Vulnerability Assessment and Systems Assurance Report

Tune Store Phase 1

Sahil Bhirud

ITIS 4221/5221

September, 2020

VULNERABILITY ASSESSMENT AND SYSTEM ASSURANCE

TABLE OF CONTENTS

1.0 GENERAL INFORMATION

1.1 Purpose

The objective of Tune Store Phase 1 is to perform penetration testing on the online music store application. The vulnerabilities which need to be addressed are SQL injection and Cross Site Scripting (XSS).

1.2 Scope

The penetration testing performed was focused on identifying 3 SQL vulnerabilities and 2 XSS vulnerabilities in the online music store application.

1.3 System Overview

The website is an online music store web application. This application, named Tune Store, has 14 use cases: Login, Logout, Register user, View profile, Change password, Add balance to account, View friends, Add a friend, View CDs, View CD comments, Buy a CD, Download a CD, Give CD as gift to friends.

1.4 Project References

1. Stuttard, Dafydd., and Marcus. Pinto. *The Web Application Hacker's Handbook Finding and Exploiting Security Flaws*. 2nd ed. Indianapolis: Wiley, 2011. Print.
2. https://searchsoftwarequality.techtarget.com/definition/SQL-injection#:~:text=A%20SQL%20injection%20(SQLi)%20is,be%20performed%20on%20a%20database.
3. https://www.ptsecurity.com/ww-en/analytics/knowledge-base/how-to-prevent-sql-injection-attacks/#2
4. https://portswigger.net/web-security/cross-site-scripting

1.5 Acronyms and Abbreviations - If you use them then put them here.

1.6 Point of Contact

Sahil Bhirud – sbhirud2@uncc.edu

2.0 VULNERABILITIES DISCOVERED

2.1 SQL Injection – Login as a random user

2.1.1 Vulnerability Description and Impact

A SQL Injection is a type of security exploit in which the attacker adds Structured Query Language (SQL) code to a Web form input box in order to gain access to unauthorized resources or make changes to sensitive data. An SQL query is a request for some action to be performed on a database. When executed correctly, a SQL injection can expose intellectual property, the personal information of customers, administrative credentials, or private business details.

In this case, the login form can be bypassed using SQL Injection without authenticating the user which can have a huge impact on the application as the unauthorized user will be able to access private and confidential information of a legit user.

2.1.2 Description of exploits used

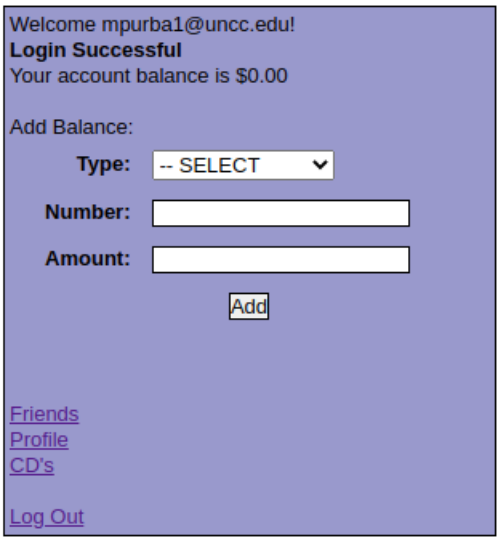
The vulnerability is located on the login form in the password field. The query which was used to detect this vulnerability is **‘ OR ‘a’=’a**

This query is closing the password field and at the same time setting a condition a=a which will always be true and hence the website is allowing the unauthorized user to access the first database entry of the users.

2.1.2.1 Exploit example

After loading the Tune Store web page, I entered the attack query **‘ OR ‘a’=’a** in the password field of the login page.

And this was the result:



2.2 SQL Injection – Login as a specific user

2.2.1 Vulnerability Description and Impact

Assuming the attacker has the username of any one of the users, he can bypass the Tune Store authentication/login form without entering the user's password.

This can result in a leak of credit card details of the user, loss of control of the account (if the attacker changes the password), spam messages or malicious links in the comment section which could lead to an additional attack of phishing on other users.

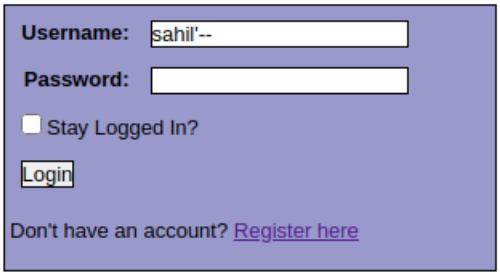
2.2.2 Description of exploits used

The vulnerability is located on the login form in the username field. The query which was used in detecting the vulnerability is **sahil' --** (generic form: username'--).

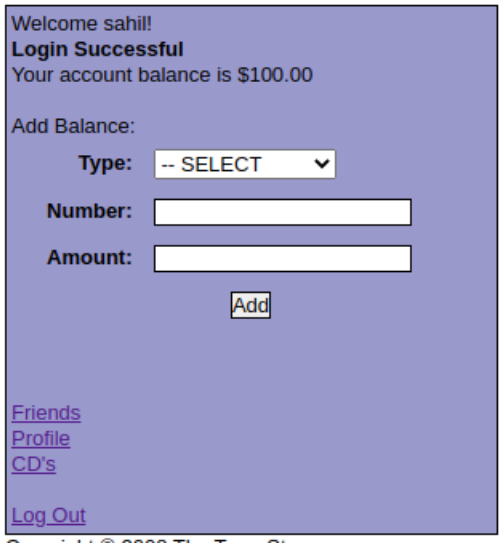
This query takes input the username and then closes the username field using the apostrophe (‘). The two hyphens (--) comment out the rest of the database query in the back end due to which the password of the user is not verified, allowing the attacker to pass through the authentication process with ease.

2.2.2.1 Exploit example

After loading the Tune Store web page, I injected the query in the username field.



And when I tried to log in with only the username field filled, this is the result I got:



I got into the account without entering the password.

2.3 SQL Injection – Register a new user with lots money in account without paying for it

2.3.1 Vulnerability Description and Impact

Using SQL Injection, an attacker can register a new user with a lot of money in the account without actually paying for it.

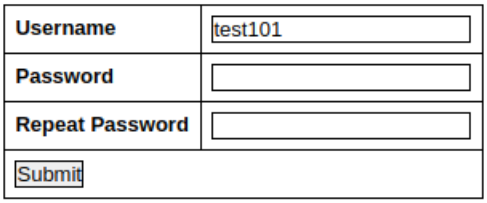
This can cause the company in losing money since the attacker is not adding balance through his credit/debit card but through manipulation of the SQL query.

2.3.2 Description of exploits used

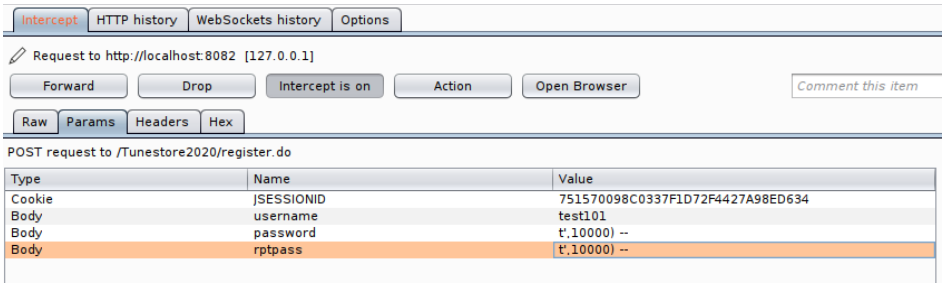
The vulnerability is located in the new user registration form. The query which was used to detect the vulnerability is **t',10000) --** where **t** is the password and **10000** is the amount which is being added to the balance of the user and the two hyphens are commenting out the rest of the original query where the default balance is set to 0.

2.3.2.1 Exploit example

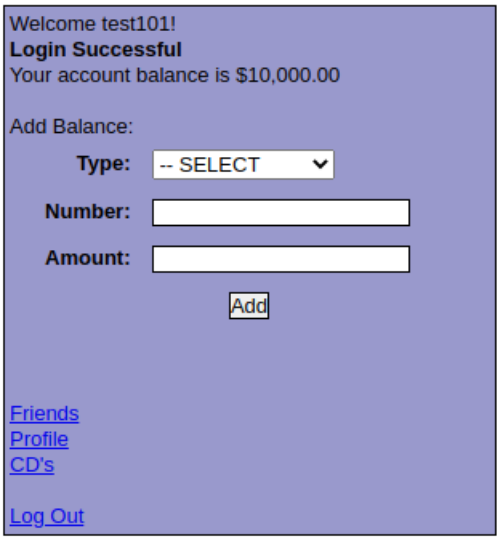
After opening the new user registration web page, I started Burp Suite to intercept requests going from the Tune Store page to its server. I entered only the username on the form and submitted it.



Burp Suite intercepted this request and I manipulated the password and repeat password fields with the query mentioned above.



New user was successfully registered and when I logged in with the credentials of the new user, I had a balance of $10,000 which I had mentioned in the query.



2.4 Reflective XSS

2.4.1 Vulnerability Description and Impact

Reflective XSS occurs when user input is immediately returned by

a web application in an error message, search result, or any other response that includes some or all of the input provided by the user as part of the request.

This vulnerability can lead to user accounts being hijacked, credentials being stolen, sensitive data being exfiltrated, and lastly, access to the client computers could be obtained.

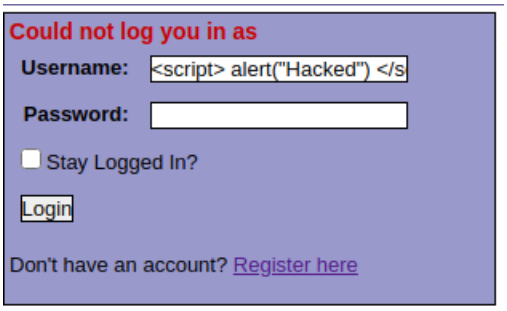
2.4.2 Description of exploits used

On the login page, if a failed login occurs, it displays back to the user, the username that had a failed login. This was a perfect place to look for an XSS exploit.

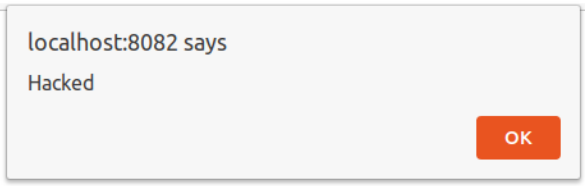
I used a normal *alert* script to check for the vulnerability. The script was:
<script>alert("Hacked")</script>. In this JavaScript, **<script>** is the script tag which indicates beginning of the JavaScript and **alert("Hacked")** is the message which will be popped up after execution of the script.

2.4.2.1 Exploit example

After loading the Tune Store web page, I entered the script in the username field and clicked on Login.



As expected, the webpage popped up the message mentioned in the script.



2.5 Stored XSS

2.5.1 Vulnerability Description and Impact

Stored XSS generally occurs when user input is stored on the target server, such as in a database, in a message forum, visitor log,

comment field. It occurs when a malicious script is injected directly into a vulnerable web application.
The nature of stored cross-site scripting exploits is particularly relevant in situations where an XSS vulnerability only affects users who are currently logged in to the application.

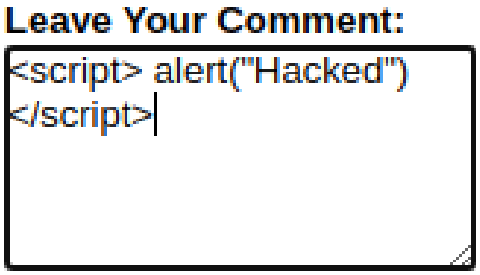
This vulnerability can lead to user accounts being hijacked, credentials being stolen, sensitive data being exfiltrated, and lastly, access to the client computers could be obtained.

2.5.2 Description of exploits used

The location of the vulnerability is the comment section of the web application as that section allows users to enter text that will be displayed back to other users. The script which I used to detect the XSS vulnerability is ***<script>alert("Hacked")</script>***

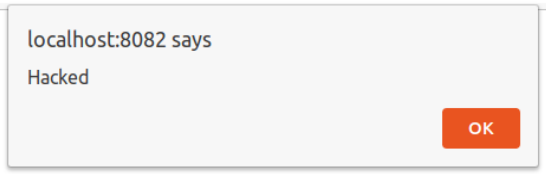
2.5.2.1 Exploit example

After logging in the Tune Store account, I went to the comment section where I inserted the script.



It doesn't show anything in the comment as the script which I entered became a part of the code of the webpage.

Whenever the page is reloaded, the script is executed.



3.0 MITIGATION RECOMMENDATIONS

3.1 For SQL Injection

3.1.1. SQL Parametrization

Parameterized queries are a means of pre-compiling a SQL statement so that you can then supply the parameters in order for the statement to be executed. This method makes it possible for the database to recognize the code and distinguish it from input data.

3.1.2. Input Validation

Input Validation checks if the user's input is allowed or not. Input validation makes sure it is the accepted type, length, format, etc. Only the value which passes the validation can be processed.

3.2 For Cross Site Scripting (XSS)

3.2.1. Output Encoding

It is the process of converting untrusted data into a secure form where the input is visible to the user without executing the code in the browser.

3.2.2. Input Validation

At the point where user input is received, filter as strictly as possible based on what is expected or valid input.