DECAMP - University of Padua
Applied WEB application Security

Final project report

Student
Pietro Valente

Student
Alessandro Leonardi

Academic Year 2020/2021

Abstract

This document has the task of describing some important security aspects of Web Applications, explaining how these are related to the proposed practical exercises.

Project demonstration video: https://www.youtube.com/watch?v=W1SajAMJ2fU.

Contents

Chapter 1

Introduction

In today's world web application security is very important. More and more companies create a web page and in many cases security is underestimated. This can cause enormous damage such as the loss of sensitive information. For a good security four Defense Mechanisms must be implemented and managed with care:

- Handling user access to the application's data and functionality to prevent users from gaining unauthorized access.
- Handling user input to the application's functions to prevent malformed input from causing undesirable behavior.
- Handling attackers to ensure that the application behaves appropriately when being directly targeted, taking suitable defensive and offensive measures to frustrate the attacker.
- Managing the application itself by enabling administrators to monitor its activities and configure its functionality.

In the next few chapters we will explain some important security issues and specify which of these groups they fall into.

Chapter 2

Laboratory 1: SQL injections

Web applications normally use one or several back-end systems to store informations such as usernames, passwords or webpage content. Linked to this aspect are the injection attacks, where a malicious person is able to make the web application execute or interpret malicious code / command. The malicious code / command can be executed / interpreted on the web server itself or in one of the backend systems (for example database structure like MySQL). In this laboratory the attacker is looking for a way to extract sensitive information from the site's database and to do it he will exploit SQL injection.

SQL injections can be classified several different ways depending on the viewpoint:

- Response from server: union based / error based / blind injections.
- Data extraction based: in-band / out of band / inferential injections.
- Impact point: first order / second order.

In this lab you will only use union based injections as the server shows possible query errors. This type is the simplest and consists in joining the site's original query to another to obtain sensitive information.

The first thing to do is to verify the existence of an SQL vulnerability. This easily done by inputting a single quote. If this returns an error message, an injection vulnerability is verified. Even if it returns no errors, it still can be vulnerable. In some cases a double quote can elicit an error message.

Your goal for this lab is to delete all posts from other users. To succeed you have to steal the credentials of other accounts, as only the person who created the post has the authorization to delete it. So, after creating an account, focused on the 'Accounts' website section.

Remember,the main steps for exploiting union based injections are:

- Find out how many columns there are in the original query (with ORDER BY or using union and adding null columns).

  Example: ‘ order by 3#

- Find out the type of the injectable parameter (if is a VARCHAR type a single quote will generate a valid query).

- Find out all table names in all database (using information_ schema.tables).

  Example: ‘ union select null, table_ name, null from information_ schema.tables#

- Find out particular columns of a table (using information_ schema.columns).

  Example: ‘ union select null, column_ name, null from information_ schema.columns where table_ name='users'#

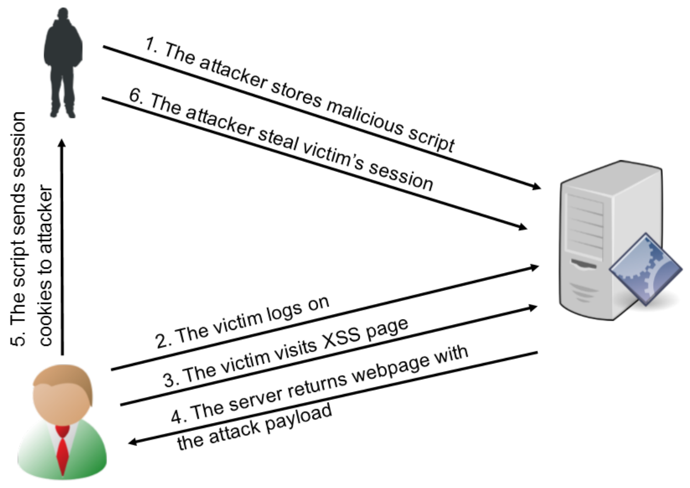
Here, the basic defense mechanisms that is violated is handling user input. Every time input is requested from the user, it should be sanitized. Furthermore in case of an error, this should never be shown to the attacker who could benefit from it, the errors should always be generic.

Chapter 3

Laboratory 2: XSS vulnerability

Cross Site Scripting (XSS) is an attack where malicious person is able to exploit a web application vulnerability in order to execute JavaScript on victim's browser. This type of vulnerability is widespread and ranks third in the OWASP top 10 list. There are three types of XSS vulnerabilities: reflected, stored and DOM based.

In this lab you will only be using a stored XSS vulnerability. In this type of vulnerability, the attacker saves his script on a server page, at this point every time the page is loaded the script will be executed. This is very dangerous, as a user browsing the site normally may not be aware of the execution of the malicious script.


1. The attacker stores malicious script
6. The attacker steal victim's session
5. The script sends session cookies to attacker
2. The victim logs on
3. The victim visits XSS page
4. The server returns webpage with the attack payload

In this laboratory your mission is to steal the user's PHP session. To reach your goal you will have to exploit the lack of validation in the body input of a post, in the 'News Feed' section.

Another important aspect is about the text before your script. It must not make the post look suspicious, this will slow down administrators in fixing the threat.

Also here, the basic defense mechanisms that is violated is handling user input. The server does not check that a body post does not contain javascript scripts. This is extremely wrong and dangerous.

Chapter 4

Laboratory 3: Attacking Access Control

Authenticated users are authorized to access web application resources. Web application users have different level access permission based on their identity, role and access control mechanisms applied in the web application.

In this laboratory your goal is to compromise the access control with a **Vertical privilege escalation** in order to obtain the flag hidden on administrator's private page. This type of privilege escalation is known as privilege elevation, where a lower privilege user or application accesses functions or content reserved for higher privilege users or applications. The access control mechanism is **role-based**, which means the access is based on the role of the user. Usually applications have at least administrator role and a normal user role.

In this web application there is a **Platform misconfiguration**, in particular the application denies the access to a resource depends on a request parameter. The attacker can circumvent the protection looking at POST and GET requests.

Keep eyes on the parameters passed by the application, they could be encoded in order to increase the difficulty to recognize the platform misconfiguration.

In this case, the basic defense mechanisms that is violated is handling user access. An application should never pass sensitive parameters (such as a user's role) via GET. This aspect should be managed server-side.

Bibliography

[1] Kimmo Sauren. *Chapter 3: WEB Application Defense Mechanisms.* Applied WEB application Security (AWAS) course, 2017.

[2] Kimmo Sauren. *Chapter 9: Code Injection Attacks.* Applied WEB application Security (AWAS) course, 2017.

[3] Kimmo Sauren. *Chapter 10: Attacking Other Users.* Applied WEB application Security (AWAS) course, 2017.

[4] Kimmo Sauren. *Chapter 8: Attacking Access Controls.* Applied WEB application Security (AWAS) course, 2017.