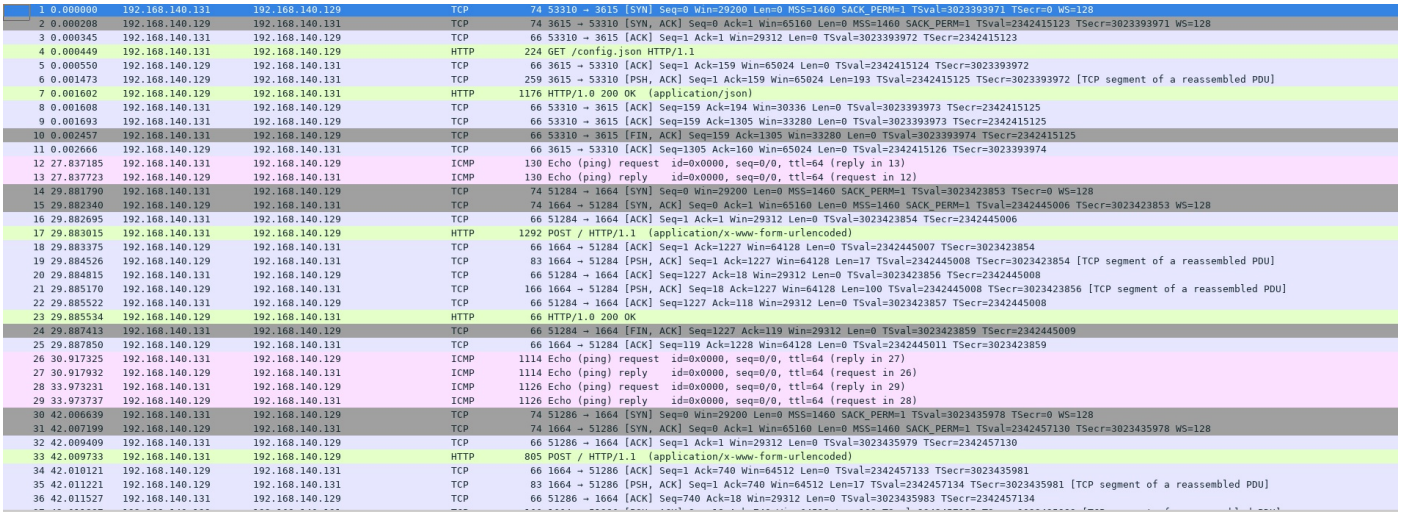
Write Up Caper at ESGI CTF

- Author : Maki
- Given file : caper.pcapng

First step : Discovering the file



1	0.000000	192.168.140.131	192.168.140.129	TCP	74 53310 → 3615 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=3023393971 TSecr=0 WS=128
2	0.000208	192.168.140.129	192.168.140.131	TCP	74 3615 → 53310 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2342415123 TSecr=3023393971 WS=128
3	0.000345	192.168.140.131	192.168.140.129	TCP	66 53310 → 3615 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=3023393972 TSecr=2342415123
4	0.000449	192.168.140.131	192.168.140.129	HTTP	224 GET /config.json HTTP/1.1
5	0.000550	192.168.140.129	192.168.140.131	TCP	66 3615 → 53310 [ACK] Seq=1 Ack=159 Win=65024 Len=0 TSval=2342415124 TSecr=3023393972
6	0.001473	192.168.140.129	192.168.140.131	TCP	259 3615 → 53310 [PSH, ACK] Seq=1 Ack=159 Win=65024 Len=193 TSval=2342415125 TSecr=3023393972 [TCP segment of a reassembled PDU]
7	0.001602	192.168.140.129	192.168.140.131	HTTP	1176 HTTP/1.0 200 OK (application/json)
8	0.001608	192.168.140.131	192.168.140.129	TCP	66 53310 → 3615 [ACK] Seq=159 Ack=194 Win=30336 Len=0 TSval=3023393973 TSecr=2342415125
9	0.001693	192.168.140.131	192.168.140.129	TCP	66 53310 → 3615 [ACK] Seq=159 Ack=1305 Win=33280 Len=0 TSval=3023393973 TSecr=2342415125
10	0.002457	192.168.140.131	192.168.140.129	TCP	66 53310 → 3615 [FIN, ACK] Seq=159 Ack=1305 Win=33280 Len=0 TSval=3023393974 TSecr=2342415125
11	0.002666	192.168.140.129	192.168.140.131	TCP	66 3615 → 53310 [ACK] Seq=1305 Ack=160 Win=65024 Len=0 TSval=2342415126 TSecr=3023393974
12	27.837185	192.168.140.131	192.168.140.129	ICMP	130 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 13)
13	27.837723	192.168.140.129	192.168.140.131	ICMP	130 Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 12)
14	29.881790	192.168.140.131	192.168.140.129	TCP	74 51284 → 1664 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=3023423853 TSecr=0 WS=128
15	29.882340	192.168.140.129	192.168.140.131	TCP	74 1664 → 51284 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2342445006 TSecr=3023423853 WS=128
16	29.882695	192.168.140.131	192.168.140.129	TCP	66 51284 → 1664 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=3023423854 TSecr=2342445006
17	29.883015	192.168.140.131	192.168.140.129	HTTP	1292 POST / HTTP/1.1 (application/x-www-form-urlencoded)
18	29.883375	192.168.140.129	192.168.140.131	TCP	66 1664 → 51284 [ACK] Seq=1 Ack=1227 Win=64128 Len=0 TSval=2342445007 TSecr=3023423854
19	29.884526	192.168.140.129	192.168.140.131	TCP	83 1664 → 51284 [PSH, ACK] Seq=1 Ack=1227 Win=64128 Len=17 TSval=2342445008 TSecr=3023423854 [TCP segment of a reassembled PDU]
20	29.884815	192.168.140.131	192.168.140.129	TCP	66 51284 → 1664 [ACK] Seq=1227 Ack=18 Win=29312 Len=0 TSval=3023423856 TSecr=2342445008
21	29.885170	192.168.140.129	192.168.140.131	TCP	166 1664 → 51284 [PSH, ACK] Seq=18 Ack=1227 Win=64128 Len=100 TSval=2342445008 TSecr=3023423856 [TCP segment of a reassembled PDU]
22	29.885522	192.168.140.131	192.168.140.129	TCP	66 51284 → 1664 [ACK] Seq=1227 Ack=118 Win=29312 Len=0 TSval=3023423857 TSecr=2342445008
23	29.885534	192.168.140.129	192.168.140.131	HTTP	66 HTTP/1.0 200 OK
24	29.887413	192.168.140.131	192.168.140.129	TCP	66 51284 → 1664 [FIN, ACK] Seq=1227 Ack=119 Win=29312 Len=0 TSval=3023423859 TSecr=2342445009
25	29.887850	192.168.140.129	192.168.140.131	TCP	66 1664 → 51284 [ACK] Seq=119 Ack=1228 Win=64128 Len=0 TSval=2342445011 TSecr=3023423859
26	30.917325	192.168.140.131	192.168.140.129	ICMP	1114 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 27)
27	30.917932	192.168.140.129	192.168.140.131	ICMP	1114 Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 26)
28	33.973231	192.168.140.131	192.168.140.129	ICMP	1126 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 29)
29	33.973737	192.168.140.129	192.168.140.131	ICMP	1126 Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 28)
30	42.006639	192.168.140.131	192.168.140.129	TCP	74 51286 → 1664 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=3023435978 TSecr=0 WS=128
31	42.007199	192.168.140.129	192.168.140.131	TCP	74 1664 → 51286 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2342457130 TSecr=3023435978 WS=128
32	42.009409	192.168.140.131	192.168.140.129	TCP	66 51286 → 1664 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=3023435979 TSecr=2342457130
33	42.009733	192.168.140.131	192.168.140.129	HTTP	805 POST / HTTP/1.1 (application/x-www-form-urlencoded)
34	42.010121	192.168.140.129	192.168.140.131	TCP	66 1664 → 51286 [ACK] Seq=1 Ack=740 Win=64512 Len=0 TSval=2342457133 TSecr=3023435981
35	42.011221	192.168.140.129	192.168.140.131	TCP	83 1664 → 51286 [PSH, ACK] Seq=1 Ack=740 Win=64512 Len=17 TSval=2342457134 TSecr=3023435981 [TCP segment of a reassembled PDU]
36	42.011527	192.168.140.131	192.168.140.129	TCP	66 51286 → 1664 [ACK] Seq=740 Ack=18 Win=29312 Len=0 TSval=3023435983 TSecr=2342457134

Into that file we can see some HTTP and ICMP data exchanged.

Second step : Data extraction

Request HTTP

File → Export Objects → HTTP With that I get a config.json File

```
{
    "plugins": {
        "http": {
            "target": "192.168.140.129",
            "port": 1664
        },
        "google_docs": {
            "target": "SERVER",
            "port": 8080
        },
        "dns": {
            "key": "evil.securityday.xxx",
            "target": "192.168.140.129",
            "port": 53
        },
        "gmail": {
            "username": "dataexfil@gmail.com",
            "password": "CrazyNicePassword",
            "server": "smtp.gmail.com",
```

```
            "port": 587
        },
        "tcp": {
            "target": "192.168.140.129",
            "port": 6969
        },
        "udp": {
            "target": "192.168.140.129",
            "port": 6969
        },
        "icmp": {
            "target": "192.168.140.129"
        },
        "slack": {
            "api_token": "xoxb-XXXXXXXXXXX",
            "chan_id": "XXXXXXXXXX",
            "bot_id": "<@XXXXXXXXXX>:"
        }
    },
    "AES_KEY": "S3cur1tyD4y",
    "max_time_sleep": 10,
    "min_time_sleep": 1,
    "max_bytes_read": 400,
    "min_bytes_read": 300,
    "compression": 1
}
```

Some informations of that file will be required for the last step.

The command to extract other data from HTTP

```
$ tshark -r caper.pcapng -Y http -Tfields -e urlencoded-form.value > data_http.b64
```

Convert base64 to clear

```
$ cat data_http.b64 |base64 -d
I5LS9MX|!|0|!|512b74ac48546bd6b065110d86c593658b15fee5c7f78f892e50442ef1aba6ec1e395c0
cda4a6d475557212bca6bb2256dc1c0b6c8a0a10b3f45eefcfbb067efcd5b4317a8f8f4ee3a3d1566257a
a6e1c15a5a5dafe9e400718843e65be2b97b47e8a17f501dffc81066f48f738a8b2b3c91c83a9038c44f4
482c784747a5841a66fdf5b174be52f8df37313501a96ed1e5f7af7e0092db73a4eedd07f5522e82b19ca
ee379a21f6eec66c8ef8488f6d85bc3b725c37d6fdb48b712e8d8cc2459eb18d6048f3cb44991016a66dc
da1cf643bd600afd9addad340baee1e364a19a1fb5d03581e0d84b7cd3d2f0f5d97429a6cbdebf9a4c4f2
342e830463066a0a0c149d95b8b478b603efe1c3b8af7f46cfd463c045bf5a906282615d0d87408e8db46
77aeeb058bdb6da1999995a2b9231183464350bee1d6b37d947b5755dd9ee18cd8c20e0a3e65306d95dfe
177290632fb74e2ab420a0de0771d64186c650bd20fb274dfe41f48098732f6f8bI5LS9MX|!|3|!|93077
6bb930de41985b536645e2b1b8bcd0b4107131480767667769f1181e03eadbc90eb65eaea01ba70ffdc54
b9490275b6e8072b3143a264b10ab6e4a4ca50ca76ff35d6e4178a4c3c62a2049992863a88b6771f5c7a8
1a3a8c6d288f3740e8b8cb720ef38af92f73a683ef548f52943d89d89486603b3112d6d14d8fe4a2ff4e0
e41e995976648d573d9a418ec60120aa30ffc3ac6f1d6f2b8138847fb357ce158789b6b912d4def3ffd5b
7905c569f78b6a46d531d3f1c
```

We can see 'I5LS9MX|!|X|!|' scheme where X is a number.

This data contain only 0 and 3, something is missing. Let's have a look at the icmp data.

Request ICMP

As the HTTP I extract some ICMP data.

```
$  tshark -r caper.pcapng -Y icmp.resp_to -Tfields -e data.data | tr -d ':' | xxd -r
-p | sed 's/ejZmO/\nejZmO/g' > data_icmp.b64
```

Once decoded

```
I5LS9MX|!|flag.txt|!|REGISTER|!|3ec59ac658986a43921d824ae06ea494I5LS9MX|!|1|!|d336c41
94b8c3d5a8f96d50e926db9efdafa006037570e2899ddba564efd7ec5cacf0910d02ef8bbe51c40e97485
e70f58737c1747dac16367c8feeb36ab6ad4325c949cfffbc81d6d2274abf75fd99ee7012f962ae71fca8
e37f8df455305b4503824ff06cb1a4f31cd699308fcf03af16e59f4f07040fc6308eb8be9552766080b88
1fe595e40ca1ae05c9a0b0c5cfa371691ea4443643f825ed9676b3f21c73b49cf7b23b1b38269bfac263b
fe4a6493425f0e6b4757a21552f73f7a58dc072f57fe73b6386326125b3fa93be68f185d523a95a3e4815
9a8244bee23ae70b3a4acf0fc917e94835a5e6f1bb0700f94472d924e1d4b89e57b46e64c5d47a00b4602
ebad118ecdea48bb860691be4d8f7d9fccf0afcab660c6305193c84f91b8550ec7e354dafd8080ff01e4d
29df192465a1da564237f583472d30886f0e9391bb6a0ddfbcfadda44b95ad069209a3e90b9bc3650831a
3e63925b0adfe1161fcf4f28c8b26ebf147edfdfbfa73359c510fbb67b6b3c662ceed56cfac75ca8d06dc
12df6e83efbbbfe2I5LS9MX|!|2|!|4030de613bb59dc4329e59e6b9b857312522cf6a12dde919ac26a3b
c96b62f98247ea32762fc0d85e19f9c33afee6925a00e8dc4865bff24b4931f7f9c4ad7039adc78009ba2
ec11fb7c9485e864f3260e4e3c20ef6356e9f5636f6e28a55e09613782d167e10280c7aa83e568de30c98
88c5c779fe6b9362118ba0e78e4b9a6bfee0305d173d7ce3e0670bd1a1c2cf1c2744e33ac1cd0fd97b323
eb59a911023c28234c1af9224722809b1b3d51f2e10b7e90d804a5867394ab5ad4b64f3d170886f8f6078
723a40bb1bd240f920a1b667de3638c1b48c0a2cec4c5674c1ae9825944fd62af5614aa3bdfc8758f6c74
787a94bc29a7a336a78a02b49a31491f85c36ad714931bf5ca163c1e8dbb96fd6d7b87debc5ab5c5a3131
d3e40f1ec659393c9e0ff6afdda92dd834fe1bd9d6e6942436242f070d044694c3fc230338f0ffcf77aef
b98c2c12908789a45be681c7972198d3832ecc73cb563b4422d7461abc83a74785f4255b1801fba76da51
991e5aba6e4a3cc99f90816ff00891ebd2da9693cb565fa66ba287cb6b1c39cI5LS9MX|!|4|!|DONE
```

Now I have all data needed.

Third step : Reconstitution of the File

I used something similar in an other CTF, the DET framework were used.

After read once again the source code, I found the same pattern and remember me that :

PATTERN|!|File_name|!|REGISTER|!|Hash_filePATTERN|!|X|!|DATA_N_TIME|!|Y|!|DONE

Where X is an integer between 0 and 3 in my case and Y is 4. When I know that I can get only data between 0 and 3 and add them to an other file.

```
512b74ac48546bd6b065110d86c593658b15fee5c7f78f892e50442ef1aba6ec1e395c0cda4a6d4755572
12bca6bb2256dc1c0b6c8a0a10b3f45eefcfbb067efcd5b4317a8f8f4ee3a3d1566257aa6e1c15a5a5daf
e9e400718843e65be2b97b47e8a17f501dffc81066f48f738a8b2b3c91c83a9038c44f4482c784747a584
1a66fdf5b174be52f8df37313501a96ed1e5f7af7e0092db73a4eedd07f5522e82b19caee379a21f6eec6
6c8ef8488f6d85bc3b725c37d6fdb48b712e8d8cc2459eb18d6048f3cb44991016a66dcda1cf643bd600a
fd9addad340baee1e364a19a1fb5d03581e0d84b7cd3d2f0f5d97429a6cbdebf9a4c4f2342e830463066a
0a0c149d95b8b478b603efe1c3b8af7f46cfd463c045bf5a906282615d0d87408e8db4677aeeb058bdb6d
a1999995a2b9231183464350bee1d6b37d947b5755dd9ee18cd8c20e0a3e65306d95dfe177290632fb74e
2ab420a0de0771d64186c650bd20fb274dfe41f48098732f6f8bd336c4194b8c3d5a8f96d50e926db9efd
afa006037570e2899ddba564efd7ec5cacf0910d02ef8bbe51c40e97485e70f58737c1747dac16367c8fe
eb36ab6ad4325c949cfffbc81d6d2274abf75fd99ee7012f962ae71fca8e37f8df455305b4503824ff06c
b1a4f31cd699308fcf03af16e59f4f07040fc6308eb8be9552766080b881fe595e40ca1ae05c9a0b0c5cf
```

```
a371691ea4443643f825ed9676b3f21c73b49cf7b23b1b38269bfac263bfe4a6493425f0e6b4757a21552
f73f7a58dc072f57fe73b6386326125b3fa93be68f185d523a95a3e48159a8244bee23ae70b3a4acf0fc9
17e94835a5e6f1bb0700f94472d924e1d4b89e57b46e64c5d47a00b4602ebad118ecdea48bb860691be4d
8f7d9fccf0afcab660c6305193c84f91b8550ec7e354dafd8080ff01e4d29df192465a1da564237f58347
2d30886f0e9391bb6a0ddfbcfadda44b95ad069209a3e90b9bc3650831a3e63925b0adfe1161fcf4f28c8
b26ebf147edfdfbfa73359c510fbb67b6b3c662ceed56cfac75ca8d06dc12df6e83efbbbfe24030de613b
b59dc4329e59e6b9b857312522cf6a12dde919ac26a3bc96b62f98247ea32762fc0d85e19f9c33afee692
5a00e8dc4865bff24b4931f7f9c4ad7039adc78009ba2ec11fb7c9485e864f3260e4e3c20ef6356e9f563
6f6e28a55e09613782d167e10280c7aa83e568de30c9888c5c779fe6b9362118ba0e78e4b9a6bfee0305d
173d7ce3e0670bd1a1c2cf1c2744e33ac1cd0fd97b323eb59a911023c28234c1af9224722809b1b3d51f2
e10b7e90d804a5867394ab5ad4b64f3d170886f8f6078723a40bb1bd240f920a1b667de3638c1b48c0a2c
ec4c5674c1ae9825944fd62af5614aa3bdfc8758f6c74787a94bc29a7a336a78a02b49a31491f85c36ad7
14931bf5ca163c1e8dbb96fd6d7b87debc5ab5c5a3131d3e40f1ec659393c9e0ff6afdda92dd834fe1bd9
d6e6942436242f070d044694c3fc230338f0ffcf77aefb98c2c12908789a45be681c7972198d3832ecc73
cb563b4422d7461abc83a74785f4255b1801fba76da51991e5aba6e4a3cc99f90816ff00891ebd2da9693
cb565fa66ba287cb6b1c39c930776bb930de41985b536645e2b1b8bcd0b4107131480767667769f1181e0
3eadbc90eb65eaea01ba70ffdc54b9490275b6e8072b3143a264b10ab6e4a4ca50ca76ff35d6e4178a4c3
c62a2049992863a88b6771f5c7a81a3a8c6d288f3740e8b8cb720ef38af92f73a683ef548f52943d89d89
486603b3112d6d14d8fe4a2ff4e0e41e995976648d573d9a418ec60120aa30ffc3ac6f1d6f2b8138847fb
357ce158789b6b912d4def3ffd5b7905c569f78b6a46d531d3f1c
```

Four step : Uncipher data

I converted the hex data into binary.

```
$ xxd -r -p data.hex > data.bin
```

Into the det.py file I found the aes_decrypt function, that will help me.

Do you remember the config.json found at the beginning ?

That file gave me the AES_KEY and if the compression were used or not.

```
{
[...]
"AES_KEY": "S3cur1tyD4y",
[...]
"compression": 1
}
```

I just need to exploit all those informations.

```
from zlib import compress, decompress
from Crypto.Cipher import AES
import hashlib

def aes_decrypt(message, key):
    try:
        # Retrieve CBC IV
        iv = message[:AES.block_size]
        message = message[AES.block_size:]
```

```
        # Derive AES key from passphrase
        aes = AES.new(hashlib.sha256(key).digest(), AES.MODE_CBC, iv)
        message = aes.decrypt(message)

        # Remove PKCS5 padding
        unpad = lambda s: s[:-ord(s[len(s) - 1:])]

        return unpad(message)
    except:
        return None


cipher = open("data.bin","rb")
uncipher = aes_decrypt(cipher.read(),"S3cur1tyD4y")
uncompress = decompress(uncipher)
print(uncompress)
```

Flag

```
ESGI{DET_1s_A_R3aly_GR3aT_t00L}
```