EECS 398 Project 2: Classic Web Vulnerabilities

Revision History

- 3.0 (October 27, 2009) – Revise CSRF attacks 1 and 2 to make them possible to complete within the constraints of the project. Clarify that CSRF attacks should not be visible – the user should see a blank page when visiting the attack. Fix the server and spec so that all four URLs respect the `csrfdefense` query parameter.
- 2.1 (October 21, 2009) – Change the format for submitting the SQL injection attack. Give out the correct command to set PYTHONPATH in the C shell. Add note about editing settings.py before running the server. Fix a bug in the server with Firefox 3.0.
- 2.0 (October 16, 2009) – Correct spec typos: 1) incorrect date on version 1.0 and 2) there aren't three sets of defenses against SQL injection. New code release: 1) do an AJAX /logout instead of redirecting to the search page, 2) remove unneeded "action" form field in login form, and 3) declare JavaScript variables with "var".
- 1.0 (October 14, 2009) – initial release.

1 Introduction

In this project, you will exploit three classic web vulnerabilities (cross-site scripting, cross-site request forgery, and SQL injection), both in their naïve forms and with several defenses. We provide an insecure website against which you will implement these attacks.

2 Ethics

This is an attack project. As a reminder, *you must not attack any other website, nor may you attempt to interfere with other students*! If you have any questions about this policy, especially as it applies to this project, you must contact the course staff before proceeding. If you do not abide by this policy, *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 Goals

1. Be able to identify common Web vulnerabilities.
2. Be able to demonstrate the severity of common Web vulnerabilities, even with naïve defenses in place.
3. Become familiar with Web architecture, JavaScript, etc.

4 Target Overview

The target website is a simple search engine written using the Python-based Django web framework (`http://www.djangoproject.com/`). Although Django has built-in mechanisms that guard against the security vulnerabilities you will exploit in this project, we have circumvented or ignored these mechanisms in several places.

In addition to search, the target website also accepts logins and tracks users' search history. It stores usernames, passwords, and search history in a SQLite (`http://www.sqlite.org/`) database. Every page includes a sidebar `div` that displays either the login form or the search history, depending on whether a user is currently logged in. It replies to three main URLs: `/`, `/search`, and `/login`. The function of each of these URLs is explained below. (You can imagine that you've spent some time reverse engineering the website to learn this. If you want an additional challenge, feel free to experiment with the target website instead of reading the remainder of this section.)

4.1 Main page (/)

The main page simply displays a search box. When submitted, the search form sends a GET request for the `/search` page, sending the search string as the query parameter "`q`".

4.2 Search results (/search)

The search page prints the search string (supplied in the "`q`" query parameter) to tell the user what he or she searched for and then displays search results. Because actual search is not relevant to this project, the search results section is always empty.

4.3 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 viewed as a standalone web page.

The login handler accepts up to four arguments via POST:

1. `user` – the name of the user.
2. `password` – the user's password in plaintext. NOTE: the password is neither sent nor stored securely, so don't use an important password! Furthermore, none of the attacks you are to implement should depend on this behavior.
3. `login_action` – one of "login", "register", or "reset". Register inserts the username and password into the database of users, unless a user with that username is in the database already. Login attempts to log in with the provided username and password. Reset attempts to reset the password of the given user using the provided answer to the secret question "What is your favorite college football team?".
4. `secretanswer` – 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 (hint: in Firefox, right-click the webpage and click "View Page Info" while logged in). (Note: the cookie format is definitely not secure. However, manipulating and/or forging the cookie is *not* part of this project; consider it as an opaque string that attackers want to steal.)

5 Tasks and deliverables

You will implement attacks for the three classic Web vulnerabilities against specific parts of the target website against varying levels of defenses.

5.1 SQL Injection

Your SQL injection attack will log you into a specific user's account without knowing his or her password, allowing you to view his or her search history. You will implement the attack against the following sets of defenses:

0. No defenses.

1. A filter that escapes single quotes in the input. In SQL, single quotes are escaped by replacing them with two single quotes. (This one is extra credit.)

The defense is controlled by the "`sqlfilter`" parameter (sent via POST) to the "`/login`" URL. The default is 0, the lowest setting. For your convenience, we have included a dropdown box in the login sidebar that controls this parameter.

You will submit an attack that uses SQL injection to log into the "jhalderm" user account without knowing the password. Your submission in each case will be a web page that, when viewed in a browser, performs the attack. (Hint: you will need to use the cross-domain POST techniques from discussion on Friday, October 23.) Place the attack that works against the version with no defenses in a file called "sqlnofilter.html". If you decide to attack the filter, place the attack it in a file called "sqlfilterquotes.html".

Note: 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. Also, for both parts, be sure that your attack is logging you in as the correct user and persists after you refresh the page, especially if the victim user is neither the most nor the least recently created user!

5.2 Cross-site Request Forgery (CSRF)

You will implement a login CSRF attack that will force the victim to log into an account of your choice, thus allowing you to steal the victim's search history. The set of defenses used is controlled by the "`csrfdefense`" query parameter to all the URLs (sent via GET, not POST). The default is 0, the lowest setting. For your convenience, we have included a dropdown in the sidebar that can be used to configure the CSRF difficulty. If you want to change the defenses while using the page manually, be sure to click the "Change CSRF Defense" button in the login sidebar to update the sidebar *before* logging in.

You must attack the following levels of defense:

0. No defenses.

1. Simple validation token: the server maintains a global counter that is incremented and added to a hidden form field in the login form; the login attempt is accepted if the submitted counter value is in the interval $[currentcounter - 10, currentcounter)$. The counter is submitted with the rest of the `/login` POST request using the parameter name "`csrfcounter`". Apparently, the site developer expected light, constant traffic to the site; you may expect this as well. Note that you may *not* assume that the counter starts at 1.

2. Referer validation: the server will reject login requests if the Referer header is for a site other than the server's. However, the server will accept requests that have no Referer at all. (Note: the provided implementation of Referer validation requires that the server is hosted on port 8000 and you are accessing the site directly at `http://127.0.0.1:8000`. If this becomes an issue, feel free to edit the Referer string in `_validate_referer` in the provided views.py.)

Your submission for the zeroth and first levels of defense will be a web page 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 your solution!) The web pages must be self-contained and static (i.e., HTML files only, but they may contain CSS and JavaScript), and must not use unnecessary JavaScript when simple HTML would do. For level of defense n, title your submitted file "`csrf`n`.html`". 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.

Your submission for the second level of defense will be in the same format as the submission for the previous levels, except that the page, when viewed by a victim, will cause the victim to be logged out of the target website if the victim is logged in.

For all levels of defense, your attack page should not display evidence of an attack (e.g., pages from the target website). Instead, it should be blank. It is OK if the page is briefly not blank before correcting itself.

For the first level of defense, you may exploit the cross-site scripting vulnerability in the `/search` page to accomplish the goal.

5.3 Cross-site Scripting (XSS)

You will implement a simple XSS attack against the search box, which does not properly filter search terms before echoing them to the results page. As with the previous attacks, the server supports a "`xssdefense`" query parameter to `/search` to control the level of defense in use. Unlike the CSRF attack, you do not need to use any button to change the difficulty of the XSS attack; simply set the difficulty before clicking the search button (and set the query parameter in your submission).

There are 6 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. In each case, you should submit the simplest attack you can find that works against that defense. In other words, 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 level of defense.

The payload (code that the attack is trying to execute) is 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:

- Steal the victim's username and password.
- Modify the username and password that will be submitted to force a login failure.
- Continue to attack after the failed login attempt (i.e., all login attempts should be stolen and forced to fail in the same way).

If the victim gives up and tries to reset his or her password, the payload must steal the victim's answer to the secret question.

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". Note that the payload should be the same in all cases; attach a readable version of the payload in an additional file called "xsspayload.js" (but 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 "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. Note that you should never send any actual sensitive information (e.g., your code) to this URL.

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

6 Running the server and accessing the site

We have provided code for the target website, which is implemented in Python using the Django Web framework (`http://www.djangoproject.org/`. You do not need to understand this code in order to complete the project, although you are free to look at it. In particular, you may wish to examine the XSS filters in server/search398/simplesearch/templatetags/xss_filters.py and the SQL query you are attacking in server/search398/simplesearch/views.py. We have left debugging messages turned on in the Django settings in case you run into trouble with the provided code.

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

```
$ 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
```

Before running the server, you need to run `setenv PYTHONPATH $HOME/lib/python` if you use the CAEN default C shell (your prompt will be a percent sign) or `export PYTHONPATH=$HOME/lib/python` if you use the bash shell (recommended; your prompt will be a dollar sign). You also need to edit settings.py to include the path to the directory where you have extracted the server. Now you can run the server by changing to the directory where you have extracted it and running `python2.5 manage.py syncdb` to create the database, which is a file in your home directory called 398p2.db. You only need to perform this step once unless you delete the database. 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.

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.

This project will be graded under Mozilla Firefox 3.0.14. Firefox 3.0.11 is available under CAEN Linux; it should be functionally equivalent for our purposes. Please note that there are some known security vulnerabilities in Firefox 3.0.11 (see `http://www.mozilla.org/security/known-vulnerabilities/firefox30.html\#firefox3.0.14`), including an XSS vulnerability. You may not and should not need to exploit any of these vulnerabilities. (By the way, the list of known vulnerabilities includes some serious recent man-in-the-middle attacks against SSL (see `http://www.thoughtcrime.org/papers/null-prefix-attacks.pdf`), and you should consider the risks before doing anything sensitive with it. This illustrates the importance of promptly applying security updates.)

7 Writeup

For each of the three attacks (XSS, CSRF, and SQL injection), write a paragraph about the techniques you would 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 and/or have suggestions about we might improve this project in the future, include them as well.

8 Hints and Guidelines

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

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 Firefox 3.0. Cross-browser compatibility is one of the major headaches of Web development. Furthermore, recent versions of Google Chrome and Internet Explorer 8 include defenses against cross-site scripting that will interfere with your testing.

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). `http://en.wikibooks.org/wiki/JavaScript/` is a gentler introduction to the language.

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.

9 Groups

You must work in groups of 2 for this project. If you wish, you may work with a different partner than you worked with for the previous project. We recommend that you sit down and 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 benefit from the project if you try to divide it into independent parts and work on the parts separately.

10 Turning it in

Submit your project via email to `swolchok@eecs.umich.edu` by 5:00 PM on Thursday, October 29, 2009. Your submission should be a gzipped tar archive named proj1.uniqname1.uniqname2.tar.gz with all the files in one directory called proj2. So that we can process submissions easily, please put your uniqnames in alphabetical order.

Late submissions are guaranteed not to receive any extra credit and may also incur substantial penalties. Start early!