6.858 Fall 2014 Lab 5: Browser security

Handed out: Lecture 14
Due: Two days after Lecture 17 (5:00pm)

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's JavaScript console and DOM inspector, both accessible from the Tools menu. The JavaScript console lets you see which exceptions are being thrown and why. The DOM Inspector lets you peek at the structure of the page and the properties and methods of each node it contains. (If the DOM Inspector isn't installed, make sure it's selected when you install Mozilla Firefox.) You might also want to try Firebug. In Firefox 16, these tools are accessible from the Web Developer menu item under the Tools menu.
- 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" ...>`.
- You can use the LiveHTTPHeaders browser extension to see the cookies and form data that a browser sends.
- 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 lab, 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.

If you are using KVM or VirtualBox, the instructions we provided in lab 1 already ensure that port 8080 on `localhost` is forwarded to port 8080 in the virtual machine. If you are using VMware, 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: 6858
```

- 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 affect as long as the SSH connection is open.

Setting up the web server

Before you begin working on these exercises, please use Git to commit your Lab 3 solutions, fetch the latest version of the course repository, and then create a local branch called `lab5` based on our lab5 branch, `origin/lab5`. Do *not* merge your lab 2 and 3 solutions into lab 5. Here are the shell commands:

```
httpd@vm-6858:~$ cd lab
httpd@vm-6858:~/lab$ git commit -am 'my solution to lab3'
[lab3 c54dd4d] my solution to lab3
 1 files changed, 1 insertions(+), 0 deletions(-)
httpd@vm-6858:~/lab$ git pull
Already up-to-date.
httpd@vm-6858:~/lab$ git checkout -b lab5 origin/lab5
Branch lab5 set up to track remote branch lab5 from origin.
Switched to a new branch 'lab5'
httpd@vm-6858:~/lab$ make
...
```

For those using the provided .zip files, please download lab5.zip from the MIT OpenCourseWare site.

Note that lab 5'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-6858:~$ ./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 staff 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 `lab5-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 `lab5-tests/`, either copy them to your local machine, or run `python -m SimpleHTTPServer 8080` and view the images by visiting `http://localhost:8080/lab5-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 the current version of Mozilla Firefox on Ubuntu 12.04 (as installed on, e.g., the Athena workstations) 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 server-side script to automatically email information captured by your client-side JavaScript code to the TAs for grading. We have provided this script for you. Please review the instructions at http://css.csail.mit.edu/6.858/2014/labs/sendmail.php and use that URL in your attack scripts to send emails. You may send as many emails as you like while working on the project, but please do not attack or abuse the email script.

> **Exercise 1: Cookie Theft.** Construct an attack that will steal a victim's cookie for the `zoobar` site when the victim's browser opens a URL of your choosing. (You do not need to do anything with the victim's cookie after stealing it, for the purposes of this exercise, although in practice an attacker could use the cookie to impersonate the victim, and issue requests as if they came from the victim.)
>
> - 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 email it to yourself using the email script.
> - 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 `+`. Here is a URL encoding reference and a handy conversion tool. 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 the attacker, 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 3.

- 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 10 zoobars from the grader's account to the "attacker" account.
- The browser should be redirected to `http://css.csail.mit.edu/6.858/2014/` 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.

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 user

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) using the email script.
    - Once the email 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, when the victim views the profile of another infected user. The scenario is that the first victim views the attacker's profile, 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).
- Your malicious profile should display the message **Scanning for viruses...** 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.
- 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).
- 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) using the email script.
> - 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 `lab5-handin.tar.gz` will be graded. You're done!

Acknowledgments

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

6.858 Computer Systems Security

Fall 2014