# Web Security Project Part 1

CS155: Computer Security
Spring 2013
Due May 13

## Update - May 5

Because of an interaction between the framebusting code used to "defend" against clickjacking, attack B has been made unnecessarily complicated. Because of this, we have adjusted the project just slightly.

We have added pages /transferB.php and /transferC.php to be targetted for attack B and C respectively. /transferC.php contains the CSRF protection described in the project description as well as the framebusting code. /transferB.php is identical except that it contains neither of these defenses. Other than these features, both pages are identical to the /transfer.php code you have access to.

Your attack C *must* target /transferC.php and not /transferB.php. Your attack B *must* target /transferB.php and not /transferC.php. If you have already built an attack for part B that works against the original /transfer.php page (it is possible, but tricky) then please note that in your submission and we will work out some extra credit. /transfer.php has not been updated.

These changes have been reflected in the problem descriptions below. Please contact the course staff with any questions. We apologize for any confusion this has caused.

## Overview

The fictional "Zoobar Foundation" is has set up a simple web application at zoobar.stanford.edu, allowing registered users to post profiles and transfer "zoobar" credits between each other. Each registered user starts with 10 zoobars.

You will craft a series of attacks on zoobar.org that exploit vulnerabilities in the website's design. Each attack presents a distinct scenario with unique goals and constraints, although in some cases you may be able to re-use parts of your code.

Although many real-world attackers do not have the source code for the web sites they are attacking, you are one of the lucky ones: source code is available. You won't actually need to look at the site's source code until Part 2, but it's there if you get stuck.

Your attacks will run in a restricted network environment that can only connect to zoobar.stanford.edu and crypto.stanford.edu. We will run your attacks after wiping clean the database of registered users (except the user named "attacker"), so any data you submitted to zoobar.stanford.edu while working on the assignment will not be present during grading. We reserve the right to delete users from the database at any time if it gets too large, so please keep local copies of any important data you submit there.

## Setup

We will grade your project with the default settings using the latest official release of the Mozilla Firefox browser at the time the project is due. 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. We recommend that you test your code on Firefox before you submit, to ensure that you will receive credit for your work.

The source code is available at: https://github.com/deian/zoobar if you want to view it. More details about how to set up your personal web server will be included in the writeup for part 2.

## Useful Resources

- Web Programming References:
  www.w3schools.com, php.net
- AJAX:
  www.openjs.com/articles/ajax_xmlhttp_using_post.php

- Myspace Worm:
  crypto.stanford.edu/cs155/hw_and_proj/proj2/advisory4.5.06.txt
- Evading XSS Filters:
  www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
- Clickjacking Attacks and Defenses:
  seclab.stanford.edu/websec/framebusting/framebust.pdf,
  media.blackhat.com/bh-eu-10/presentations/Stone/BlackHat-EU-2010-Stone-Next-Generation-Clickjacking-slides.pdf

## Attack A. Cookie Theft

- Your solution is a URL starting with zoobar.stanford.edu/users.php?
- The grader will already be logged in to zoobar.stanford.edu before loading your URL.
- Your goal is to steal the document cookie and make an HTTP GET request to zoobar.stanford.edu/cookies.php?cookie=...cookie data here...
- Except for the browser address bar (which can be different), the grader should see a page that looks exactly as it normally does when the grader 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's ok if the page looks weird briefly before correcting itself.)
- Hint: /users.php?user="><script>alert(document.cookie);</script>is an example attack to use as a starting point.

## Attack B. Cross-Site Request Forgery

- Your solution is a short HTML document that the grader will open using the web browser.
- The grader will already be logged in to zoobar.stanford.edu before loading your page.

- Transfer 10 zoobars from the grader's account to the "attacker" account. As soon as the transfer is complete the browser should be redirected to http://crypto.stanford.edu/cs155/ (so fast the user might not notice).
- The location bar of the browser should not contain zoobar.stanford.edu at any point.
- Note: There is no framebusting code in /transferB.php.
- /transferB.php is the intended target for this attack. If your attack already works on /transfer.php, please note that in your submission.

## Attack C. Clickjacking

- Your solution is one or two HTML documents. The grader will open one of them (the one named c.html) using the web browser.
- The grader will already be logged in to zoobar.stanford.edu before loading your page.
- The grader will interact with the web page in a way that is reasonable. This means if there is a button on the page for the user to click the grader will click it. The same goes for any other interaction on the page.
- After interacting with the page, 10 zoobars should be transferred from the grader's account to the "attacker" account. As soon as the transfer is complete the browser should be redirected to http://crypto.stanford.edu/cs155/.
- It should not be obvious that your page is loading content from zoobar.stanford.edu.
- Note 1: For this attack, some CSRF protection code has been added to transferC.php. If a POST request to /transferC.php is made without the proper token then no database updates will occur but all other behavior will be the same. Your attack MUST work in the presence of this code. This means that you must have the user interact with a form on an actual zoobar.stanford.edu page. This code does NOT appear in

the source you have access to since it would give away the defense for Attack B.

The way that this CSRF protection code is implemented actually makes your lives a little easier! Think about how it is implemented and what you can do with it.

- Note 2: There is some basic framebusting code that protects the application from this sort of attack (it appears in includes/common.php in the repo). You must come up with a way of bypassing this defense.
- Hint: Make sure to test your page layout after resizing your browser to make sure the attack will work for reasonable page resolutions (we will test using a minimum resolution of 800x600).

## Attack D. Profile Worm

- Your solution is a profile that, when viewed, transfers 1 zoobar from the current user to a user called "attacker" (that's an actual username) and replaces the profile of the current user with itself.
- Your malicious profile may include a witty message to the grader (optional, but it helps us see that it replicated).
- To grade your attack, we will cut and paste the submitted profile file into the profile of the "attacker" user and view that profile using the grader's account. We will then view the copied profile with more accounts, checking for the transfer and replication.
- The transfer and replication should be reasonably fast (under 15 seconds). During that time, the grader will not click anywhere.
- During the transfer and replication process, the browser's location bar should remain at http://zoobar.stanford.edu/users.php?user=username, 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 zoobars.
- You will not be graded on the corner case where the user viewing the profile has no zoobars to send.

- Hint: The site allows a sanitized subset of HTML in profiles, but you can get around it. This MySpace vulnerability may provide some inspiration.

## Extra Credit. Password Theft

- Your solution is a short HTML document that the grader will open using the web browser.
- The grader will not be logged in to zoobar.stanford.edu before loading your page.
- Upon loading your document, the browser should immediately be redirected to http://zoobar.stanford.edu/. The grader will enter a username and password and press the "Log in" button.
- When the "Log in" button is pressed, steal the username and password by making a HTTP GET request to http://zoobar.stanford.edu/credentials.php?username=...&password=... (insert the correct username and password).
- The login form should appear perfectly normal to the user. No extraneous text (e.g. warnings) should be visible, and assuming the username and password are correct the login should proceed the same way it always does.
- Hint: The site uses htmlspecialchars() to sanitize the reflected username, but something is not quite right.

## Deliverables

- Create files named a.txt, b.html, c.html (you can optionally include a second file c2.html), d.txt, and e.html (if you did it) containing your attacks.
- You are to provide a tarball (i.e., a .tar.gz or .tar.bz2 file) containing the attacks.
- Along with your attacks, you must include file called ID which contains, on a single line, the following: your SUID number; your Leland username; and your name, in the format last name, comma, first name.

If you did the project with a partner, then both of you will submit only one solution and the ID file will have two lines giving the relevant information.

You may want to include a README file with comments about your experiences and suggestions for improvement.

## Grading

Beware of Race Conditions: Depending on how you write your code, all four of these attacks attacks could potentially have race conditions that affect the success of your attacks. Attacks that fail on the grader's browser during grading will receive less than full credit. To ensure that you receive full credit, you should wait after making an outbound network request rather than assuming that the request will be sent immediately.

## Late Policy

Every student in the class is given a total of 72 late hours that can be applied to the projects and homeworks. These late hours must be taken in chunks of 24 hours (essentially 3 late days) for example, submitting a homework 3 hours later than its due counts as 24 late hours used. After all your late hours are used up, the assignment score gets halved with every 24 hours the assignment is late for example, someone submitting a project 47 hours late after having used all her late days will get $(1/2)2 = 1/4$ of 4the grade. If a project has more than one part, each part is considered a separate assignment for late days for example, if you submit part 1 72 hours late and part 2 24 hours late, then part 1 gets full credit and part 2 gets 50% credit.

Note that no late hours can be used on the last programming project.