EMBEDDED CRYPTO

Author: CONSORTIUM

[Publish Date]

European Cyber Security Challenge

1.Initial Write-Up

Description

An embedded device serves as an electronic bank teller. You’ve managed to get your hands on a dump of the flash attached to it and the communication performed against the back-office.

Can you get the secret account number?

2. Challenge specifications

- Category: Network/Forensics/Cryptography
- Difficulty : Easy/Medium
- Estimated time: 2h – 3h

3.Questions and answers

What is the flag ?

CTF{1C6683F328F32D1016FA055C32F8917BD706F9A870600B515B9AAB5E801C84C5}

What is the string for SHA256SUM?

SHA256SUM over a random set of data

4.Artefact hashes

FILES	MD5	SHA256
capture.bin	F82D45F16249707D46044477E8CF50F4	1551D5DBC06224298B2E4DB1B0E622482C7D33B918C0861E7739DE635B2092FE
dump.bin	4814FFC8746EEAC485E20AEB7FFE35DC	A062506BDDD1B6BFEFD1FD1DD2BAF6AF13C4A069571F4F77B01F6982D75B8ABA

5.Tools needed

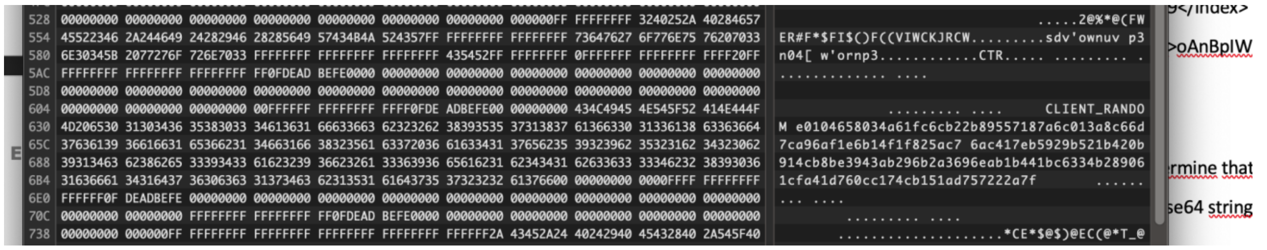
- Python3
- CTF CryptoTool (writen in Python) - https://github.com/karma9874/CTF-CryptoTool
- Wireshark
- AES crypto tools (OpenSSL, online, Python, etc.)

6.Walkthrough (writeup)

The challenge is based on a network capture file and a memory dump.

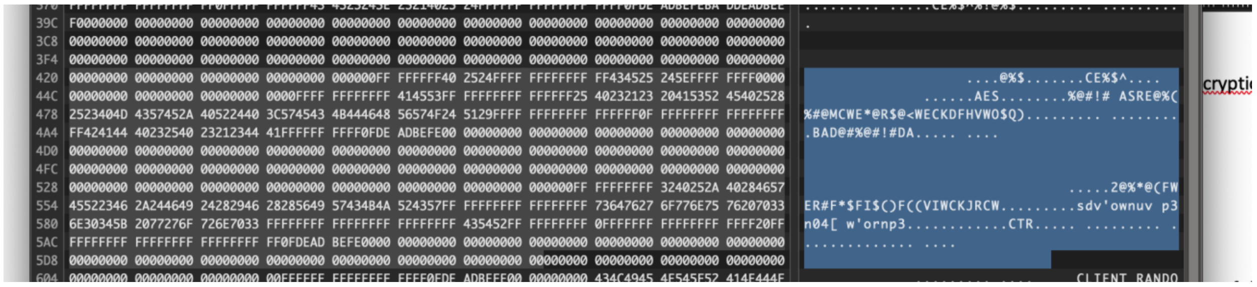
The capture file contains one encrypted (TLS) communication stream. The memory dump contains several hints and required information for decrypting the stream and cipher.

Master keys for TLS decryption:



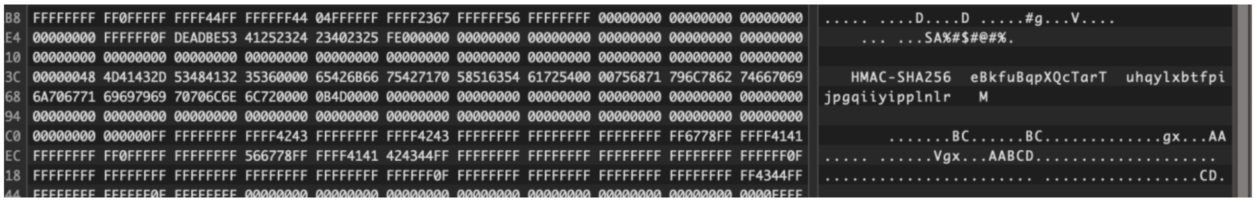
```
528 00000000 00000000 00000000 00000000 00000000 00000000 00000000 000000FF FFFFFFFF 3240252A 40284657              .....2@%*@(FW
554 45522346 2A244649 24282946 28285649 57434B4A 524357FF FFFFFFFF FFFFFFFF 73647627 6F776E75 76207033 ER#F*$FI$()F((VIWCKJRCW.........sdv'ownuv p3
580 6E30345B 2077276F 726E7033 FFFFFFFF FFFFFFFF FFFFFFFF 435452FF FFFFFFFF 0FFFFFFF FFFFFFFF FFFF20FF n04[ w'ornp3............CTR..... ......... .
5AC FFFFFFFF FFFFFFFF FFFFFFFF FF0FDEAD BEFE0000 00000000 00000000 00000000 00000000 00000000 00000000 ............. ....
5D8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
604 00000000 00000000 00000000 00FFFFFF FFFFFFFF FFFF0FDE ADBEFE00 00000000 434C4945 4E545F52 414E444F           ......... ....     CLIENT_RANDO
630 4D206530 31303436 35383033 34613631 66633663 62323262 38393535 37313837 61366330 31336138 63363664 M e0104658034a61fc6cb22b89557187a6c013a8c66d
65C 37636139 36616631 65366231 34663166 38323561 63372036 61633431 37656235 39323962 35323162 34323062 7ca96af1e6b14f1f825ac7 6ac417eb5929b521b420b
688 39313463 62386265 33393433 61623239 36623261 33363936 65616231 62343431 62633633 33346232 38393036 914cb8be3943ab296b2a3696eab1b441bc6334b28906
6B4 31636661 34316437 36306363 31373463 62313531 61643735 37323232 61376600 00000000 0000FFFF FFFFFFFF 1cfa41d760cc174cb151ad757222a7f       ......
6E0 FFFFFF0F DEADBEFE 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 ... ....
70C 00000000 00000000 FFFFFFFF FFFFFFFF FF0FDEAD BEFE0000 00000000 00000000 00000000 00000000 00000000       ......... ....
738 00000000 000000FF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFF2A 43452A24 40242940 45432840 2A545F40      ....................*CE*$@$)@EC(@*T_@
```

Hints about the encryption used:



```
39C F0000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  .
3C8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
3F4 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
420 00000000 00000000 00000000 00000000 000000FF FFFFFF40 2524FFFF FFFFFFFF FF434525 245EFFFF FFFF0000                      ....@%$.......CE%$^....
44C 00000000 00000000 00000000 0000FFFF FFFFFFFF 414553FF FFFFFFFF FFFFFF25 40232123 20415352 45402528                 ......AES........%@#!# ASRE@%(
478 2523404D 4357452A 40522440 3C574543 4B444648 56574F24 5129FFFF FFFFFFFF FFFFFF0F FFFFFFFF FFFFFFFF  %#@MCWE*@R$@<WECKDFHVWO$Q)......... ........
4A4 FF424144 40232540 23212344 41FFFFFF FFFF0FDE ADBEFE00 00000000 00000000 00000000 00000000 00000000  .BAD@#%@#!#DA..... ....
4D0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
4FC 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
528 00000000 00000000 00000000 00000000 00000000 00000000 00000000 000000FF FFFFFFFF 3240252A 40284657                               .....2@%*@(FW
554 45522346 2A244649 24282946 28285649 57434B4A 524357FF FFFFFFFF FFFFFFFF 73647627 6F776E75 76207033  ER#F*$FI$()F((VIWCKJRCW.........sdv'ownuv p3
580 6E30345B 2077276F 726E7033 FFFFFFFF FFFFFFFF FFFFFFFF 435452FF FFFFFFFF 0FFFFFFF FFFFFFFF FFFF20FF  n04[ w'ornp3............CTR..... ......... .
5AC FFFFFFFF FFFFFFFF FFFFFFFF FF0FDEAD BEFE0000 00000000 00000000 00000000 00000000 00000000 00000000  ............. ....
5D8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
604 00000000 00000000 00000000 00FFFFFF FFFFFFFF FFFF0FDE ADBEFE00 00000000 434C4945 4E545F52 414E444F                            CLIENT RANDO
```

Information required for decryption:



```
B8 FFFFFFFF FF0FFFFF FFFF44FF FFFFFF44 04FFFFFF FFFF2367 FFFFFF56 FFFFFFFF 00000000 00000000 00000000  ..... ....D....D .....#g...V....
E4 00000000 FFFFFF0F DEADBE53 41252324 23402325 FE000000 00000000 00000000 00000000 00000000 00000000      ... ...SA%#$#@#%.
10 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
3C 00000048 4D41432D 53484132 35360000 65426B66 75427170 58516354 61725400 00756871 796C7862 74667069     HMAC-SHA256  eBkfuBqpXQcTarT  uhqylxbtfpi
68 6A706771 69697969 70706C6E 6C720000 0B4D0000 00000000 00000000 00000000 00000000 00000000 00000000  jpgqiiyipplnlr   M
94 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
C0 00000000 000000FF FFFFFFFF FFFF4243 FFFFFFFF FFFF4243 FFFFFFFF FFFFFFFF FFFFFFFF FF6778FF FFFF4141         .......BC......BC..............gx...AA
EC FFFFFFFF FF0FFFFF FFFFFFFF 566778FF FFFF4141 424344FF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFF0F  ..... ......Vgx...AABCD....................
18 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFF0F FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FF4344FF  ...................... ..................CD.
44 FFFFFFFF FFFFFF0F FFFFFFFF 00000000 00000000 00000000 00000000 00000000 00000000 00000000 0000FFFF
```

After the decryption of tls stream will result an http stream with the following data:

```
<html>
        <head>
        </head>
        <body>
        <data>idi9cwo34xQ1621MbvwFAW9PqLXzgv9jKn2JyW7VACYlc3Rs61boVd7Wsjg6rBG2dkdttpv/
MkvPUpBBm3O9WGCod9EC</data>
        <index>199</index>
        <startwith>oAnBpIWRLGrmMNPnRCQLBA==/<startwith>
        </body>
</html>
```

We can easily determine that data and startwith is information base64 encoded.

We decode the base64 strings and get some binary ones.

Data:89d8bd730a37e31435eb6d4c6efc05016f4fa8b5f382ff632a7d89c96ed500262573746ceb56e855de d6b2383aac11b676476db69bff324bcf5290419b73bd5860a877d102

Startwith:A009C1A485912C6AE630D3E744240B04

From the memory dump we can see something which resembles to pbdkf2 parameters :

HMAC-SHA256, pass, salt, iterations.

Using Python (or online) we can derive the AES key used for encryption.

```
>>> key=hashlib.pbkdf2_hmac('sha256' ,b'eBkfuBqpXQcTarT',b'uhqylxbtfpijpgqiiyipplnlr',19723)

>>> binascii.hexlify(key)
```

Output: b'fb0b74c930b714282baba01046ca11b42f3faccb056e6e60c25113c5976a9a60'

Here is a trick, the memory contains the value for iterations in little endian format so 0b4d is 19723 and not 2893 as this value would normally decode.

We now have the key, we have used startwith word in the html for the IV(should be clear for someone who understands crypto), data contains the encrypted key.

We perform AES-CTR decryption (online or with Python) and get the flag.

7. References

https://cryptii.com/pipes/aes-encryption

https://docs.python.org/3/library/hashlib.html

https://www.base64decode.org/

https://www.comparitech.com/net-admin/decrypt-ssl-with-wireshark/