havoc

April 2, 2023

Disclaimer: I'm writing in we because I is just weird.

1 Start

We are given a wireshark capture and have to look for a channel which is used to exfiltrate data. Apparently, someone is looking for a KeePass implementation and browses to `keepass.xyz` to download it. However, afterwards the person doesn't get what he wants and looks for the downloaded binary on `virustotal.com`. Dissatisfied with the downloaded `keepass`, the user proceeds to look for the *right* domain. In the end, he got what he wanted: `Keepass` and a `Malware`. How do we know?

2 Data dump

We first of start to dump all http payloads that were sent to `10.0.2.15`. This leaves us with the many `status.php`,`index.php` and one `.html` file. The `.html` seems to contain a binary that the user will download. We quickly load the page into a browser to get the binary to save time reversing the javascript. `Binwalk` tells us that the binary is for Windows (Laughing in mac, I'm safe). I loaded the binary into `BinaryNinja` but after some minutes I thought it may be smart to ask VirusTotal what it thinks about it. Microsoft labels the virus as `VirTool:Win64/Havokiz.D!MTB`. So let's .... google for it. We quickly come across the github repository for a fully developed C2C solution.

3 How does the damn thing work

Because I developed with `go` at work and the codebase was with go, I couldn't resist to clone the repo. After understanding how everything works and looking into some videos of *john hammond*, I tried to poke into the codebase. How does a victim aka `Agent` communicate with the controller aka `TeamServer`.

3.1 We are lucky

(After scrolling through the wiki, we noticed how `Agents` are set up.)[https://github.com/HavocFramework/Havoc/blob/main/WIKI.MD#Listeners]

```
Secure      = true
UserAgent   = "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecl
Uris        = [
    "/funny_cat.gif",
```

```
            "/index.php",
            "/test.txt",
            "/helloworld.js"
        ]
```

Aahhh! Remember the bunch of `index.php` files? Maybe take a look at them. Nah, they look like random gibber, so maybe encrypted. Furthermore, we notice that the attacker disabled TLS this time, which gave us the hint to look into the exchanged messages for the flag.

3.2 Init a DEMON

Inside `Teamserver/pkg/handlers/handler.go` we can see how the connection is established.

```
        /* TODO: rework this. */
        if Command == agent.DEMON_INIT {
            Agent = agent.ParseResponse(Header.AgentID, Header.Data)
            if Agent == nil {
                return Response, false
            }

            go Agent.BackgroundUpdateLastCallbackUI(Teamserver)

            Agent.TaskedOnce = false
            Agent.Info.MagicValue = Header.MagicValue
            Agent.Info.Listener = nil /* TODO: pass here the listener instance */

            Teamserver.AgentAdd(Agent)
            Teamserver.AgentSendNotify(Agent)

            Packer = packer.NewPacker(Agent.Encryption.AESKey, Agent.Encryption.AESIv)
            Packer.AddUInt32(uint32(Header.AgentID))
```

```
// TODO: change Command IDs. use something more readable and understandable.
const (
    COMMAND_GET_JOB                 = 1
    DEMON_INIT                      = 99
    COMMAND_CHECKIN                 = 100
    ...
)
```

The `ParseResponse()` functions tells us how the data format looks like

```
[ SIZE         ] 4 bytes
[ Magic Value  ] 4 bytes
[ Agent ID     ] 4 bytes
[ COMMAND ID   ] 4 bytes
[ AES KEY      ] 32 bytes
[ AES IV       ] 16 bytes
AES Encrypted {
    [ Agent ID     ] 4 bytes // <-- this is needed to check if we successfully decrypted the da
```

```
    [ User Name    ] size + bytes
    [ Host Name    ] size + bytes
    [ Domain       ] size + bytes
    [ IP Address   ] 16 bytes?
    [ Process Name ] size + bytes
    [ Process ID   ] 4 bytes
    [ Parent  PID  ] 4 bytes
    [ Process Arch ] 4 bytes
    [ Elevated     ] 4 bytes
    [ OS Info      ] ( 5 * 4 ) bytes
    [ OS Arch      ] 4 bytes
    ..... more
}
```

So we have to find a message that inits the connection to get the keys and then decrypt everything.

3.3 Encryption

Let's go down further to find an usual AES encryption in `CTR` mode. It's obvious that it was AES because of the key naming above.

```
func XCryptBytesAES256(XBytes []byte, AESKey []byte, AESIv []byte) []byte {
    var (
        ReverseXBytes = make([]byte, len(XBytes))
    )

    block, err := aes.NewCipher(AESKey)
    if err != nil {
        logger.Error("Decryption Error: " + err.Error())
        return []byte{}
    }

    stream := cipher.NewCTR(block, AESIv)
    stream.XORKeyStream(ReverseXBytes, XBytes)

    return ReverseXBytes
}
```

4 Finding keys

By using the wireshark filter `ip.addr == 10.0.2.15`, we are looking for rather big http requests. Actually, the first *POST* to `status.php` was everything we need:

```
0000   08 00 27 b1 9d 67 08 00 27 d9 e0 af 08 00 45 00   ..'..g..'.....E.
0010   00 f3 c3 03 40 00 80 06 00 00 0a 00 02 06 0a 00   ....@...........
0020   02 0f c7 e3 00 50 da 4f 9d 76 87 71 5d 0c 50 18   .....P.O.v.q].P.
                         SIZE------- Magic------ Agent
0030   20 14 18 fa 00 00 00 00 00 c7 de ad be ef 5c d9    .............\.
       ----- Command ID  AES KEY + IV
```

```
0040   f7 4c 00 00 00 63 4a ba 74 dc f2 86 fc 2e b4 66   .L...cJ.t......f
0050   5c 80 40 3e 76 9a 1c a8 00 a4 da 9a 5c bc 36 9a   \.@>v.......\.6.
0060   6c 22 8e 92 6c de 70 9c 2c 3a 74 a6 58 0a 72 2c   l"..l.p.,:t.X.r,
0070   ac fc 8c 5e 26 ca 18 2e e0 6f fa c3 a3 48 7c 7d   ...^&....o...H|}
0080   12 b2 3e 6e 43 02 81 ed 16 25 40 ec 27 33 16 6b   ..>nC....%@.'3.k
0090   5d fd c5 96 15 19 aa 73 78 ad ef 20 42 80 63 ac   ].......sx.. B.c.
00a0   28 13 ce fb 43 85 37 23 df b9 3b a2 36 7b 0d 62   (...C.7#..;.6{.b
00b0   b5 80 04 48 5b 85 3d 3d aa 69 02 27 45 9d 54 19   ...H[.==.i.'E.T.
00c0   f9 79 38 6b bd 12 0c 87 51 5c 47 06 9a ac 04 cd   .y8k....Q\G.....
00d0   39 cd d0 4e b8 e8 b9 b7 a5 a0 93 a1 72 e6 72 29   9..N........r.r)
00e0   a0 a0 a3 4f 9c 6a 20 e2 05 5c e7 39 c1 d4 fa 15   ...O.j ..\.9....
00f0   cf f6 2c eb 14 0c 74 35 e9 7c 8f 4a aa 79 d3 2a   ..,...t5.|.J.y.*
0100   0c
```

Is this the right message? Yes it is, because `0x63` is the `DEMON_INIT` command. The rest should be now straightforward to implement. Let's see what it decrypts to.

[ ]:
```
from cryptography.hazmat.primitives.ciphers import Cipher,modes,algorithms
```

[ ]:
```
payload = bytes.fromhex("""
000000c7deadbeef5cd9f74c000000634aba74dcf286fc2eb4665c80403e769a1ca800a4da9a5cbc369a6c228e926c
""")
```

[ ]:
```
header,data = payload[:4*4],payload[4*4:]
```

[ ]:
```
header
```

[ ]:
```
b'\x00\x00\x00\xc7\xde\xad\xbe\xef\\\xd9\xf7L\x00\x00\x00c'
```

[ ]:
```
#        DEMON_INIT                    = 99
assert header[-1] == 99
```

[ ]:
```
key = data[:4*8]
iv = data[4*8:6*8]
key.hex(),iv.hex()
```

[ ]:
```
('4aba74dcf286fc2eb4665c80403e769a1ca800a4da9a5cbc369a6c228e926cde',
 '709c2c3a74a6580a722cacfc8c5e26ca')
```

[ ]:
```
def decrypt(key,iv,ciphertext):
    aes = Cipher(algorithm=algorithms.AES256(key),mode=modes.CTR(iv))
    dec = aes.decryptor()
    return dec.update(ciphertext)
```

[ ]:
```
package_data = data[6*8:]
decrypt(key,iv,package_data)
```

[ ]:
```
b'\\\xd9\xf7L\x00\x00\x00\x0fDESKTOP-
9MG6AFM\x00\x00\x00\x05user\x00\x00\x00\x00\x00\x00\x00\x00\x1010.0.2.6\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00#C:\\Users\\user\\Downloads\\keepass.exe\x00\
x00\x1a\\\x00\x00\x00\x00\x00\x00\x00\x02\x00\x00\x00\x00\x00\x00\x00\n\x00\x00\
x00\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00Jd\x00\x00\x00t\x00\x00\x00\x00'
```

Okay, we got the key and iv. Now we simply need the flag. We use the wireshark filter

`ip.dst_host == 10.0.2.15 && http.request.method == "POST"`

and sort by size. Let's try to do it manually right now instead of automating it.

[ ]:
```
int1 = bytes.
↪fromhex("00000158deadbeef5cd9f74c0000000f44f71722fac3a347383841d72921292fe4a0041372aa0f3364
```

[ ]:
```
decrypt(key,iv,int1[4*4:])
```

[ ]:
```
b'\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00.C\x00:\x00\\\x00U\x00s\x00e\x00r\
x00s\x00\\\x00u\x00s\x00e\x00r\x00\\\x00D\x00o\x00w\x00n\x00l\x00o\x00a\x00d\x00
s\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x1d\x00\x00\x0
0\x01\x00\x00\x07\xe7\x00\x00\x00.\x00\x00\x00\x08\x00\x00\x00\x13\x00\x00\x00\x
02.\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x1d\x00\x00\
x00\x01\x00\x00\x07\xe7\x00\x00\x00.\x00\x00\x00\x08\x00\x00\x00\x13\x00\x00\x00
\x04.\x00.\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x1a\x00\x00\x00\x1d\x
00\x00\x00\x01\x00\x00\x07\xe7\x00\x00\x00\x18\x00\x00\x003\x00\x00\x00\x12\x00\
x00\x00\x16d\x00e\x00s\x00k\x00t\x00o\x00p\x00.\x00i\x00n\x00i\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00"\x00\x00\x00\x1d\x00\x00\x00\x01\x00\x00\x07\xe7\
x00\x00\x00$\x00\x00\x00\x06\x00\x00\x00\x13\x00\x00\x00\x18f\x00l\x00a\x00g\x00
.\x00t\x00x\x00t\x00.\x00t\x00x\x00t\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01
H\x00\x00\x00\x00\x1d\x00\x00\x00\x01\x00\x00\x07\xe7\x00\x00\x00\x11\x00\x00\x0
0\t\x00\x00\x00\x13\x00\x00\x00\x16k\x00e\x00e\x00p\x00a\x00s\x00s\x00.\x00e\x00
x\x00e\x00'
```

[ ]:
```
int2 = bytes.
↪fromhex("00000052deadbeef5cd9f74c0000000f44f71729fac3a35f5e382df90b21742f96a0251379aa1e3338
```

[ ]:
```
decrypt(key,iv,int2[4*4:])
```

[ ]:
```
b'\x00\x00\x00\n\x00\x00\x00\x18f\x00l\x00a\x00g\x00.\x00t\x00x\x00t\x00.\x00t\x
00x\x00t\x00\x00\x00\x00"CSCG{1n53cur3_Tr4n5p0rt_3ncrpt10n}'
```

easy

[ ]: