Project 2: Web Security Pitfalls

This project is due on **Thursday, October 28** at **5 p.m.** and counts for 9% of your course grade. Late submissions will be penalized by 10% plus an additional 10% every 12 hours until they're received. The professor may grant individual extensions, but only under truly extraordinary circumstances. As always, *we recommend that you begin early.*

This is a group project; you will work in **groups of two** and submit one project per group. If you wish, you may work with a different partner than you worked with for the previous project. We recommend that you and your partner work through the simple attacks together to make sure that you both understand the site infrastructure and the attacks. You will not get the intended benefits from the project if you divide it up and work separately.

The code your group submits must be entirely your own work. You are free to discuss the project with other members of the class, but you may not look at any part of someone else's solution. You may consult published resources, provided that you appropriately cite them (with program comments), as you would in an academic paper.

We will answer questions about the project in the Friday discussion section and the CTools forum. As per the collaboration policy, do not post any code from your project to the forum. Please check the forum regularly for updates and clarifications.

Update 10/21: Fixed minor typos and Python version

1 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 naïve defenses.
- Gain experience with web architecture and with HTML, JavaScript, and SQL programming.

2 Target Website

A startup called 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 EECS 398, so the investors have hired you to perform a security evaluation before it goes live.

The site is written using the Python-based Django[1] web framework. Although Django has built-in mechanisms that help guard against many 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 SQLite[2] database. Every page includes a sidebar `div` that displays either the login form, or, if a user has logged in, the user's search history.

Before being granted access to the source code, you reverse engineered the site and determined that it replies to three main URLs: `/`, `/search`, and `/login`. *[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 simply displays a search form. When submitted, the form issues a `GET` request for the `/search` page, sending the search string as the query parameter "`q`".

Search results (`/search`) The search page prints the search string (supplied in the "`q`" query parameter) to tell users what they searched for and then displays search results. *[Because actual search is not relevant to this project, the search results section is always empty.]*

Login handler (`/login`) The login handler returns an HTML snippet containing new content for the login/search history sidebar. It is not intended to be directly viewed as a standalone page. The login handler accepts up to four arguments via `POST`:

`user` – The name of the user.

`password` – The user's password in plaintext. *[The password is neither sent nor stored securely, but none of the attacks you implement should depend on this behavior. Never use an important password to test an insecure site!]*

`login_action` – One of "`login`", "`register`", or "reset". The `register` action inserts `user` and `password` into the database of users, unless a user with that username already exists. The `login` action attempts to log in with the provided username and password. The `reset` action attempts to reset the password for the given user using the provided answer to the secret question, "What is your favorite college football team?"

`secretanswer` – The answer to the secret question if the `reset` action is being used.

Whether or not a user is logged in is tracked by a login cookie. Determining the format of this cookie is left as an exercise to the reader. *[The cookie is definitely not secure, but manipulating or forging it is* ***not*** *part of this project. Treat it as an opaque string that attackers want to steal.]*

[1] http://www.djangoproject.com/

[2] http://www.sqlite.org/

3 Tasks and Deliverables

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.

3.1 READ THIS FIRST

This project asks you to develop attacks and test them against your own private instance of the target site. Attempting the same kinds of attacks against other websites without authorization may result in *fines, expulsion, and jail time*. **You must not attack any website without authorization!** This includes attempting to interfere with another student's instance of the target site. If you have any questions about this policy, especially as it applies to this project, contact the course staff before proceeding. Per the course ethics policy, *you must respect the privacy and property rights of others at all times,* ***or else you will fail the course.*** See the "Ethics, Law, and University Policies" section on the course website at http://www.eecs.umich.edu/courses/eecs398/ for further information.

3.2 SQL Injection

Your first goal is to demonstrate SQL injection attacks that log you in as a specific user (without knowing the password) and allow you to view the user's search history. Specifically, for each of the following defenses, create a web page with the specified filename that, when viewed in the browser, logs you in as the user "`jhalderm`":

0. No defenses.
   [`sql0-nodefense.html`]

1. *Extra credit:* The server escapes single quotes (`'`) in the input by replacing them with two single quotes.
   [`sql1-escape.html`]

Guidelines and Hints

The defense is selected by the "`sqlfilter`" parameter (sent via `POST`) to `/login`. The default is `0`, the lowest setting. Each page you submit must set this parameter to the defense it is targeting. To make it easier to manually test the site easier, the developers have included a dropdown box in the login sidebar that controls this parameter.

To see the SQL query you are attacking, look in `server/search398/simplesearch/views.py`.

Be sure that your attacks are logging you in as the correct user, and that you remain logged in after refreshing the page. Make sure they work even if the victim user is neither the most nor the least recently created user!

The SQL injection defense does not properly handle passwords that would be modified by the filter. Avoid setting passwords containing single quotes while working on this part.

3.3 Cross-site Request Forgery (CSRF)

Your next goal is to demonstrate CSRF 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. Specifically, for each of the following defenses, create a web page with the specified filename that, when viewed by a victim, causes the victim to be logged in to the account "`honey`" with the password "`l33th4x`": *[Be sure to create this account when you are testing!]*

0. No defenses.
   [`csrf0-nodefense.html`]

1. Simple validation token: The server maintains a global counter that is incremented and placed in a hidden field in the login form called "`csrfcounter`"; the login attempt is accepted if the submitted counter value is in the interval $[currentcounter - 10, currentcounter)$.

   Apparently, the site developer expected light, constant traffic to the site, so you may expect this as well; you may ***not*** *assume that the counter starts at 1. You are allowed to exploit the cross-site scripting vulnerability in the* `/search` *page to accomplish your goal.*

   [`csrf1-token.html`]

For the following defense, create a web page with the specified filename that, when viewed by a victim, will cause the victim to be *logged out* of the target site if the victim is logged in:

2. `Referer` validation: The server rejects login requests if the HTTP `Referer` header is for a site other than the server's. However, the server accepts requests that have no `Referer` at all.

   The provided implementation of `Referer` *validation requires the server to be hosted on port 8000 and accessed directly at http://127.0.0.1:8000. If your setup is different, edit the Referer string in* `_validate_referer` *in the file* `views.py`*.*

   [`csrf2-referer.html`]

Guidelines and Hints

The web pages you submit must be self-contained (i.e., HTML files only), but they may contain embedded CSS and JavaScript. Don't use unnecessary JavaScript when simple HTML would do.

Your solutions should not display evidence of an attack (e.g., pages from the target site). Instead, the page should be blank. It is OK if the page is briefly not blank before correcting itself.

The defense is selected by the "`csrfdefense`" query parameter to all the URLs (sent via `GET`, not `POST`). The default is `0`, the lowest setting. You may not attempt to subvert the mechanism for changing the level of defense in your attacks; *all requests for a particular defense* n *must set* `csrfdefense` *to* n*.*

For your convenience, the developers have included a dropdown in the sidebar that you can use to change the CSRF defense manually. If you use it, be sure to click the "Change CSRF Defense" button in the login sidebar to update the sidebar *before* logging in.

3.4 Cross-site Scripting (XSS)

Your final goal is to demonstrate XSS attacks against the search box, which does not properly filter search terms before echoing them to the results page.

Payload The payload (the code that the attack tries to execute) will be an extended password theft. If the victim is already logged in, the payload should steal the victim's login cookie, log the victim out, and continue the attack. If and when the victim is not logged in, the payload should do the following when the victim attempts to log in:

a. Steal the victim's username and password.
b. Modify the username and password that will be submitted to force a login failure.
c. Continue to attack after the failed login attempt (i.e., repeat (a) and (b) as long as the user stays on the page and keeps trying to log in).

If the victim gives up and tries to reset the password, the payload should steal the victim's answer to the secret question.

Defenses There are six levels of defense, starting with 0, which has no defenses, and ending in 5, which is the Django 1.0 XSS filter. Previously unknown attacks against the Django 1.0 XSS filter will result in extra credit. The code for the XSS filters is in `server/search398/simplesearch templatetags/xss_filters.py`. In each case, you should submit the simplest attack you can find that works against that defense; you must not simply attack the highest level and submit your solution for that level for every level. You must also use a different technique for each defense.

We may release additional levels of defense during the project as optional extra credit challenges. If we do, they will be announced on CTools. There is a hook in the provided code to support this (see `search398/simplesearch/templatetags/xss_filters.py`), so you will not need to download a new version of the server.

Guidelines and Hints As with the previous attacks, the server supports a "`xssdefense`" query parameter to `/search` to select the defense in use. The developers have again included a dropdown menu that you can use to select the XSS defense manually. Unlike the CSRF attack, you do not need to click a separate button to change the difficulty of the XSS attack; simply set the difficulty before clicking search.

Your submission for each level of defense will be a URL that, when clicked by the victim, executes the payload. The URL will be for the `/search` page in all cases, and you should simply omit the domain and start the URL with "/search". The URL must include the appropriate `difficulty` query parameter as the first query parameter. For each level of defense *n*, put your URL in a text file entitled "`xss`*n*`.txt`". The *payload* should be the same in all cases; attach a readable version of the payload in an additional file called "`xss-payload.js`". Note that your URLs may not depend on the existence of this file.

To steal a piece of data, send it to http://127.0.0.1:31337/stolen using a GET request and query parameters. Use the parameter "`user`" to send the stolen username, "`password`" to send the stolen password, "`secretanswer`" to send the answer to the secret question, "`cookie`" to send the stolen cookie, etc. Be careful not to send any actual sensitive information (e.g., your code) to this URL.

3.5 Writeup: Better Defenses

For each of the three attacks (SQL injection, CSRF, and XSS), write a paragraph about the techniques BUNGLE! should use to defend against that attack. Place these paragraphs in a file entitled "`writeup.txt`". If you find any additional security vulnerabilities in the site or have suggestions about how we might improve this project in the future, include them as well.

4 Mechanics

We have provided code for the target website, which is implemented in Python using the Django Web framework. You do not need to understand this code in order to complete the project, although you are free to look at it. We have left debugging messages turned on in the Django settings in case you run into trouble with the provided code.

4.1 Installing and running the server

Prerequisites You will need to install Django 1.0 to run the provided server. You can install it in the CAEN environment with the following command sequence:

```
$ mkdir ~/packages
$ cd ~/packages
$ wget http://www.djangoproject.com/download/1.0.4/tarball/
$ tar zxf Django-1.0.4.tar.gz
$ cd Django-1.0.4
$ python2.5 setup.py install --home $HOME
```

Server Setup Download the project code distribution from the course website and extract it. Then:

1. Set the Python library path:
   Run `setenv PYTHONPATH $HOME/lib/python` if you use the CAEN default C shell (your prompt will be "% "); or run `export PYTHONPATH=$HOME/lib/python` if you use the bash shell (recommended; your prompt will be "$ ").
2. Edit `settings.py` to configure the path where you have extracted the server.
3. Create the database by changing to the directory where you extracted the server and running: `python2.5 manage.py syncdb`. The database is created in a file in your home directory called `398p2.db`. You only need to perform this step once, unless you delete the database.

Running Whenever you want to start the target server, run: `python2.5 manage.py runserver` The server should print its URL. If you get an error about the port already being in use, you can specify a port after "`runserver`". To quit the server, just press control-C.

Connecting This project will be graded under Mozilla Firefox 3.6.9, which is available under CAEN Linux. If you are trying to work on the project remotely, make sure that you run Firefox *inside* the CAEN Linux VNC session. You can also use SSH port forwarding so that you can run Firefox locally, but explaining how to do so is outside the scope of this document.

4.2 General Guidelines and Resources

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.

We strongly recommend that you do not try to develop this project targeting a browser other than the Firefox 3.6 series. Cross-browser compatibility is one of the major headaches of web development. Furthermore, recent versions of Google Chrome and Internet Explorer include defenses against cross-site scripting that will interfere with your testing.

The Firebug JavaScript debugger (http://getfirebug.com/) will be a tremendous help for this project.

JavaScript is a relatively complicated programming language and has many confusing parts. In this project, focus on producing simple code that does what you need it to do without getting bogged down in JavaScript esoterica. For a JavaScript/DOM API reference, you might try http://www.w3schools.com/jsref/default.asp or *JavaScript: The Definitive Guide*, 5th Edition, by David Flanagan (2006). For a gentler introduction to the language, see: http://en.wikibooks.org/wiki/JavaScript/.

We encourage you to read the `ha.ckers.org` Cross-Site Scripting Cheat Sheet, available at http://ha.ckers.org/xss.html. It is an amazingly comprehensive directory of ways to bypass partial XSS filters. For SQL injection, you might start by consulting http://bobby-tables.com as well as http://ha.ckers.org/sqlinjection. In addition, there is searchable documentation for Django at http://docs.djangoproject.com/en/dev/ to help you decipher the provided code.

4.3 Submitting

This project is due **Thursday, October 28** at **5 p.m.** For submission, place your files in a directory named "`proj2`" and archive it as "`proj2.`*`member1uniqname`*`.`*`member2uniqname`*`.tar.gz`". (To help us process submissions easily, please put your uniqnames in alphabetical order). Attach the archive file to an email to `eecs398@umich.edu` with "`Proj2 Submission`" as the subject. You should receive a confirmation email within 15 minutes.