser opens an HTML document that you construct. Do not exploit cross-site scripting vulnerabilities (where the server reflects back attack code), such as the one involved in exercise 1 above, or logic bugs in transfer.py that you fixed in lab 2.

Your solution is a short HTML document named answer-2.html that the grader will open using the web browser.
Be sure that you do **not** load the answer-2.html file from http://localhost:8080/..., because that would place it in the same origin as the site being attacked, and therefore defeat the point of this exercise.
The grader (victim) will already be logged in to the zoobar site before loading your page.
Your goal is to transfer 1 zoobar from the grader's account to the "attacker" account and 9 zoobars to the “attacker1” account. The browser should be

redirected to http://moodle.cse.iitk.ac.in as soon as the transfer is complete (so fast the user might not notice).
The location bar of the browser should not contain the zoobar server's name or address at any point. This requirement is important, and makes the attack more challenging.

Hint: One way to construct the attack is to develop answer-2.html in small steps that incrementally meet all the requirements.
Hint: You might find the target attribute of the HTML form element useful in making your attack contained in a single page.

For exercise 2, you should test if your attack works by opening your answer-2.html file in your browser, and seeing if you achieve the desired result while meeting the requirements for the attack.

For exercise 2, you will need to synthesize an HTTP POST request from your HTML page. To do so, consider creating an HTML form whose action attribute points to .../index.cgi/transfer, and which contains <input> fields with the necessary names and values. Look at the source of the HTML that's generated by index.cgi/transfer to get an idea of what this form should look like. You can submit a form by using JavaScript to invoke the click method on the submit button, or the submit method on the form itself. Try to develop the entire attack using only a single form in your html page.

Exercise 3: Side Channels and Phishing. Construct an attack that will steal a victim's zoobars, if the user is already logged in (using the attack from exercise 2), or will ask the victim for their username and password, if they are not logged in. The attack scenario is that the victim opens an HTML document that you constructed.

Your solution is an HTML document named answer-3.html that the grader will open using the web browser.
As with the previous exercise, be sure that you do **not** load the answer-3.html file from
http://localhost:8080/.
The grader will run the code once while logged in to the zoobar site before loading your
page.
The grader will run the code a second time while *not* logged in to the zoobar site before
loading your page.
When the browser loads your document, the document should sniff out whether the use is logged into the zoobar site:
If the user is not logged in, present an HTML document visibly identical to the
zoobar login page, by copying the HTML from the real zoobar login page (this
should be self-contained in the HTML file you turn in):
When the "Log in" button is pressed, send the username and password
(separated by a comma) as the payload to the listening server.
Once the payload is sent, log the user into the real zoobar website (the
hostname should change to localhost:8080).
The behavior for the *Register* button is left unspecified.
If the user is logged in, then forward to the attack from exercise 2.

Hint: The same-origin policy generally does not allow your attack page to access the contents of pages from another domain. What types of files can be loaded by your attack page from another domain? Does the zoobar web application have any files of that type? How can you infer whether the user is logged in or not, based on this?
Hint: develop your attack in steps, incrementally addressing all of the above requirements.

Exercise 4: Profile Worm. Create a worm that will transfer 1 zoobar from the victim to the attacker, and spread to the victim's profile increasing the number of zoobars to be transferred gets increased by 1 everytime, when the victim views the profile of another infected user. The scenario is that the first victim views the attacker's profile and the attacker gets 1 zoobar from the victim and then the worm gets copied into the victims' profile such that whoever visits the victims' profile now gets a worm which sends 2 zoobars to the attacker and the worm spreads increasing the number of zoobars to be transferred by 1 everytime, and the worm spreads onward from there.

Your solution is a profile that, when viewed, transfers 1 zoobar from the current user to a user called "attacker" (that's the actual username) and replaces the profile of the current user with itself (i.e., the attack profile code) increasing the number of zoobars to be transferred by 1 everytime. So the attack scenario is as follows: "victim1" visits "attacker" profile and 1 zoobar is sent from "victim1" to "attacker" and now the profile code for "victim1" becomes such that whoever visits "victim1" 2 zoobars are transferred from their account to "attacker". So now when "victim2" visits "victim1" 2 zoobars are transferred from "victim2" to "attacker" and "victim2" gets a profile such that whoever visits his profile 3 zoobars are transferred from their account to the "attacker". So when "victim3" visits "victim2" 3 zoobars are transferred from "victim3" to "attacker" and now whoever visits "victim3" 4 zoobars are transferred from their account to "attacker". Now when "victim4" visits "victim3" 4 zoobars are transferred from "victim4" to "attacker".
Your malicious profile should display the message **I Love CS628A!** when viewed, as if that was the entirety of the viewed profile.
To grade your attack, we will cut and paste the submitted profile code into the profile of the "attacker" user, and view that profile using the grader's account. We will then view the grader's profile with more accounts, checking for both the zoobar transfer and the replication of profile code with the correct number of zoobars to be transferred.
The transfer and replication should be reasonably fast (under 15 seconds). During that time, the grader will not click anywhere.
During the transfer and replication process, the browser's location bar should remain at http://localhost:8080/zoobar/index.cgi/users?user=*username*, where *username* is the user whose profile is being viewed. The visitor should not see any extra graphical user interface elements (e.g., frames), and the user whose profile is being viewed should appear to have 10 zoobars, and no transfer log entries. These requirements make the attack harder to spot for a user, and thus more realistic, but they make the attack also harder to pull off.
You will not be graded on the corner case where the user viewing the profile has no zoobars to send.

Hint: Start by writing a simple HTML profile and uploading it, just to familiarize yourself with how an HTML profile works in zoobar. Next, develop the solution profile in small steps (e.g., first arrange that the malicious profile

code transfers 1 zoobar to the attacker, and then make it spread to the visitor's profile by increasing the amount of zoobars to be transferred).
Hint: This MySpace vulnerability may provide some inspiration.

Please write your profile in a file named answer-4.txt.

For exercise 4, you may need to create an iframe and access data inside of it. You can use the DOM
methods document.createElement and document.body.appendChild to do so. Getting access to form fields in an iframe differs by browser, and only works for frames from the domain (according to the same-origin policy). In Firefox, you can do iframe.contentDocument.forms[0].zoobars.value = 1;. Another approach may be to use XMLHttpRequest instead of an iframe.

Challenge: Password Theft. Create an attack that will steal the victim's username and password, even if the victim is diligent about entering their password only when the URL address bar shows http://localhost:8080/zoobar/index.cgi/login.

Your solution is a short HTML document named answer-chal.html that the grader will open using the web browser.
The grader will not be logged in to the zoobar web site before loading your page. Upon loading your document, the browser should immediately be redirected to http://localhost:8080/zoobar/index.cgi/login. The grader will enter a username and password, and press the "Log in" button.
When the "Log in" button is pressed, send the username and password (separated by a comma) as payload to the listening server.
The login form should appear perfectly normal to the user. No extraneous text (e.g., warnings) should be visible, and assuming the username and password are correct, the login should proceed the same way it always does.

For this final attack, you may find that using alert() to test for script injection does not work; Firefox blocks it when it's causing an infinite loop of dialog boxes. Try other ways to probe whether your code is running, such as document.loginform.login_username.value=42.

Deliverables

Make sure you have the following files: answer-1.txt, answer-2.html, answer-3.html, answer-4.txt, and if you are doing the challenge, answer-chal.html, containing each of your attacks. Feel free to include any comments about your solutions in the answers.txt file (we would appreciate any feedback you may have on this assignment).

Run **make submit**. The resulting lab3-handin.tar.gz will be graded. You're done!

Acknowledgments

Thanks to Stanford's CS155 course staff for the original version of this assignment.