

# Reversing Teerac

Jason Reaves

February 11, 2016

## 1 Introduction

2016 is filled with what seems like a new Ransomware every day, whether the influx is due to the recent sale of the CryptoWall source code or the actors involved in Dyre have since moved on to something profitable after the reported takedown, it would appear that for the time being pushing Ransomware is the new hip thing in the malware world. Most of the big names in Ransomware have had plenty of papers and research done but lots of the newer variants while possibly being based on either leaked or sold code will more often than not make changes in order to make themselves unique. Teerac which is a variant of TorrentLocker with a subdomain generation feature to the hardcoded domain is no exception to this as the malware matches multiple reports on TorrentLocker with the exception of an added subdomain generation.

## 2 Setup

Teerac, which is a TorrentLocker variant, will commonly run from an orphaned explorer.exe process before doing some system setup, first it creates two Cryptographic Service Provider handles, one will be used to create handles from three different AES 256 bit keys in CBC mode all using an IV of 16 NULL bytes and the other will be used with CryptGenRandom. Using one of the AES keys stored in the binary the bot encrypts a copy of itself into a random named folder in `C:\ProgramData\` and sets up persistence in registry with a random named run key. The location of the binary will either be under the `C:\Windows\` directory as a random named executable or will be in `C:\ProgramData\` as a random named executable, the location depends on the privilege level of the malware when it runs. From the samples analyzed it appears the encrypted file 00000000 contains some stats such as how many times the malware has ran, file 01000000 is the encrypted exe and the encrypted file 02000000 contains the location of the non encrypted binary on disk. I've seen sandbox reports and portions of the malware code that indicate this file count can go up to 5 but I've yet to analyze a live sample with a working C2 so most of my research has been purely static analysis.

## 3 Obfuscation

Teerac employs some basic string obfuscation using a xor key and a lookup table of string addresses and sizes(Figure 3). This is easy to get around using a basic ida python script once we've located the lookup table(Figure 1). Running the script shows us some interesting strings(Figure 12). Some other obfuscation techniques used will be a routine that looks up a corresponding function by hash(Figure 2) and then calls it using a jump table and the occasional dead code.

```
#ida python script to decode the strings in the following sample
#teerac_unpacked.exe
#sha256: a92f8b3647fe466ac32d7eee35580888418a47009037e99f9d6374d345f3ceff
```

```
xorkeyaddr = 0x413b20
stringTableAddr = 0x413d48

def decode_string(addr, slen, xor_key):
    out = ""
    for i in range(0, slen):
        out += chr((Byte(xor_key) ^ Byte(addr)) & 0xFF)
        addr += 1
        xor_key += 1
    return out

ptr = stringTableAddr
stringLoc = Dword(ptr)
ptr += 4
slen = Dword(ptr)
ptr += 4
while slen < 255:
    print(decode_string(stringLoc, slen, xorkeyaddr))
    stringLoc = Dword(ptr)
    ptr += 4
    slen = Dword(ptr)
    ptr += 4
```

Figure 1: Teerac String Decoding

```
push    0
push    0
push    0
push    ecx
push    eax
push    edx
push    6
push    0B71ED0D1h
push    3
call    FindByHash_and_Call_401C80
add     esp, 24h
```

Figure 2: Find By Hash

## 4 Interesting Routines

### 4.1 Pseudo Random 1

Teerac uses two variations of the same routine for generating random strings, while the routine isn't very complex one of the variations that is only used once

```

off_413D48      dd offset sub_413D48      ; DATA XREF: sub_4
; int dword_413D4C[]
dword_413D4C    dd 0Ch                    ; DATA XREF: sub_4
; sub_408620+0To
dd offset unk_413B64
db 0Bh
db 0
db 0
db 0
dd offset unk_413B70
db 9
db 0
db 0
db 0
dd offset aCGovs4KKSBe ; "C+0000+4'_K']~{
db 0Ch
db 0
db 0
db 0
dd offset aCGovs4KKSBe+0Ch ; "J~{s ++be+"
db 0Ah
db 0
db 0
db 0
dd offset unk_413B94
db 0Ch
db 0
db 0
db 0

```

Figure 3: Encoded String Lookup Table

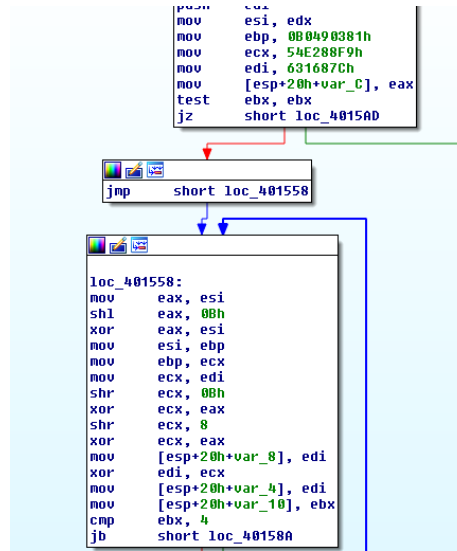


Figure 4: Teerac Rand Bytes

caught my attention because it led me to a similar routine that was used by HesperBot. The routine from Teerac(Figure 4) matches with a routine that was reversed and published by Vinnu on garage4hackers[3]. This routine is used in a similar fashion as it was in HesperBot where it generates a long string of bytes which then feed into another routine, like a stream of salt values, striking a resemblance to the same report, this routine uses the byte stream to generate a string of characters in a seemingly random fashion. One major difference I see in Teerac is a separation of consonants and the vowels where the it will pull a



```

push    eax
mov     esp, 0fh
lea     edi, [esp+80h+var_6C]
call    StringDecrypt_400680 ; Software\Microsoft\Internet Explorer\PhishingFilter
mov     eax, 10h
lea     edi, [esp+80h+var_90]
call    StringDecrypt_400680 ; Enabled00
push    a
lea     eax, [esp+80h+var_84]
push    a
push    a
mov     ecx, edi
push    ecx
push    00000000h
push    0
lea     edx, [esp+80h+var_6C]
mov     [esp+80h+var_84], 0
call    RegisterApplet_4002F0
mov     eax, 10h
call    StringDecrypt_400680 ; Enabled00
push    a
lea     edx, [esp+80h+var_84]
push    edx
push    a
mov     eax, edi
push    eax
push    00000000h
push    0
lea     edx, [esp+80h+var_6C]
mov     [esp+80h+var_84], 0
call    RegisterApplet_4002F0
mov     ecx, [esp+80h+var_4]
add     esp, 30h

```

Figure 7: Teerac Turning Off SmartScreen



Figure 8: Teerac MAPI via COM

```

arg_0= duword ptr 4
push    ebp
mov     ebp, [esp+4+arg_0]
push    edi
push    offset aOutlook ; "Outlook"
push    00000002h ; hKey
mov     ecx, offset aSoftwareHicr_0 ; "Software\Microsoft\Internet Account H".
call    sub_402EF0
mov     edi, eax
add     esp, 0
test    edi, edi
jz      short loc_403037

```

Figure 9: Teerac Outlook Harvest

to store its configuration data and plugin modules according to a report by welivesecurity.com[4].

The other two AES keys used came hardcoded in the binary, one was used for encrypting the files the malware used in the random folder in **ProgramData** and the other key was used for C2 communication. As previously mentioned all keys are used in CBC Mode with an IV of 16 NULL byte.

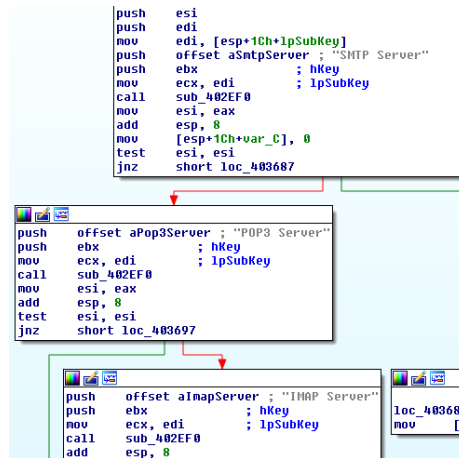


Figure 10: Teerac Email Server Info

02590859	b4 9b 59 f5 34 00 0c 08 02 00 00 10 66 00 00 20 00 00 00	5 4 . . . . f . . . .
02590860	21 10 70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	1 0 . . . . 0 . . . .
02590870	32 14 c5 37 0a 16 f2 07 b3 07 ca 53 a6 00 00 00 00 15 aa	2 7 . . . . Q . . . .
02590880	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
02590890	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
025908a0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
025908b0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
025908c0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
025908d0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
025908e0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
025908f0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
02590900	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
02590910	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
02590920	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
02590930	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
02590940	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
02590950	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
02590960	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
02590970	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
02590980	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
02590990	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
025909a0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
025909b0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
025909c0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
025909d0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
025909e0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
025909f0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .
02590a00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	. . . . .

Figure 11: Hardcoded C2 AES Key

## 5 Worker Threads

Right before it begins trying to find a live C2 the bot spins up 5 threads. One of these threads will delete shadow copies using vssadmin.exe. Another will disable some security features in the registry, phishing filter and SmartScreen(Figure 7). Another simply returns immediately. The other two will utilize COM through CoInitialize, one thread enumerates the registry trying to find various outlook(Figure 9) and account settings(Figure 10) while the other uses MAPI(Figure 8) in an attempt to harvest your contacts.

## 6 C2 Communications

C2 subdomains are constructed by using a hardcoded domain and then generating a random subdomain to add to the beginning of it. From what I've seen when you have a live C2 usually any subdomain will resolve to the same IP of the hardcoded domain. First checkin to the C2 the bot will take the botid which is generated based on the computer name and some hex data converted to ascii which is comprised of random generated data that's been encrypted and

a static string 'main-4' which I've also seen 'main-5' so this appears to be a campaign designator. This data will be encrypted using one of the hardcoded AES keys previously mentioned(Figure 11) and then POSTed up to the C2. The information that follows is purely static for reasons I previously mentioned but I can at least confirm a few things about what the bot expects as response from the C2.

1. The data from the C2 is encrypted with the same key the bot used to encrypt the data that is sent to the C2.
2. The decrypted response data will be parsed in the following structured format `< c2DATA > %%1%% < c2DATA > %%2%% < c2DATA >`.
3. The bot performs validation on the data and will not continue execution until it has gotten a proper response.

## 7 Conclusions

Sample SHA256: f5f7cb83a8f229b96a39f2be7a686fdec717f2519ffe5b62bc98ff439b6f583

Other Samples:

```
89edb283b3a3c892cb8ed7fa893aff5f36982fc3f4657c3b0723351212ded3e6
c4928426873726e4eeb341aeea33d07f41cef58193eb1655bfe1ee6a97afd4c8
2c6b46b60b4ddb5e75a45a9ba2e57a60a1d95bd798bac6b3036ecde237ddd74
56cbf1281a50e0082a1db873bec0097b61c6074152d40598f73c094d37674ea6
6ef7c2cd280b17ea104f7c9c75711992176bb2b854424b779e6da7becda8d998
43d0b93f825a60c676eeab175cc11eea07f1b598bee08bf57d99c64f41a9b8c6
580c61c84d588f32b0cb6b4203cf5918a0c63a15b1529d5ea0ba105b59ab4373
7db8759c7260b71866d896c9a381f47b8d7e452aea3d1d8aab41e38085ccfb70
fe17addfb458cf66f4a922f342baf4337ec33e9e1aa3b715ec94e676ca74417b
c9e9f81c9438ea7a062b41bbb1c121f88b6a372c4eb15030c50a3f16b714b62d
3c38e1e5956c2a9f6fe4f33d52d5c1ddbdc2e43abeda25b16f7ae4aa7eaa610f
545f991909341b92702a0aa2aa18c4ccceefad207af2180aeed24f5c1b346037
```

C2 domains observed:

```
megezawone.net
vodleklinia.org
pyjtoxoyr.org
ioytoxpaire.net
kdiertyjoxeg.com
vjivebilan.org
jgiwoxoqlwez.com
rygzatyee.com
asoiжайsojais.net
nemexcikx.net
kheoyostowe.net
lderktdfphje.net
```



```

explorer.exe
svchost.exe
ntdll.dll
kernel32.dll
user32.dll
advapi32.dll
shlwapi.dll
wininet.dll
%ProgramFiles%
%ProgramW6432%
.encrypted
CryptoLocker
.exe
.html
.txt
Software\Microsoft\Internet Explorer\PhishingFilter
EnabledV8
EnabledV9
vssadmin.exe Delete Shadows /All /Quiet
UNKNOWN
Common Files\System\wab32.dll
Thunderbird\Profiles\
abook.mab
history.mab
DisplayName
PrimaryEmail
// <!-- <mdb:mork:z v="1.4"/> -->
@$$${
{@
@$$}
}@
(^%X~
~%x)
(%X=
about:blank

```

Figure 12: Teerac Decoded Strings

```

C2 IPs observed:
31.170.104.60
188.225.34.221
80.78.253.130
91.214.114.122

```

## References

- [1] Hex-Rays Decompiler, <http://www.hex-rays.com/products/decompiler/index.shtml>.
- [2] Python, <https://www.python.org/>
- [3] Hesperbot DGA, <http://garage4hackers.com/entry.php?b=3091>
- [4] Hesperbot Technical analysis part 1/2, <http://www.welivesecurity.com/2013/09/06/hesperbot-technical-analysis-part-12/>
- [5] Emerging Threats, <http://www.emergingthreats.net/>
- [6] Update on the Torrentlocker ransomware, <http://blog.fox-it.com/2014/10/21/update-on-the-torrentlocker-ransomware/>