NSA Codebreaker 1-5 Write-up 2019

Task 1

For the first challenge we are given a pcap file to do some analysis on. The solution to this challenge was quite easy. First I opened the file in wireshark, then used the File -> Export Objects -> HTTP to export all objects from the pcap. Looking at the files from my terminal, I could see that it included the APK and registration files. To hash the apk I did:

```
chase@chase:~/codebreaker$ sha256sum terrorTime.apk
6bdec1403b2338312c0518cba6bff4ad3280360aadd0200256818003b0776f73  terrorTime.apk
```

There was also a README.md file, that included the following information.

```
terrorTime.apk -- Most current version of terrortime APK for android mobile devices
faith--vhost-2961@terrortime.app -- First Terrortime test account client id
veronica--vhost-2961@terrortime.app -- Second Terrortime test account client id
MJu2wXGSXtMJ4V -- First Terrortime test account client secret
Nfy8UdyDdBlcli -- Second Terrortime test account client secret
```

Submitting this information was all that was needed to solve the first challenge.

Task 2

The TerrorTime APK file contains metadata that describes some security properties of the application. In this task we are asked to use the apk file to find the following information:

1. App Permissions
2. The SHA256 hash of the Code Signing Certificate
3. The Common Name of the Certificate Signer

The process of identifying this information is as follows.

To get the app permissions we could use the aapt utility on the apk. The command is shown below.

```
chase@chase:~/codebreaker/solutions$ aapt d permissions
~/codebreaker/terrorTime.apk
package: com.badguy.terrortime
uses-permission: name='android.permission.INTERNET'
uses-permission: name='android.permission.ACCESS_NETWORK_STATE'
```

So our permissions are INTERNET and ACCESS_NETWORK_STATE. The second two parts could be solved using apksigner as shown below.

```
chase@chase:~/codebreaker$ apksigner verify --print-certs terrorTime.apk
Signer #1 certificate DN: CN=dev_terrorTime_185031, OU=TSuite
Signer #1 certificate SHA-256 digest:
2ea265ad5d04530fc8f3b95c58f0f7747347441e9ffe5051e71cab5fee7b91d0
Signer #1 certificate SHA-1 digest: 8301354f25e1fa5db9e0ca96fd00039f430c0688
Signer #1 certificate MD5 digest: 07b4e1cadb13b62cd04d3a2557656380
```

This command dumps both the sha256 hash 2ea265ad5d04530fc8f3b95c58f0f77473... and information about the name: dev_terrorTime_185031. This information was enough to solve the second challenge.

Task 3

The third challenge was even easier than the previous 2. For this one we need only identify the IP addresses of the OAUTH and XMPP servers. To get the addresses for the servers we need to open the database with sqlite3, as shown below.

```
clientDB.db  m-pin.txt  pin2.txt  pin3.txt  pin.txt  README.developer  solutions/
terrorTime.apk  terrorTimeExtract/
chase@chase:~/codebreaker$ sqlite3 clientDB.db
SQLite version 3.29.0 2019-07-10 17:32:03
Enter ".help" for usage hints.
sqlite> from Clients select *
   ...>
   ...> ;
Error: near "from": syntax error
sqlite> select * from Clients;
annabelle--vhost-2961@terrortime.app||annabelle--vhost-2961@terrortime.app|chat.ter
rortime.app
...
```

The important bits of information are the chat.terrortime.app and register.terrortime.app. The next step is to find the addresses of these, which can be done with dig as shown below.

```
chase@chase:~/software/android-studio/bin$ dig chat.terrortime.app

; <<>> DiG 9.11.5-P4-5.1ubuntu2.1-Ubuntu <<>> chat.terrortime.app
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 18376
```

```
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;chat.terrortime.app.            IN     A

;; ANSWER SECTION:
chat.terrortime.app.      696    IN     A      54.91.5.130

;; Query time: 0 msec
;; SERVER: 127.0.0.53#53(127.0.0.53)
;; WHEN: Thu Dec 05 20:38:24 EST 2019
;; MSG SIZE  rcvd: 64
```

And the second one below.

```
chase@chase:~/software/android-studio/bin$ dig register.terrortime.app

; <<>> DiG 9.11.5-P4-5.1ubuntu2.1-Ubuntu <<>> register.terrortime.app
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 57920
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;register.terrortime.app. IN     A

;; ANSWER SECTION:
register.terrortime.app. 642     IN     CNAME  codebreaker.ltsnet.net.
codebreaker.ltsnet.net.   107    IN     A      54.197.185.236

;; Query time: 2 msec
;; SERVER: 127.0.0.53#53(127.0.0.53)
;; WHEN: Thu Dec 05 20:39:16 EST 2019
;; MSG SIZE  rcvd: 104
```

So then we have the IP addresses 54.91.5.130 and 54.197.185.236, which are the solutions to the task.

Task 4

This task required a bit more work than the previous ones. The first step is to open the database and dump some of the information to a pin file.

```
chase@chase:~/codebreaker/solutions$ sqlite3 clientDB.db
sqlite> .mode ascii
sqlite> .output pin3.txt
sqlite> select checkpin from Clients;
sqlite> .output stdout
sqlite>
```

This should have dumped the hashed pin information to the pin3.txt file. First I used the xxd command to dump the hex data and filtered this so I had a hex string that corresponded to the hash. Since we know this has been hashed using the sha256 algorithm, we can now use hashcat to crack the hash as shown below.

```
hashcat -m 1400 -a 3 -i --increment-min 6 --increment-max 6 m-pin.txt
“?d?d?d?d?d?d”
```

This gave us the pin 578356, which we know goes with user annabelle--vhost-2961@terrortime.app, which can be used to log into the app.

Task 5

For this task we had to obtain messages from the lead terrorist. I already have the following password.

```
578356
```

First I needed to set up the clientDB.db to use the lead terrorist username. After opening it with sqlite3, I ran the following command.

```
UPDATE clients SET xname="sebastian--vhost-2961@terrortime.app" WHERE
cid="annabelle--vhost-2961@terrortime.app";
```

Next I needed the data dump associated with the login. To accomplish this, I used a frida script, which is listed below.

```
import frida, sys
from time import sleep
script = """
    console.log("[*] Starting instrumentation");
    Java.perform(function() {
        console.log("[*] Entering");
        var bClass = Java.use("com.badguy.terrortime.Client");
```

```
        var cClass = Java.use("java.util.Arrays");

bClass.decryptMessage.overload('com.badguy.terrortime.Message').implementation =
function(v) {
                console.log(cClass.toString(v.getContent()));
            }
            console.log("[*] Hook installed")
    });
"""
device = frida.get_usb_device()
pid = device.spawn("com.badguy.terrortime")
session = device.attach(pid)
script = session.create_script(script)
script.load()
device.resume(pid)
sys.stdin.read()
```

After running this and logging in with the credentials captured in the previous task, we get a relatively large data dump, the beginning of which is shown below.

```
[123, 34, 109, 101, 115, 115, 97, 103, 101, 75, 101, 121, 34, 58, 123, 34, 81, 71,
51, 99, 88, 85, 116, 57, 111, 51, 119, 99, 55, 111, 81, 90, 76, 102, 70, 75, 69,
73, 57, 49, 99, 86, 54, 118, 117, 121, 120, 72, 71, 119, 73, 112, 106, 115, 89, 43,
79, 71, 52, 61, 34, 58, 34, 78, 119, 110, 52, 114, 55, 81, 90, 77, 98, 51, 48, 66,
51, 110, 74, 111, 115, 43, 49, 112, 97, 106, 82, 75, 107, 104, 50, 71, 103, 103,
119, 78, 118, 81, 66, 57, 78, 50, 108, 76, 71, 81, 82, 76, 86, 117, 74, 48, 110,
121, 74, 83, 48, 80, 110, 49, 90, 109, 115, 69, 100, 99, 72, 68, 114, 82, 88, 114,
...
```

It looks like these numbers correspond to ascii characters, which translates to the message below.

```
{"messageKey":{"QG3cXUt9o3wc7oQZLfFKEI91cV6vuyxHGwIpjsY+OG4=":"CQnpowRFRHSQUDNowDQN
MSj/kYEzd3byJ3JFw8pob0FpyFXd1PMgLPYilJtwUxOhxvuZ9EBE9YgYcmrferLfXz25NsRfTZ47ZdVio4V
JL1A+tEqb+ng9sWtcn4vjjUafSikQw8jBUpINFricPcC2DJSlxmnU5GRKIs8G7rchma+KVIk/36TuVcMFE0
fAfnpop+PNXZTgfYGdxLm1ZJZDKkhikdN98vW35HaHi+PSL57Ya7DPiIc3JAhO/lhGFd3SyU/aP+Hb7uHBL
qC6MZXGxDmJo7ZkbbZSA1KaBiPbcAhj6hPNww2n1amKvC+8mg5rvFEBxOA2VE2q81OEnb9ZMA==","MCyIJ
tm05htTwbXdS2wsP+j5OWa6lqJKzhOpyGHOtX4=":"HnEWo7hbjheN54eZORHDexfoxFxw0o7iJgY2To3TG
zZ8pHdgS9ewLfm9oJ7X0MQotehuCiQU+XKn4ZGC1NHzeOjHdWvjl8b+myWiUpONJsP3ni7qZP1951wkYJQs
vMJBpjNVQdgfPNygkn6h3ztM4cOj5oIE2dIRuMsXrvy1ne21EvhlMg87crnz934j939ET3m8G5cYfdWRJEK
4NJHy/xVlSJUTE+A3prVmiWDDOFGvfuMUBijlcjPNlhXPZC7jz5zXK0285hdpuOtq2pzIrl5TJaIJN9cB4k
3wOKUVT4TJGUsfr7KksORRUTZRpEK2KY8IRTS0DVgMIF/01+Sj8g=="},"messageSig":"CTbq+8LhHGZM
zguhskEszjydOKl5HmFTCFcYY286kM0=","message":{"msg":"WIr0hkSesVRRtR7siicp+h0qpWBzzML
z3jDl3ITyUilyMV2sV84aD35WD5O8dq+rs0Kx09rUsq03fIi0hnAng6pwUprqsBZhCj0VVA/9nEyJ9R0kDw
6w4uLhzH8x5XvjuQCy+/KeMP0OTkGPmqqYb8Bt2gelItkun3wPSzGtInrDvTbDA6pm/AedbAe7/egNHDD7c
6SFtVUqjDmW2vuUFk2y06ciAB5pQT9K0JK2yMtlAzQi4KezRdRb9JjOO6nZjp2KCcylVpDEZDYWv9wwJA==
","iv":"Si3n4zGZuzWx2bA+IzXQww=="}}
```