CS 161 Computer Security
Spring 2010 Paxson/Wagner

Project 3

Last updated: 04/16/2010 10:21pm

Due Thursday, April 29, 11:59pm

StockBank is a stock management web application, hosted at `http://lilac.cs.berkeley.edu/`, which allows registered users to post profiles, buy "stocks" and transfer them to each other. Each registered user starts with a balance of 10,000 "dollars" to buy stocks with. In this project, your task will be to construct 4 different attacks against the StockBank web site.

WARNING

You will be executing real attacks on a real web site served from a real machine. You must limit yourselves to the attacks assigned, and you must not attempt to execute malicious code (shell code, native code, etc.) on the server. You must not attempt to compromise any of the user accounts on the server or explore its file system. All of the attacks you will be executing are attacks on other users of the web application, not on users of the machine hosting the application. Additionally, you must not attempt to DoS the server or prevent other students from working on the project in any way.

Getting Started

Begin by exploring the StockBank application hosted at `http://lilac.cs.berkeley.edu/`. Next, download and browse through the source code that the server is running. A tarball containing this code can be found at `http://inst.eecs.berkeley.edu/~cs161/sp10/projects/proj3-code.tgz`. Although many real-world attackers do not have the source code for the web sites they are attacking, you are one of the privileged ones.

Collaboration

You may work with at most one other person on this project. If you are in need of a partner, please use the newsgroup to find one.

You may not collaborate with any students other than your partner. You may share general information on web technologies (e.g., JavaScript, HTML, PHP) if it is not specific to the questions on this project, but you must not share tips, advice, hints, etc. on how to solve any of the questions on this project with anyone other than your partner.

You and your partner must write up solutions entirely on your own. The two of you may work together to jointly write the solution the two of you will submit, but no one else may help you. You must never read or copy the solutions of any other students, and you must not share your own solutions—not even partial

solutions—with students other than your partner.

Submissions and Grading

Like Projects 1 and 2, all submissions for this project will be electronic. You will submit (7-bit ASCII) text files named `a.txt` and `d.txt` for parts (a) and (d) respectively. You will submit HTML documents named `b.html` and `c.html` for parts (b) and (c) respectively. The submission system will accept and grade any subset of these files. You must also include a file named `collaborators.txt` in your submission, which must contain a whitespace-delimited list of the logins (e.g., "cs161-xy") of both members of your group. It does not matter which student in the group submits.

All questions for this project will be graded completely automatically by a continuously-running autograder. Each iteration of the autograder's loop will grade (in order of submission time) all the submissions it has not yet graded. The loop sleeps between iterations, so you should not expect immediate feedback from the autograder. Unlike in Project 2, you may submit your code for autograding as many times as you like.

Feedback from the autograder will come in the form of email to your class account (and to that of your partner, if applicable). Instructions on how to retrieve email delivered to this account can be found here: `http://inst.eecs.berkeley.edu/connecting.html#email` . Timestamps reported by the autograder are in UTC, not local time. You can subtract 7 hours from UTC to get Pacific Daylight Time.

Constructing Your Attacks

Testing site vs. grading site

We have set up two copies of the web application, a testing site and a grading site. You will use the testing site, which is at at `http://lilac.cs.berkeley.edu/` . You may freely try out attacks, using the testing site. In contrast, the autograder will be grading your attacks using the grading site, which is at `http://lilac.cs.berkeley.edu/grading/` . You cannot access the grading site, but the attacks you submit must be designed to work on the grading site, not the testing site. Your submitted attacks will not pass the autograder's tests if they are designed to work against the testing site.

Payload recipient

Some of the attacks ask you to steal some private information and send it off somewhere. In attack A, you will steal a cookie and send it somewhere, while attack C involves sending a username and password somewhere. In a real attack, you would send these payloads off to yourself so that you would receive the credentials. In this project, however, you're going to craft your attack to submit this information to a web site.

Where should the private information be sent? We have set up a special web page where you can submit this information, so you can see whether your attack is working. During testing, you may submit the private information to `http://lilac.cs.berkeley.edu/log.php` . The private information should be passed in a query string parameter named `payload`. For instance, if your attack script has found out that the secret password was `abc123`, then your attack could make an HTTP request to the URL `http://lilac.cs.berkeley.edu/log.php?payload=abc123` . On the test site (the one you have access to), the `log.php` script simply outputs the payload it gets so that you can check that it is what you expect it to be. That's how to test out your attack.

Once your attack is working against the test site, you will need to modify it so it will work with the grading site. For autograding, your solution needs to submit the private information to a different URL: namely, to `http://lilac.cs.berkeley.edu/grading/log.php`. For example, the attack might make a HTTP request to `http://lilac.cs.berkeley.edu/grading/log.php?payload=abc123` to demonstrate to the autograding script that you managed to learn the secret value `abc123`. On the autograding site, the payload will be logged to a file that the autograder will use while it is grading your attack. Once you get your attack working against the test site, make sure to change the logging URL to point to the grading site before submitting your solution for grading.

Browser

We will grade your project with default settings using Mozilla Firefox 3.6.3 (the latest version as of the date this project was released). 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 in Internet Explorer (for example) may not work in Firefox. In particular, you should use the Mozilla way of adding listeners to events (see `https://developer.mozilla.org/en/DOM/element.addEventListener`). We recommend that you test your code on Firefox before you submit, to ensure that you will receive credit for your work.

Database

The autograder will run each of your attacks after wiping clean the database of registered users (on the grading site). Any data you submitted to the testing site and any accounts you created on the testing site while working on the assignment will not be present during grading. The attacks you submit must be designed to work on an unaltered, isolated instance of the site. You won't be able to interact with the grading site other than by submitting the single document that each problem calls for.

Additionally, the testing site uses one database for everyone, which means that all students will be creating users within the same namespace. When creating test users, the username you submit will automatically have random digits appended to it. This will help avoid collisions between usernames created by different students. To see the actual login that was created for you, look for the link after registration that says "Log out username". For example, if you register with user "foo", you might see a link that says "Log out foo4567", which means the login created for you is "foo4567".[1]

Finally, we reserve the right to delete users from the database of the testing site at any time during the course of the project if it gets too large, so please keep local copies of any important data you submit there.

Odds and ends

It is important to check the newsgroup for announcements and information on this project. This specification may be updated with clarifications as they arise.

Attacks C and D are significantly more challenging than attacks A and B, so pace yourself wisely.

You should not wait until the last minute to test your attacks. You should expect that `lilac` will be overloaded close to the submission time.

[1] Updated 04/16/2010 3:36pm.

Attacks

1. **(25 pts.) Attack A: Cookie Theft**
   In this attack you must figure out how an attacker could steal a user's cookie and log the value of the cookie via the log page (`log.php`, as described above). Construct a malicious URL with the following property: if some poor victim who is logged into StockBank visits your URL, the attack will cause the cookie their browser has associated with StockBank to be submitted to the log page.

   Except for the browser address bar (which can be different), when the victim visits your malicious URL, the victim should see a page that looks exactly as it normally does when the victim visits `users.php`. 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 is ok if the page looks weird briefly before correcting itself.)

   Put your URL in a file named `a.txt`. The URL you put in the file should point to the grading site, i.e., it should start with `http://lilac.cs.berkeley.edu/grading`. The autograder will log into StockBank on the grading site, then load your URL in the browser and check that all the conditions above have been met.

   Hint: You can use this example attack as a starting point.
   `http://lilac.cs.berkeley.edu/users.php?`
   `user=%22%3E%3Cscript%3Ealert%28document.cookie%29%3B%3C/script%3E`

2. **(25 pts.) Attack B: Cross-Site Request Forgery**
   Next, you must figure out how an attacker could mount a CSRF attack against StockBank. Construct a malicious HTML page with the following property: If some poor victim who is logged into StockBank visits your HTML page, then it will cause 10 dollars to be transferred from the victim's account to the account of a user named `attacker`. As soon as the transfer is complete, the browser should be redirected to `http://lilac.cs.berkeley.edu/grading/` (this should happen so fast the user might not notice).

   Put your malicious HTML document in a file named `b.html`. The autograder will log into the StockBank grading site under some victim's account and then open your HTML file in the browser. Thus, you can assume that the autograder will already be logged in to StockBank before loading your page. Also, you can assume that the victim account used by the autograder will have at least 10 dollars in it.

3. **(25 pts.) Attack C: Password Theft**
   We're not done yet. Now let's see how a malicious web site could exploit a vulnerability to steal passwords of StockBank users. Construct a malicious HTML page, so that if some poor victim who is not already logged into StockBank visits your page, they will be taken to the StockBank login page, such that if the poor victim then logs in to the StockBank site, the attacker learns their username and password.

   Upon loading your HTML document, the browser should immediately be redirected to the StockBank grading site at `http://lilac.cs.berkeley.edu/grading/`. The login form should appear perfectly normal to the victim. No extraneous text (e.g., warnings) should be visible. The browser's address bar should display the legitimate `http://lilac.cs.berkeley.edu/grading/` site. Assuming the victim enters in their username and password correctly on this page, the login should proceed the same way it always does; the victim should not notice anything out of the ordinary.

   If the victim types their username and password into the login form displayed after loading your document, the attack should cause the victim's username and password to be logged using the site's log page (`log.php`). The username and password should be separated by a comma (and no space): for example, if the username is `mikey` and the password is `crikey`, the attack should cause a HTTP request to be sent to

`http://lilac.cs.berkeley.edu/grading/log.php?payload=mikey,crikey`. (Of course, for testing you can log to the corresponding `log.php` page on the testing site, to check that everything is working.)

Submit your HTML document in a file named `c.html`. The autograder will open your HTML document, `c.html`, in a web browser. The autograder will then enter a username and password and press the "Log in" button. Everything should work as described above. The autograder will not be logged in to StockBank before opening your submitted HTML document in the browser.

Hint: The site uses the PHP `htmlspecialchars()` function to sanitize the reflected username, but something is not quite right.

Partial credit: If you are unable to generate a payload that steals the username and password, you can earn partial credit if your attack takes the user to the login form and causes a JavaScript alert dialogue box to pop up. This will let us know that you found the security hole (which is half the battle), but that you were unable to generate the correct payload for the password stealing attack.

4. (25 pts.) Attack D: Profile Worm

WARNING You must not look at the "profile" of any other student's account on the StockBank web site. Doing so will be viewed as the equivalent of cheating.[2]

The StockBank site allows any user to create their own profile by editing their profile on the page labelled "Home". Also, one can view the profile of other users by visiting the page labelled "Users" and entering in the username of the other user. The goal in this problem is to create a malicious user profile that replicates in a worm/virus-like fashion.

Here is how the attack should work. Every time a new user B views the profile of some infected user A, user B's profile should become infected with the malicious content. If user C then views user B's altered profile, user C's profile should also be altered so C is infected, and so on.

In addition, you (the attacker) might as well make some money while you're at it. So you must also arrange that every time your attack infects a new user's profile, it also steals 1 dollar from that user's account.

Submit a file `d.txt` containing a malicious profile that, when viewed, transfers 1 dollar from the user viewing the profile to the user with username `attacker` and overwrites the profile of the viewing user with itself (the contents of `d.txt`).

To grade your attack, the autograder will paste the submitted profile into the profile of the `attacker` user, log into the `victim` user, and view `attacker`'s profile using account `victim`. The autograder will subsequently check that the profile of `victim` is identical to that of `attacker`. The autograder will also check that one dollar has been transferred from `victim` to `attacker`. The transfer and replication should be reasonably fast (under 15 seconds). During that time, the autograder will not click anywhere.

During the transfer and replication process, the browser's location bar must contain `http://lilac.cs.berkeley.edu/grading/users.php?user=username` throughout, 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 dollars (regardless of that user's true balance).

You do not need to concern yourself with the corner case in which the user viewing the infected profile has no dollars to send. This situation will not be part of the grading.

[2] Updated 04/16/2010 10:21pm.

Hint: The site allows a sanitized subset of HTML in profiles, but you can get around it. The following MySpace vulnerability may provide some inspiration for this attack:
`http://www.securityfocus.com/archive/82/430263/30/120/threaded`.[3]

[3] On a related note, the Samy worm makes for an interesting story. If you are curious, see the writeup here: `http://namb.la/popular/` and `http://namb.la/popular/tech.html`.