Security Lab – Finding and Exploiting Vulnerabilities in a Web Application

VMware

- You will work with the **Ubuntu image**, which you should start in networking-mode **Nat**.

1 Introduction

In this lab, you will search and exploit vulnerabilities in a web application. The application is a simple webshop that was developed by two students during a students project at ZHAW. The students didn't have specific security know how at that time and the project was also a test about how (in)secure such an application turns out without adequate security knowledge.

As expected, the application is full of vulnerabilities and you task is to analyze the application to uncover and exploit them. Try to find as many vulnerabilities as possible and don't be satisfied once you have found the required number to get the lab points (details can be found at the end of this document) because every found vulnerability will help you to get more security aware to avoid such mistakes in your own programs.

You don't receive the source code of the application, but you get some information in the following that will help you to solve this lab in a reasonable time frame.

2 Starting and Stopping the Web Application

Everything you need is on the Ubuntu image. The application is a Java web application based on servlets and JSPs and is deployed on the Tomcat server on the image. Start Tomcat in a terminal as *user* as follows:

- `cd /home/user/tomcat7/bin`
- `./startup.sh`

To terminate Tomcat, do the following in the same directory:

- `./shutdown.sh`

You reach the entry page of the application with a browser using the following URL:

- `http://localhost:8080/Webshop`

The application also contains a secure area (HTTPS). Ignore the certificate warning you'll get when accessing it and don't consider this as a vulnerability, as this is only a testing environment. You can also accept the certificate permanently to get rid of the warning.

3 Description of the Webshop

This section describes some details of the webshop application that will be helpful during your analysis.

3.1 User groups

There are four different user groups:

- **Anonymous users** (not logged in) – they can search for and look at products and register themselves as customers.
- **Customer** (requires login) – they can buy products and rate products.
- **Seller** (requires login) – they can enter new products.

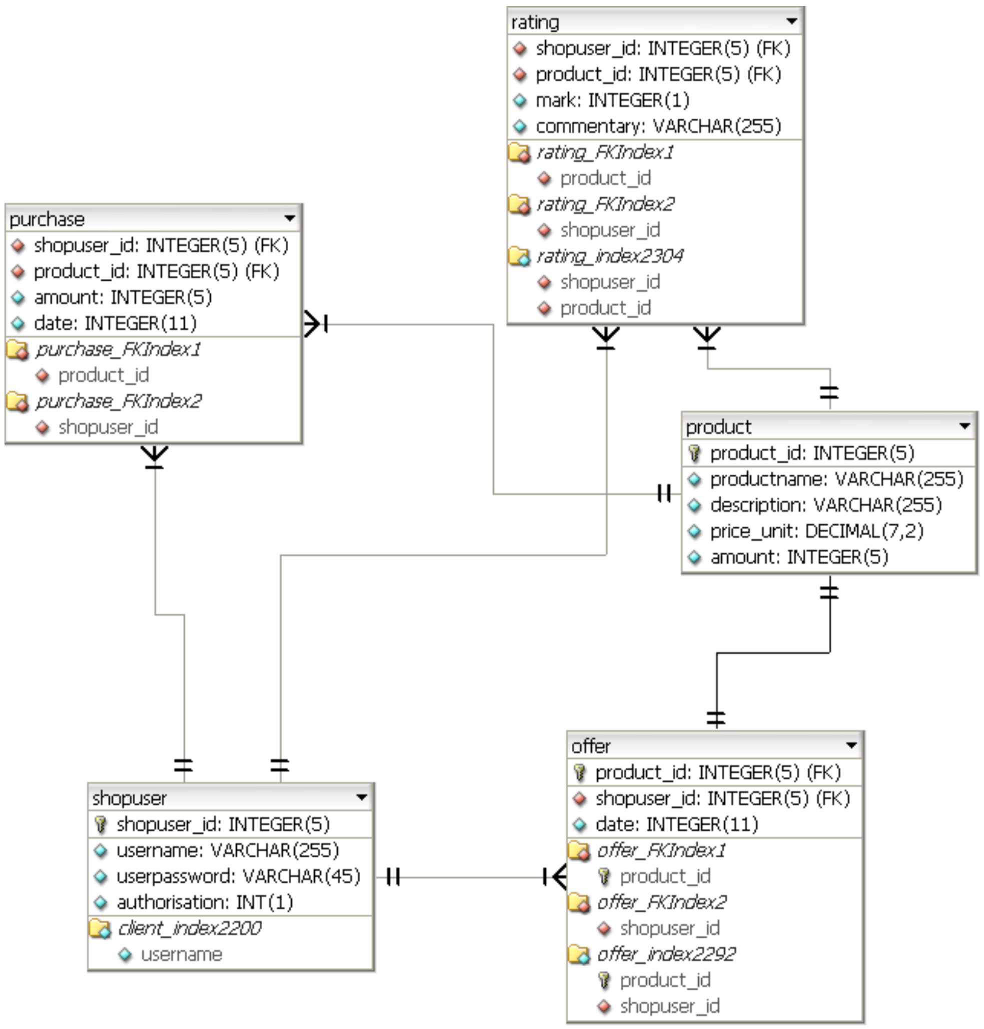
- **Administrators** (requires login) – they can activate customers (a user that registers himself as a customer must always be activated by an administrator before he gets the corresponding rights) and create, edit, and delete customers and sellers.

The following users are available; the password is always the same as the user name:

- 2 activated customers: *customer1*, *customer2*
- 2 non-activated customers: *customer3*, *customer4*
- 2 sellers: *seller1*, *seller2*
- 1 administrator: *admin*

3.2 Database scheme

A MySQL database is used. The scheme is illustrated below and should help you to exploit possibly existing SQL injection vulnerabilities.



The tables contain the following (ignore the indexes):

- All users are stored in table *shopuser*. The attribute *authorisation* contains the user group (0 – 3) according to the list above.
- Table *product* contains the shop products including their price (*price_unit*) and the available quantity (*amount*).

- Table *offer* associates products (table *product*) with a seller (table *shopuser*).
- Table *purchase* gets an entry when a customer buys something. For every product bought, an entry that contains the quantity that was bought (*amount*) is created.
- Table *rating* contains the ratings (*mark*, 1 – 10) that were given by customers (*shopuser_id*) to products (*product_id*).

In addition to the users described above, the tables already contain some test data. If you want to reset these data to the initial state during your tests, you can do this as follows:

- Open the *MySQL Workbench* (Menu *Applications* → *Programming*).
- Double-click on the left to *Local MySQL Server* and enter *root* as password.
- Choose *Open SQL Script...* in the menu *File* and select the file */securitylab/findexploit_webshop/Webshop.sql*.
- Click the *Execute* icon.

4 Hints for Testing

Read the following hints before you start testing:

- First, you should play around with the entire application using different users to understand its functionality.
- The lecture slides contain various hints about how the tests can be performed – use them. If you want to understand in detail how a test you performed worked or didn't work, it is often helpful to study the source code (HTML) of the received page. Security testing has a lot to do with "trying and drawing the right conclusions depending on the behavior of the application".
- The application mostly uses GET requests. This is not to be considered as a vulnerability and was deliberately chosen during application development as it makes testing the application a bit easier (e.g. when inspecting parameters, manipulating parameters, or replying of requests).
- The Firefox add-on *Tamper Data* and *Burp Suite* are installed on the image to support your tests.
    - Tamper Data is started in Firefox via *Tools* → *Tamper Data*. The tool is easily understandable and allows recording, analyzing, and manipulating requests.
    - You should already be somewhat familiar with Burp Suite from the lecture. Start it in a terminal (as *user*): `java -jar /opt/BurpSuite/burpsuite_free_v1.6.jar`. Burp Suite must be used by the browser as a proxy and with the installed Firefox add-on *FoxyProxy*, activating a proxy is easy: Simply select in Firefox *Tools* → *FoxyProxy Standard* the entry *Use Proxy „localhost:8008" for all URLs* (Burp Suite is already configured to listen on port 8008). To stop using the proxy, select *Completely disable FoxyProxy*.

5 Task

Your task is finding and exploiting vulnerabilities in the application. For each vulnerability, you must document the following:

- A **number** to enumerate the vulnerabilities.
- A comprehensible **technical description** of the vulnerability including a proof-of-concept that shows how the vulnerability can be exploited. For instance the injection string with which an SQL injection vulnerability can be exploited or the string with which an HTML injection vulnerability can be demonstrated. A proof-of-concept is good enough, a detailed construction of a realistic and elaborate exploit (e.g. an e-mail message to trick the user in case of an XSS vulnerability) is not required.
- a classification of the attack according to **OWASP Top Ten**[1] (this may not be possible in every case).
- What are the **opportunities for the attacker** that arise from the vulnerability? To be more precise: What attack goals could the attacker achieve by exploiting the vulnerability and how should he proceed in practice to achieve the goal? How will be negatively affected by the attack? Describe at least one realistic scenario.
- Make an assumption about the **implementation error** that was likely made during development such that this attack became possible

The following (fictitious) example illustrates how a vulnerability should be documented:

Vulnerability 1	
Technical description	During login, entering `' OR ''='` for user name and password allows logging in as administrator.
OWASP Top Ten	A1 – Injection
Opportunities for the attacker	The attacker can use the application as administrator without requiring the help of other users. As a result, he can perform all administrative activities, for example: • Deactivating all customers, which corresponds to a DoS attack against the webshop. • Registering as a seller and offering a large amount of products that don't exist. As a consequence, legitimate customers would hardly find real products any more (so it's also a DoS attack), which would significantly reduce the usability and which would result in a significant loss of trust/goodwill in the application.
Implementation error	• Generating SQL queries via string concatenation instead of using prepared statement. • Poor input validation.

[1] https://www.owasp.org/index.php/Top_10_2013

6 Some more Hints...

To get up to speed quickly, some additional hints:

- Try to find vulnerabilities with all types of users.
- There are plenty of vulnerabilities: SQL injection, XSS, HTML injection, session management issues, CSRF, access control problems, parameter tampering, information disclosure...
- The method to search for products looks as follows in the source code:

```
public ResultSet searchProducts(String searchTerm)
                 throws ServletException {
    try {
        Statement stmt = connection.createStatement();
        rs = stmt.executeQuery("SELECT * FROM product WHERE
             productname LIKE '%" + searchTerm + "%'");
        return rs;
    } catch (SQLException exc) {
        throw new ServletException("SQL-Exception", exc);
    }
}
```

- Sometimes, the result of an attack attempt is not directly visible. Maybe it's possible a user can place a Javascript somewhere, which then attacks another user (possibly from another user group) "at another place within the application".
- Try to submit requests that "should be available" only to one specific user group as a user of another user group. You never know...

7 Found Vulnerabilities

Document the found vulnerabilities on the following pages. You can also document your answers in another way, but it's important that your answers basically follow the given scheme above. Document all vulnerabilities you find, including the ones that may not be too critical from your point of view. If you find bugs that are not security-relevant, just ignore them.

Lab Points

For **4 lab points**, you must document at least six fundamentally different vulnerabilities according to the scheme above and show your answers to the instructor. "Fundamentally different vulnerabilities" means that for instance, two different XSS vulnerabilities only count as one answer. Likewise, a found vulnerability (e.g. with a specific parameter) may only be used for one attack and not, e.g., for XSS and HTML injection. SQL injection attacks may be used in two answers if one allows reading data (SELECT...) and one allows writing data (e.g. INSERT...).

Vulnerability 1	
Technical description	
OWASP Top Ten	
Opportunities for the attacker	
Implementation error	

Vulnerability 2	
Technical description	
OWASP Top Ten	
Opportunities for the attacker	
Implementation error	

Vulnerability 3	
Technical description	
OWASP Top Ten	
Opportunities for the attacker	
Implementation error	

Vulnerability 4	
Technical description	
OWASP Top Ten	
Opportunities for the attacker	
Implementation error	

Vulnerability 5	
Technical description	
OWASP Top Ten	
Opportunities for the attacker	
Implementation error	

Vulnerability 6	
Technical description	
OWASP Top Ten	
Opportunities for the attacker	
Implementation error	

Vulnerability 7	
Technical description	
OWASP Top Ten	
Opportunities for the attacker	
Implementation error	

Vulnerability 8	
Technical description	
OWASP Top Ten	
Opportunities for the attacker	
Implementation error	

Vulnerability 9	
Technical description	
OWASP Top Ten	
Opportunities for the attacker	
Implementation error	

Vulnerability 10	
Technical description	
OWASP Top Ten	
Opportunities for the attacker	
Implementation error	