ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
TECHNICAL UNIVERSITY OF CRETE

Ασφάλεια Συστημάτων και Υπηρεσιών.

(HPY 413)

LAB 3

Ομάδα Χρηστών 42

Κωνσταντίνος Νικολός (2019030096)

Αλέξανδρος Νικόλαος Κουφούδης (2020030123)

Introduction:

This assignment was to perform a basic form of SQL injection attack and identify some vulnerabilities on a mock server.
The server at first prompts the user for a password, then once logged in allows him to search for some items in the database. There is also an admin login page.
The first goal was to bypass the first login prompt as user, without knowing the user's password, to use the "Search box" to retrieve the administrator's password. Since we find the admin password, we can just log in as administrator and reveal the flag.
The second goal was to identify and exploit some vulnerabilities, more specifically a reflected cross-site scripting (XSS) and a DOM-XSS vulnerability. Reflected XSS aims to embed client-side data to the server-side code in HTML documents, while in DOM-based XSS, the malicious payloads are referenced and executed on the client-side (browser) environment.

The process of finding the flag

Bypassing the login page:
We entered to the "Enter User's password" box the following SQL code: *' OR 1=1 --*
By reading the app.py file that was given, we noticed in line 41, that if we nest the above SQL code, we can get access to user's page.

- The first single quotation mark (') is used to exit the one that is open for the password to be placed inside

- Then we used an OR condition that always evaluates to true (1=1). This means that query returns something, but not NULL, cause this is what the code checks in line 47 of the app.py file.

- At the end we added the comment characters (--) in order to comment out the following code. We need to make sure that query ends up to our always true statement. If we omit the comment characters an error would be caused because of the trailing quotation mark.

Retrieving the admin password

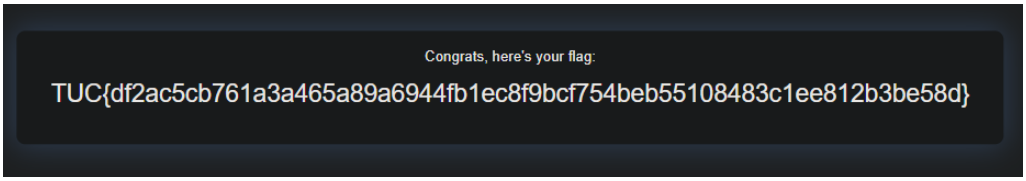
The first thought was to try the same SQL code to admin's page. However this didn't work, cause now the password is grabbed from the request (line 107 app.py file).
So we tried to find the password though user's page, by using the Search Box.
By reading again the app.py file, we noticed that in line 82, we can again nest SQL code, that will be the ''item'' to search for. So we entered the following code to the box:
' UNION SELECT id, username, password from users WHERE username = 'superadmin' –

- we use the first single quotation mark to exit the open one, just like above

- Then we union the item's table with users' table. As we know from SQL, when combining two tables using a UNION , they must have the same number of columns. The columns do not need to have the same name, because they are combined based on column positions rather than the column names. However, they should have similar data types. In line 82, we can easily see that we have a table of 3 rows (name, category, price) and we just union this table-query with the actual query we are interested in. As a result, we get the admin's password.
  Note: If no ''WHERE'' clause was specified the result would be the same, as the first resulting tuple would be the one with id=1, which is the admin account.

- Again, we comment out the rest of the code as we did in order to bypass the login page.

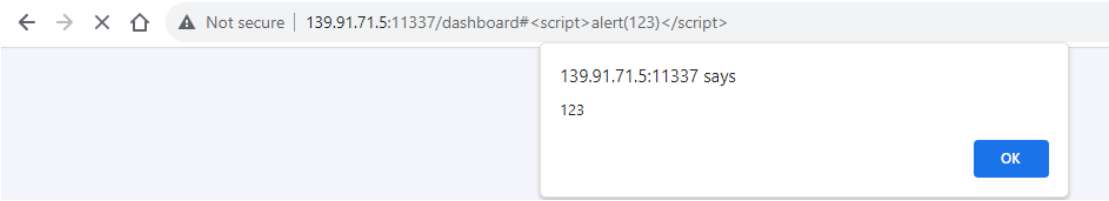
Now by just typing the password that we received ($thisIsUncrackable$) to the administrator's login page, we get the flag that we were searching for:



Identifying and exploiting the reflected XSS and DOM-XSS vulnerability

- Reflected XSS vulnerability

XSS Reflected vulnerability can be found in the JavaScript code executed when the user enters the /dashboard html page. In lines 3,4 of greet.js file code gets the username from the URL without any sanitation , it just decodes it which doesn't apply any filter for <script> and etc. The payload is sent to server as a request, server does not filter it as seen in the code and then sends it back to the client as a response resulting in a 'Reflected Attack'.



- DOM-XSS vulnerability

Dom Xss vulnerability can be found in the next line of the same file (line5). Document.write is used and the input of this document.write is given again by the URL, (#..) which is treated as a fragment resulting in the browser to not forwarding it back to the server. The browser updates the DOM (the html body of the page) to contain Welcome <script> … </script> , then the browser finds the script and executes it.

Scripts Used:

http://139.91.71.5:11337/dashboard#%3Cscript%3E%20var%20navAgt%20=%20navigator.userAgent;%20if%20(navAgt.indexOf(%22MSIE%22)!=-1)%20{%20document.write(%22DOM-XSS%20Vulnerabillity%20Detected%20%20:%20%22%20+%20document.location.href%20+%

20%22.%22);%20}%20else%20{%20document.write(%22DOM-%20XSS%20Vulnerabillity%20Detected.%22);%20}%20%3C/script%3E

http://139.91.71.5:11337/dashboard#%3Cscript%3Ealert(123)%3C/script%3E