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.
# A Python Script to Decode Strings in the Following Sample

```python
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)
p = 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 8871ED00h
push 3

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
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 Vinmm 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
vowel every even iteration of the loop and vice-versa (Figure 5). In the samples I looked at this routine was only used to generate the string for the mutexs, folder names and part of the bot id.

4.2 Psuedo Random 2

The second random string generator is used for generating everything else from filenames, subdomain names and exe names. The biggest difference between the two is that instead of using the seed routine detailed earlier it uses CryptGenRandom and GetTickCount to create bytes that will be used to index into the character arrays in order to build the random strings along with an extra check almost making it look like a generation algorithm with a slight bias (Figure 6).

4.3 AES Key Generation

As previously mentioned one of the routines matches the DGA routine that was previously used by HesperBot but as it turns out it’s not the only piece of code that shares a striking resemblance to analysis reports on HesperBot. The first half of the AES key the bot generates is based on various information about the machine, specifically it uses the InstallDate value from HKLM\SOFTWARE\Microsoft\WindowsNT\CurrentVersion as the first dword of the key. The second is a hardcoded value depending on if the version is x86 x64 or IA64. The third is the adler32 hash of the value of MachineGuid from HKLM\SOFTWARE\Microsoft\Cryptography. The fourth is the adler32 hash of the value of DigitalProductId from HKLM\SOFTWARE\Microsoft\WindowsNT\CurrentVersion. This was the same information was used to build an encryption key used by HesperBot.
to store it’s 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.
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: f5f7cb83a8f229b96a39f2be7a686fdec9d717f2519ffe5b62bc98f439b6f583

Other Samples:
89edb283b3a3c892cb8ed7fa893aff5f36982fc3f4657c3b072351212ded3e6
c4928426873726e4ebeb341aeea33d07f41cefe58193eb165bf1e1e6a97af24c8
2c6b4660b4db5e75a45a9ba2e57a60a1d95bd798bac6b3036ecde237dddb74
56cbf1281a50e082a1db873bec0097b61c6074512d4059f75c309437674e6a6
6ef7c2cd280b17ea104f7c9c75711992176bb2b854424b779e6da7becda8d998
43d6b93f825a60c676eab175cc11ea07f1b598bee08bf57d99c64f41a9b8c6
580c61c84d588f32b0cb64b203cfc5918a0b63a15b129d5ea0b105b9ab4373
7db87597260b71866d8969a381f47b7d452a2a3d18aab41e38085cfc5f70
fe17addf458cf66f4a922f342baf4337ec3e9e1aa3b715ec94e676ca4717bc9e9f81c9438ea7a062b41b1bc121f88b6ca372c4eb1503c60a3f16b714b62d
3c3b61e59562ca9f6fe4e33d52d5c1ddbc2e43a6eda2b16f7e4aa7ea8610f
545f991909341b92702o0aa2aa18c4cceeefadb207af2180aeed24f5c1b346037

C2 domains observed:
megezawone.net
vodleklina.org
pyjtoxyor.org
iyoxytaire.net
kdiertyjoxeg.com
vjiivebilan.org
jgiwoxqwez.com
ryzzatyee.com
asoijaisoja.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
//@$$
{@
//@$$}
}@
("%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


