Network Security Theory and Practice

Lab 03

Due April 15, 2022

POLICIES:

1. **Coverage**
   Web Security: SQL Injection, CSRF, XSS, etc.
2. **Grade**
   Lab 03 accounts for 20% of the final grade
3. **Individual or Group**
   Individual based, but group discussion is allowed and encouraged
4. **Academic Honesty**
   Violation of academic honesty may result in a penalty more severe than zero credit for an assignment, a test, and/or an exam.
5. **Submission**
   Soft copy of report.pdf on course.zju.edu.cn
6. **Late Submission**
   NO late submission is permitted.
   Late submissions after April 15, 2022 will NOT be graded.

PREPARATION:

1. **Lab Goal**
   Lab 03 aims to practice common web vulnerabilities and protection schemes.
2. **Setup Environment**
   For security reasons, Lab 03 is recommended to be performed in a virtual machine environment. Also, Chrome is not recommended as the practicing browser (because it's too secure…).
   PHPStudy: install the application and start the service.
   DVWA: unzip the file and rename DVWA/config/config.inc.php.dist to DVWA/config/config.inc.php. Then, edit the db_database, db_user, db_password in file config.inc.php the same as the database in the PHPStudy. Finally, move the folder DVWA to PHPStudy/WWW/. Now, you can access 127.0.0.1 through your browser and create a database to start.

LAB REQUIREMENTS:

- Set the security level to "low" by clicking the "DVWA Security" button after deploying DVWA successfully. For those who seek for more challenges, they can choose a "medium" or "high" security level.
- For all the challenges, you need to click the "view source" button to see the backend code and understand how the attack works. Then, try to attack!

1. Command Injection

About: View source and find it doesn't filter the user input. What's more, it puts the unfiltered input into the system shell!

***Objective*:** *Try to input something to find out the user of the web service on the OS, as well as the machines hostname via RCE.*

PS: **Never** put user input (even filtered) into system shell!

2. CSRF (Cross-Site Request Forgery)

About: CSRF is an attack that forces an end user to execute unwanted actions on a web application in which they are currently authenticated. With a little help of social engineering (such as sending a link via Email/chat), an attacker may force the users of a web application to execute actions intended by the attacker.

Objective: *You are supposed to make the current user change their own password, without them knowing about their actions.*

Tips: View the source and find it simply check whether password_new equals to password_conf without a hidden token. Thus, we can forge a url to entice people to click.

3. File Inclusion

About: Some web applications allow the user to specify input that is used directly into file streams or allows the user to upload files to the server. At a later time the web application accesses the user supplied input in the web applications' context. By doing this, the web application is allowing the potential for malicious file execution. If the file chosen to be included is local on the target machine, it is called Local File Inclusion (LFI). But files may also be included on other machines, which then the attack is a Remote File Inclusion (RFI).

Objective: *You are supposed to read all* ***five*** *famous quotes from "../hackable/flags/fi.php" using only the file inclusion. You can achieve the goal through local file inclusion (LFI) or remote file inclusion (RFI).*

Tips: Click "file*.php" and observe the url, while viewing the source, we can find that the server gets file by passing a "GET" parameter, which means we can force the server to access a designated file. Besides, if you want to execute remote file inclusion, you should set "allow_url_include" to "on", which is "off" at present.

If you don't want a php file to be executed while accessing, try to use "php://filter".

4. File Upload

About: Uploaded files represent a significant risk to web applications. The first step in many attacks is to get some code to the system to be attacked. Then the attacker only needs to find a way to get the code executed. Using a file upload helps the attacker accomplish the first step.

Objective: *You are supposed to execute any PHP function you choose on the target system.*

Tips: Upload a file and you can get the direct storage path. Then you can upload a php file including trojan-virus.

5. SQL Injection

About: A SQL injection attack consists of insertion or injection of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (insert/update/delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system (load_file), and in some cases issue commands to the operating system.

Objective: *You are supposed to steal 5 users' passwords in the database through SQL injection.*

Tips: View the source and notice that the url returns the submitted parameter, meaning the server uses a "GET" method. You should check whether there exists SQL injection, whether the injection is character or numeric, how many numbers of fields are in SQL query statement, the order of the displayed fields and the current database, tables, and fields. What's more, once you get the result, remember you only get the **ciphertext** of password.

6. SQL Injection (Blind)

About: When an attacker executes SQL injection attacks, sometimes the server responds with error messages from the database server complaining that the SQL query's syntax is incorrect. Blind SQL injection is identical to normal SQL Injection except that when an attacker attempts to exploit an application, rather than getting a useful error message, they get a generic page specified by the developer instead. This makes exploiting a potential SQL Injection attack more difficult but not impossible. An attacker can still steal data by asking a series of True and False questions through SQL statements, and monitoring how the web application responds (valid entry returned or 404 header set).

Objective: *You are supposed to steal 5 users' passwords in the database through SQL injection (blind). You can achieve the goal through boolean-based SQL injection or time-based SQL injection.*

Tips: The basic steps are as same as those in SQL injection. The only difference is that this time you may need to guess the solution item by item, so try to be patient.

7. Weak Session IDs

About: Knowledge of a session ID is often the only thing required to access a site as a specific user after they have logged in, if that session ID is able to be calculated or easily guessed, then an attacker will have an easy way to gain access to user accounts without having to brute force passwords or find other vulnerabilities such as Cross-Site Scripting.

Objective: *View backend code and understand why these IDs can be predicted.*

8. XSS (DOM)

About: DOM-based XSS (or as it is called in some texts, "type-0 XSS") is an XSS attack wherein the attack payload is executed as a result of modifying the DOM "environment" in the victim's browser used by the original client-side script, so

that the client-side code runs in an unexpected manner. That is, the page itself (the HTTP response that is) does not change, but the client-side code contained in the page executes differently due to the malicious modifications that have occurred in the DOM environment.

Objective: *Run your own JavaScript to get the cookie and explain how the attack works in the real scene.*

Tips: F12 to see the front-end source code. It writes the user's unfiltered input passed with get directly into the html element, which leads to XSS vulnerability.

PS: In order to reduce the workload, in the three questions of XSS, you are both the attacker and the victim, which is different from the real scene.

9. XSS (Reflected)

About: Reflected XSS attacks, also known as non-persistent attacks, occur when a malicious script is reflected off of a web application to the victim's browser. The script is activated through a link, which sends a request to a website with a vulnerability that enables execution of malicious scripts. The vulnerability is typically a result of incoming requests not being sufficiently sanitized, which allows for the manipulation of a web application's functions and the activation of malicious scripts.

Objective: *Run your own JavaScript to get the cookie and explain how the attack works in the real scene.*

Tips: view backend code.

10. XSS (Stored)

About: Stored XSS (also known as persistent or second-order XSS) arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way. The data in question might be submitted to the application via HTTP requests; for example, comments on a blog post, user nicknames in a chat room, or contact details on a customer order. In other cases, the data might arrive from other untrusted sources; for example, a webmail application displaying messages received over SMTP, a marketing application displaying social media posts, or a network monitoring application displaying packet data from network traffic.

Objective: *Run your own JavaScript to get the cookie and explain how the attack works in the real scene.*

Tips: view backend code.

PS: You can read this for real scene attack.

REPORT REQUIREMENTS:

1. Report Template

NetSec-Lab-Report-Template.doc

2. Language

English

3. Content Highlights

For each of lab requirements, please use screenshots to showcase the correct processes for solving each challenge.

For certain steps, necessary discussions may be provided to demonstrate your understanding.

4. Page Limit

Please keep the report as concise as possible.