ROYAL INSTITUTE
OF TECHNOLOGY

School of Electrical Engineering and Computer Science
Division of Theoretical Computer Science

LAB W
Web attacks: XSS, XSRF and SQL injection

NAME	KTH USERNAME

DATE :: _____-_____-_____

TEACHING ASSISTANT'S NAME :: _____

LAB W PASSED (TA'S SIGNATURE) :: _____

BONUS PASSED (TA'S SIGNATURE) :: _____

Computer Security

Compile date: August 24, 2021

DD2395 / HT2021

Contents

1 Introduction **1**

1.1 Preparation 1

2 XSS and XSRF **3**

2.1 XSS: Cookie Theft 3

2.2 XSRF: Stealing Zoobar Credits 5

2.3 Protection against XSS and XSRF 6

2.4 Acknowledgments 7

3 SQL injection **8**

3.1 Preparation: Burp Proxy Setup 8

3.2 Wraithmail 8

3.3 Cloaknet 9

3.4 Protection against SQL injection 12

4 History **13**

1 Introduction

The lab is done in a virtual machine, see the The Dasak VM on Canvas for up-to-date instructions on how to download and set it up. After logging into the VM, you have to start the webservers for this lab by opening a terminal and executing the following command:

```
sudo lxc-start -n webattack
```

Make sure to execute the command after each reboot of the VM if you want to work with the webattacks. For the webattacks, you should use the Firefox browser (v42) present in the VM. It is important that you don't update the browser, since security updates may introduce difficulties during your attack. **Do not** go to the "about" menu or try update it. Another useful tool already available on the VM is "BurpSuite". For the purposes of this lab, BurpSuite can act as HTTP-proxy to allow inspection and interception of HTTP traffic between the browser and the web server, as well as URL encoding and decoding.

The goal of this lab is to get familiar with web security. The lab covers four common web application vulnerabilities: Cross-Site Scripting (XSS), Cross-Site Request Forgery (XSRF/CSRF), Insecure Direct Object References and SQL injection. According to the Open Web Application Security Project (OWASP), these vulnerabilities are among the top ten critical web application security flaws. This lab will help you find and exploit vulnerabilities in several web applications that run on a test-bed web server.

If you need help, you are welcome to come to the scheduled lab sessions, but since these might be crowded, make sure to book a seat by signing up in advance. You can, however, also work remotely **from home**, but you will still have to show your answers and solutions to a lab assistant once you are finished. In order to accomplish this requirement you will have to sign up for a regular lab session. This lab should be solved in a **group of three students**. If you do not have a partner nor find one (e. g., by asking on the course web, i.e., Canvas), let us know as soon as possible.

Deadline

The deadline for the lab can be found on the course website:
https://kth.instructure.com/courses/27095

1.1 Preparation

The lab exercises require a certain level of expertise. If you do not have much background knowledge about HTTP, HTML and Javascript, you are strongly advised to prepare by going through the relevant sections of the OWASP Webgoat or Google Gruyere projects. You can also download some virtual machines from the OWASP Broken Web Applications Project. Gruyere is browser-based, so you don't need to download anything.

Do not attack other websites except those intended for this lab

Do not try anything you learn in this lab on other websites than those specified in this lab instructions. Be aware that already inserting a strange string containing some JavaScript syntax into e. g., a search field on a website can be considered an attack and that you are potentially identifiable via your IP address.

The first part of the lab covers Cross-Site Scripting (XSS) and Cross-Site Request Forgery (XSRF/CSRF).

If you need help with JavaScript, you have a tutorial and cheat sheet here:
http://www.w3schools.com/js/default.asp
https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet.

Firefox's *Web Console* and source *Inspector* are both accessible from the `Tools/Web Developer` menu. The Web Console lets you see which JavaScript exceptions are being thrown and why they are happening. The Inspector tool lets you peek at the structure of the page and the properties and methods of the corresponding DOM tree nodes. You can also use the Web Console to analyze HTTP request headers, including the content of cookies sent along with the requests (clicking on a GET request line opens a popup with detailed information). You may need to use Cascading Style Sheets (CSS) to make your attacks invisible to the user. You should know which basic syntax you may need, for instance, `<style>.warning{display:none}</style>`.

If you need to encode certain special characters, such as newlines, take a look at this tutorial:
https://www.permadi.com/tutorial/urlEncoding/

The second part of the lab covers SQL injection. The following SQL tutorial might be useful for refreshing SQL commands and syntax:
http://www.w3schools.com/sql/

We also advise you to have a look at some SQL injection cheat sheets:
https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
https://websec.wordpress.com/2010/03/19/exploiting-hard-filtered-sql-injections/

For this part of the lab you will need an intercepting proxy that will allow you not only to capture the HTTP packets but also modify them before sending them further. One such tool is the free version of the *Burp Proxy*. In Section 3 you find instructions how to install and use this proxy.

2 Cross-site Scripting (XSS) and Cross-site Request Forgery (XSRF)

This section will help you understand what cross-site scripting (XSS) and cross-site request forgery (XSRF/CSRF) are. You will craft a series of attacks against the *Zoobar* web site that will exploit varying 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 solution code.

Zoobar website

The URL for the Zoobar web site is `http://zoobar/`.
You can find Zoobar source files at `https://dasak-vm-lab-server.eecs.kth.se/labw/zoobarPhpFiles.tar`.

Do not use real passwords

You will need to create user accounts on the zoobar website to test your attacks. Do **not** use passwords that you use somewhere else to create the accounts here. The login information will be transferred unencrypted and the passwords are not stored securely on the server, so other students or attackers might be able to read your passwords.

Use Mozilla Firefox from the desktop of the VM

We will grade your attacks with default settings. We chose this browser for grading because it is widely available and it can be run on a great variety of operating systems, but there are subtle quirks in the way HTML and JavaScript are handled by different browsers, e. g., some attacks that work in Internet Explorer may not work in Mozilla Firefox. In particular, you should use the Mozilla way of adding listeners to events.

2.1 Cross-site Scripting: Cookie Theft

You will construct an attack that will steal a victim's cookie for the Zoobar site when the victim's browser opens an URL of your choice. This type of attack is called Reflected Cross-site Scripting attack. You do not need to do anything with the victim's cookie after stealing it for the purposes of this exercise, although in practice an attacker could use the cookie to impersonate the victim and issue requests as if they came from the victim.

Your goal is to steal the document cookie and post it to a log using the Log-Write script found in
`https://dasak-vm-lab-server.eecs.kth.se/logger/log.php`,
a link to the log is found in the same page.

Log Script

In https://dasak-vm-lab-server.eecs.kth.se/logger/log.php, we have links to the log-write and log scripts. The log-write script is used to send messages to the server, and the log script shows the latest messages sent. This log is cleared periodically.
Since the message storage is easily accessible, be sure to not send any sensitive information there.

The victim will already be logged in to the Zoobar site before loading your URL. Except for the browser address bar (which can be different), the victim should see a page that looks exactly as it normally does when the victim visits the site. No changes to the site appearance or extraneous text should be visible. Avoiding the red warning text is an important part of this attack (but it is all right if the page looks weird briefly before correcting itself).

Guiding Questions

What is the *Same Origin Policy*? How does the reflected XSS attack bypass this policy?

A web application vulnerable to reflected XSS will send the received unvalidated input back to the user (e. g., in a search result page). How can you test for the reflected XSS based on this behavior?

Hint: sending the string `<script>alert("XSS works");</script>` via an input field might not suffice. Look at the source code of the returned page and check how and where exactly your input script is reflected.

Which is the vulnerable input element on the website?

How do you use the log script to send data to the logger? Provide an example in JavaScript language.

Which HTML DOM object has a cookie property?

The victim is not supposed to see any error messages. What does the following script do?
`document.getElementById("x").style.display = ...;`

You can also try using a location object. How do you use it to hide the error messages?

Milestone
Milestone 1: Report your progress to a lab assistant.

2.2 Cross-Site Request Forgery: Stealing Zoobar Credits

Now, in this new scenario, you will transfer ten Zoobars from the victim's account to your account. The victim is very naive and will load any HTML document that you send (e. g., as an e-mail attachment – an attacker might use social engineering on less naive victims, to accomplish this).

You need to create one or two short HTML documents that the victim will open using its web browser. You can assume that the victim is logged in (authenticated with the Zoobar application) before loading your document but it should not notice that a transfer of credits has occurred. Therefore, you should either redirect the victim to the course web page https://kth.instructure.com/courses/27095 after the transfer has been completed (fast enough so that the victim might not notice the transfer) or show something else to the victim without redirecting. The main point is that the victim does not become suspicious. In particular the location bar of the browser must not contain ”zoobar” at any point of time (not even for a millisecond). Notice that a funny cat GIF suffices and does not require a redirection - who does not like those?

Guiding Questions

Describe the browser behavior and problems with session management that make XSRF possible.

Look at the source code and the HTTP headers during the transfer of Zoobars. Your crafted request should look exactly like the original. What information (input) is being sent? What type of request should you craft, GET or POST?

One way to submit the form using JavaScript is to call the button's click method:
`document.getElementById('mybutton').click();`
Describe some other alternative manner to submit the form with an example.

You can use also *iframes* to hide your attack and submit the form in a hidden iframe instead of the current page. Describe at least two methods on how you can hide iframes.

Describe some other technique, besides hidden iframes, to make your attack invisible to the victim.

Milestone
Milestone 2: Report your progress to a lab assistant.

2.3 Protection against XSS and XSRF

Questions

Describe how to prevent XSS flaws in a web application.

What can be done by a user (at the browser level) to protect against XSRF? Name at least two alternatives.

How can you prevent XSRF vulnerabilities on the web application (server) level?

Milestone

Milestone 3: Report your progress to a lab assistant.

2.4 Acknowledgments

Thanks to Nickolai Zeldovich at MIT and Stanford's CS155 course staff for the foundation of this lab assignment.

Thanks to Anton Bäckström for restricting some of the vulnerabilties in the web application.

3 SQL injection

The following exercise will help you understand the SQL injection techniques and systematic vulnerability testing. While the main focus is on the SQL injection, there is also one Insecure Direct Object References vulnerability that you are supposed to exploit.

The first part of the lab exercise is based on the Hackxor webapp hacking game. The game consist of a set of sites that you are supposed to hack. The first two steps of this hacking game, which include *Wraithmail* (an e-mail service) and *Cloaknet* (an anonymizing proxy service), are used for this exercise.

If you want to try your skills directly, we encourage you to try solving these first two steps of the game without reading the lab instructions in the following section. You will, however, in any case have to answer all questions.

3.1 Preparation: Burp Proxy Setup

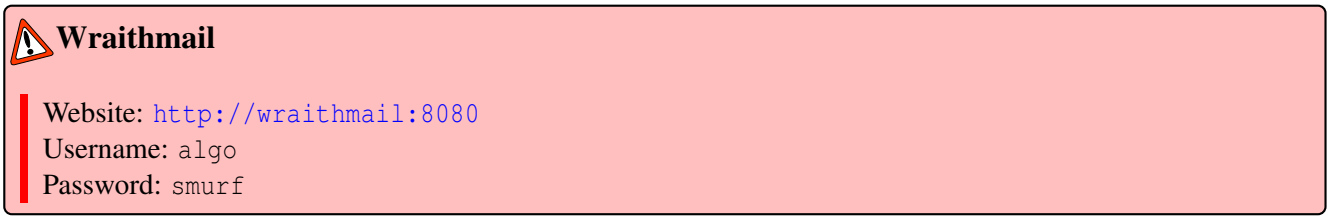
You will need an intercepting web proxy for the exercises, so before getting started, set up the burp proxy:

- Run the proxy from the desktop
- In Firefox v42, select the Edit/Preferences menu, go to Advanced → Network → Settings:
  1. Tick "Manual proxy configuration:"
  2. Set
     - HTTP proxy: 127.0.0.1
     - Port: 8080
  3. Tick "Use this proxy for all protocols"
  4. Delete the "No Proxy for" line

In the Burp proxy software, go to the tab named `proxy`, and when the button named `intercept is on` is enabled (which it is by default) all requests to websites you visit in the browser will be intercepted and held by the proxy software. Once a request has been intercepted, you always have the possibility to change it (just edit the displayed text) before actually sending it out (by pressing the button `forward`).

3.2 Wraithmail

The game scenario is as follows: you are a hacker who got the task to track down a person who performed an attack on one of the Wraithmail accounts. For that, you should login to Wraithmail and read the message named "trace job".



In this first part, you will exploit an Insecure Direct Object References vulnerability. You are encouraged to try any other attack to check whether the site is vulnerable to any of them or not.

Hints

- Pay attention to the *referer* line in the "trace job" letter. What does the referer field mean?
- Explore the site, which functionalities (like composing a message, etc.) does it offer? Is one of them vulnearable? Note: You will need an intercepting proxy to exploit the vulnerability.
- If your attack succeeds, you will be able to see that the *abuse contact* is from *Cloaknet*.

Questions

Briefly describe the attempt attack of the hacker that you are tracking (no details, just the rough idea).

What username was used for the attack and what was the logged IP address?

Milestone

Milestone 4: Report your progress to a lab assistant. _____

3.3 Cloaknet

By now, you have got evidence that the attacker used a proxy service called Cloaknet to hide its IP address. In order for you to find out who the attacker is, you should proceed with the Cloaknet website:

Cloaknet

Website: `http://cloaknet:8080`

There, your task is to perform an SQL injection attack to retrieve the account credentials (login, password, id,...) of all registered users at Cloaknet. You should also be able to confirm that the source of the attack was from `GGHB`, that the target was `wraithmail:8080/send.jsp`, and that the date matches the one you could see in the Wraithmail logs.

You might need to look at the SQL Injection Cheat Sheet mentioned in Section 1.1 to complete this task. Note that testing for the right way to get around the input filters can take some time. Try to test it systematically, but if you get stuck you can ask the lab assistants for hints.

Guiding Questions

Name at least 3 ways (i.e., ”methods” in terms of the HTTP protocol) that are used to transfer data from the browser to the web application (on the webserver) and that can be utilized to perform injection attacks.

A database has many different data types, but all of them can still be classified into two groups based on how they are represented in the SQL statements. Name these groups.

One of the values that goes as input to the SQL query in Cloaknet is user (the username that you type on the login page). Name two other values that might be used as input in the SQL queries.

Hint 1: These two values are sent to the server with two different ”methods” (see the previous question).
Hint 2: Look at the two HTTP requests sent to the server when logging in.

What do you think how many SQL queries are issued, when you login to the website? What inputs can these queries use? How do you think these SQL queries look like? Write them down as precise as possible (e.g. using SQL syntax/pseudo-code).

Check whether the Cloaknet web application’s inputs are vulnerable to SQL injection or not. How do you do that? Give two examples.

Were you able to cause an SQL error or an HTTP error (500) to be displayed on the webpage? How did you do it? What information did you learn? What kind of database and web server are being used?

Why is it useful to know which database is used?

When you try to inject a meaningful SQL string, a first obstacle you might encounter is that white space characters are treated as delimiters in the cookie string in the HTTP header. How can you anyways use space characters in a cookie value?

All inputs are filtered! The filter escapes some characters and removes some words (e.g. `SELECT`). Name at least four techniques to bypass filters in general (with examples).

Hint: check the cheat-sheets mentioned in the introduction.

Table and column names are very predictable in this lab. What could be the name of the table containing all usernames and passwords?

The `UNION` operator is used to combine output from several `SELECT` statements. What requirements should these statements meet in order to be "unioned"? What do you have to do if the tables you want to union do not have the same number of columns?

You should be able to infer the number of columns used in the query statement from the output you see on the page. Sometimes, though, not everything is shown, and then it is important to find out the number of columns. How can it be done? Provide an example.

You can union tables even without specifying column names. How can you include all columns of a table in the `UNION` statement without explicitly naming them?

What are the username and the password of the attacker? How did you get the list of usernames and passwords? How did you know which one was the attacker?

Milestone

Milestone 5: Report your progress to a lab assistant. _____

3.4 Protection against SQL injection

There are many mitigation techniques for SQL injection.

Questions

Describe how parametrized queries help protect against SQL injection. What other protection techniques can be used on the application's code level?

What is a stored procedure? Describe how it protects against SQL injection. What other protection techniques can be used at the database level?

What combination of protection techniques would you choose, to mitigate the SQL injection threat?

Milestone

Milestone 6: Report your progress to a lab assistant.

4 History

Version	**Contribution**	**Author** (Affiliation)	**Contact**
1.0	First development	Oleksandr Bodriagov (CSC/KTH)	obo@kth.se
1.1	Updates for HT2013	Benjamin Greschbach (CSC/KTH)	bgre@kth.se
1.2	Updates for HT2015	Benjamin Greschbach (CSC/KTH)	bgre@kth.se
1.3	Updates for HT2017	Andreas Lindner (CSC/KTH)	andili@kth.se
1.4	Updates for HT2018	Andreas Lindner (EECS/KTH)	andili@kth.se
1.5	Add VM URLs HT2018	Sonja Buchegger (EECS/KTH)	buc@kth.se
1.6	Updates for HT2019	Andreas Lindner (EECS/KTH)	andili@kth.se
1.7	Add new logger info HT2020	Md Sakib Nizam Khan (EECS/KTH)	msnkhan@kth.se
1.8	Adaption for HT2021	Md Sakib Nizam Khan (EECS/KTH)	msnkhan@kth.se
2.0	Improvements (Text fields)	Henrik Karlsson (EECS/KTH)	henrik10@kth.se