CSE 543 Fall 2013 - Assignment 4

December 3, 2013

1 Dates

- **Out**. *3rd Dec 2013 (Tuesday)*
- **Due**. *11th Dec 2013 (Wednesday), 11:59PM*

2 Introduction

In this project, you will exploit a series of web vulnerabilities discussed in class on WebGoat [4], an insecure web application designed for educational purposes. **Note: This project is for educational purposes only, and should *never* be used outside of this class**.

The website for the WebGoat web application is `http://localhost:8080/WebGoat/attack` on your assigned MITM machines from assignment 1, with username and password both `guest`.

3 Vulnerabilities

3.1 HTTP Response Splitting

An attacker uses input filtering deficiencies in header elements (e.g., redirect URLs, cookies) to split the HTTP response into two, the second of which is completely controlled by the attacker. The second response can be used to poison the cache of a proxy server, so any future requests will serve that page instead!

Read "What is HTTP Response Splitting?" in [3]. Using this knowledge, solve Stage 1 of HTTP Splitting in WebGoat (General → HTTP response splitting).

3.2 Cross-Site Scripting

In a cross-site scripting (XSS) attack, an attacker executes her code in the context of a trusted website. In this exercise, we will look at the stored and reflected variants of XSS.

In a stored XSS attack, an attacker stores injected code permanently into a website (e.g., forum). In a reflected XSS attack, an attacker uses social engineering or other flaws to craft a URL with some code that is "reflected" back by the server to the client, where it is executed in the context of the server's code.

Read [2] upto the "Attack Scenarios" section. Then, solve Stored XSS attacks and Reflected XSS Attacks (Cross Site Scripting → Stored XSS Attacks, Reflected XSS Attacks – not the ones under the LAB). Once solved, green ticks appear on the side of these links.

3.3 Cross-Site Request Forgery

In cross-site request forgery, an attacker exploits the trust the server has in the browser to execute a request of the attacker's choosing when the victim is authenticated with the server. A CSRF attack can be carried out by tricking the victim into clicking a link using social engineering, or even using a stored XSS attack, the latter being used in the exercise.

Read [5] upto the section "Misconceptions". Then, solve the CSRF exercise on WebGoat (Cross Site Scripting → Cross Site Request Forgery (CSRF)). Once solved, a green tick appears on the side of the link.

3.4 Session Hijacking – Session Fixation

There are several ways that an attacker can get a session (i.e., authenticate) with a server as another user without knowing the user's password or other authentication token. One such technique is to hijack a session by stealing cookies (as FireSheep [1] did). Another way to hijack is to fixate a session, which will be the focus of our exercise. In a session fixation attack, an attacker forces a victim to use a chosen session ID using social engineering or other techniques. Once the victim logs in, that session ID changes state to authenticated, and the attacker can use the session ID and be authenticated as the victim.

Read the wikipedia article on session fixation [6]. Solve the WebGoat exercise on Session Fixation (Session Management Flaws → Session Fixation). Once solved, a green tick appears on the side of the link.

3.5 Insecure Cookie Design

Another way an attacker can authenticate with a server without knowing the user's password is to forge an insecure cookie. We will demonstrate an example of insecure cookie design by solving the WebGoat exercise on spoofing an authentication cookie (Session Management Flaws → Spoof an Authentiation Cookie).

Since this exercise involves analyzing and changing cookies, we will use two Firefox extensions to help: (1) Live HTTP headers, and (2) Tamper data. First, open the Live HTTP headers extension (under Tools menu), and login as the users specified in the exercise. Analyze the cookies for patterns. You should find a (forgeable) relation between the username and the cookie set (hint: reverse and caesar cipher). Next, you need to design the cookie to be authenticated as the user "Alice". Once designed, fire up the Tamper data extension, and click on "Start Tamper". Submit a login request. The tamper data extension will automatically ask if you want to modify the outgoing HTTP request. Use this to change the cookie to the one for Alice. Once solved, a green tick appears on the side of the link.

4 Instructions

It is not necessary to submit any code for this exercise. The green ticks will indicate whether the exercises have been solved successfully. Once you have completed all exercises, send an email stating the same to `hvijay at cse psu edu` and copy `tjaeger at cse psu edu` with an answer to the following question:

- Which of the above attacks can the same-origin policy defend against?

5 Evaluation

- HTTP Response Splitting **- 5 points**
- Cross-Site Scripting **- 3 points**
- Cross-Site Request Forgery **- 3 points**
- Session Fixation **- 5 points**
- Insecure Cookie Design **- 7 points**
- Answer to question **- 2 points**
- Total **- 25 points**

Hayawardh Vijayakumar

References

[1] Eric Butler. Firesheep. `http://codebutler.com/firesheep/`, 2013.

[2] Sudhanshu Chauhan. Cross-site scripting (xss). `http://resources.infosecinstitute.com/cross-site-scripting-xss/`, 2012.

[3] Arvind Doraiswamy. Http response splitting attack. `http://resources.infosecinstitute.com/http-response-splitting-attack/`, 2011.

[4] OWASP. Category:owasp webgoat project. `https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project`, 2013.

[5] Pavitra Shankdhar. Fixing csrf vulnerability in php applications. `http://resources.infosecinstitute.com/fixing-csrf-vulnerability-in-php-application/`, 2013.

[6] Wikipedia. Firesheep. `http://en.wikipedia.org/wiki/Session_fixation`, 2013.