Hunting PrivateLoader: Pay-Per-Install Service

tavares.re/blog/2022/06/06/hunting-privateloader-pay-per-install-service/

Jun 6 2022

PrivateLoader is a downloader, first seen on early 2021. It's part of a pay-per-install malware distribution service available on underground forums and so it's used by multiple threat actors to distribute ransomware, information stealers, banking trojans, downloaders, and other commodity malware on windows machines. The malware payloads are selectively delivered to victims based on certain criteria such as location, financial activity, environment and specific software installed. It's delivered through websites that claim to provide cracked software.

Let's have a look at the malware and try to find a way to detect and hunt it.

Encrypted Stack Strings#

Here's a sample analyzed by Zscaler on April 2022:

```
aa2c0a9e34f9fa4cbf1780d757cc84f32a8bd005142012e91a6888167f80f4d5
```

Let's open it on Ghidra. Going into the entry point, following the code, looking for interesting functions, I quickly spot the function at `0x406360`. It's calling `LoadLibraryA` but the `lpLibFileName` parameter is built dynamically at runtime using the stack. Its seems that we found a string encryption technique. Both the string and the xor key are loaded into the stack. Looking a bit more through the function, its seems that this is the way most of the strings are loaded:

```
LEA       EAX=>local_50,[ESP + 0x10]
MOV       dword ptr [ESP + local_50[0]],0x84038676
MOV       dword ptr [ESP + local_50[4]],0xeb71eb3c
MOV       dword ptr [ESP + local_50[8]],0x36fb7b30
MOV       dword ptr [ESP + local_50[12]],0xab7d1f0c
MOVAPS    XMM1,xmmword ptr [ESP + local_50[0]]
MOV       dword ptr [ESP + local_30[0]],0xea71e31d
MOV       dword ptr [ESP + local_30[4]],0xd9428759
MOV       dword ptr [ESP + local_30[8]],0x5a971fle
MOV       dword ptr [ESP + local_30[12]],0xab7d1f0c
PXOR      XMM1,xmmword ptr [ESP + local_30[0]]
PUSH      EAX
MOVAPS    xmmword ptr [ESP + local_50[0]],XMM1
CALL      ESI=>KERNEL32.DLL::LoadLibraryA
```

After XOR the encrypted string with the key, we get `kernel32.dll`.

Detecting The Malware#

This uncommon string decryption technique can be leveraged to build a Yara rule for detection and hunting purposes. To reduce the number of false positives and increase the rule performance, we can add a plaintext unicode string used on the C2 communication and a few minor conditions. Here's the rule:

After running this rule on VirusTotal retro hunting, I got over 1.5k samples on a 1 year timeframe. By manually analyzing some of the matches, I couldn't find any false positives. As a first attempt of hunting and detecting PrivateLoader, this rule seems to yield good results.

Decrypting The Strings#

Now, to faster analyze the malware and better understand its behavior, we should build a string decryptor to help us on our reversing efforts and better document the code. With the help of Capstone disassembly framework, and some trial and error, here's the script:

```
import pefile
from capstone import *

def search(instructions, offset):
  dwords = []
  for inst in instructions:
    if inst[2] == 'mov':
      try:
        dword = int(inst[3].split(' ')[-1], 16).to_bytes(4, 'little')
        dwords.append(dword)
      except:
        pass # not the mov we want
      if inst[3].split(', ')[0].split(' ')[-1] == offset:
        return b''.join(dwords[::-1][:4]) # 16 bytes str chunk

# disassemble .txt section
pe = pefile.PE('aa2c0a9e34f9fa4cbf1780d757cc84f32a8bd005142012e91a6888167f80f4d5')
md = Cs(CS_ARCH_X86, CS_MODE_32)
instructions = []
for (address, size, mnemonic, op_str) in md.disasm_lite(pe.sections[0].get_data(),
0):
  instructions.append((address, size, mnemonic, op_str))

# search, build and decrypt strings
strings = []
addr = None
string = ''
for i, inst in enumerate(instructions):
  if inst[2] == 'pxor':
    try: # possible string decryption found
      key_offset = inst[3].split(' ')[-1]
      key = search(instructions[:i][::-1], key_offset)
      insts = instructions[:i][::-1] # from pxor up
      for j, inst in enumerate(insts):
        if inst[2] == 'movaps':
          # encrypted string being moved to xmm1
          str_offset = inst[3].split(' ')[-1]
          encrypted_str = search(insts[j:], str_offset)
          # str chunk decryption
          string += bytearray(key[i] ^ encrypted_str[i] for i in
range(len(key))).decode()
          break # next chuck

      if not addr:
        addr = hex(inst[0])
      if '\x00' in string:
        strings.append((addr, string.replace('\x00', '')))
        string = ''
        addr = None
      except:
        pass # not the pxor we want
```

After running it against the sample we are analyzing, we get the following strings:

```
0x3ee GetCurrentProcess
0x469 CreateThread
0x4ba CreateFileA
0x506 Sleep
0x572 SetPriorityClass
0x5ec Shell32.dll
0x657 SHGetFolderPathA
0x83b null
0x1078 rb
0x157c http://212.193.30.45/proxies.txt
0x1795 :1080
0x1839 \n
0x1f2d :1080
0x1fd1 :
0x26ce .
0x28ac .
0x2972 .
0x2a34 .
0x32ad http://45.144.225.57/server.txt
0x33c0 HOST:
0x346e :
0x3760 pastebin.com/raw/A7dSG1teëä
0x38a3 HOST:
0x3965 HOST:
0x3b93 http://wfsdragon.ru/api/setStats.php
0x3dcd HOST:
0x3f84 :
0x40ae 2.56.59.42
0x4350 /base/api/statistics.php
0x4439 URL:
0x44b6 :
0x4a5e https://
0x4ad8 .tmp
0x4bf6 \
0x53e9 kernel32.dll
0x544a WINHTTP.dll
0x54a5 wininet.dll
0x65a8 WinHttpConnect
0x6682 WinHttpOpenRequest
0x671a WinHttpQueryDataAvailable
0x67b2 WinHttpSendRequest
0x684a WinHttpReceiveResponse
0x68e2 WinHttpQueryHeaders
0x6956 WinHttpOpen
0x69b5 WinHttpReadData
0x6a20 WinHttpCloseHandle
0x6b09 Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/93.0.4577.63 Safari/537.36
0x7402 http://
0x74ab /
0x7582 ?
0x851a HEAD
0x8fa8 Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/93.0.4577.63 Safari/537.36
0x91f0 wininet.dll
```

```
0x925b InternetSetOptionA
0x92ef HttpOpenRequestA
0x938d InternetConnectA
0x9421 InternetOpenUrlA
0x949e InternetOpenA
0x94f2 HttpQueryInfoA
0x9567 InternetQueryOptionA
0x95fb HttpSendRequestA
0x9694 InternetReadFile
0x9737 InternetCloseHandle
0x97ad Kernel32.dll
0x9801 HeapAlloc
0x9852 HeapFree
0x98a3 GetProcessHeap
0x98f3 CharNextA
0x9938 User32.dll
0x9994 GetLastError
0x99e5 CreateFileA
0x9a36 WriteFile
0x9a87 CloseHandle
```

We can now go back to Ghidra and continue our analysis, now with more context of what might be the malware's behavior.

Network IOCs#

As a bonus, we get some network IOCs that can be used for defense and tracking purposes:

```
http://212.193.30.45/proxies.txt
http://45.144.225.57/server.txt
pastebin.com/raw/A7dSG1te
http://wfsdragon.ru/api/setStats.php
2.56.59.42
/base/api/statistics.php
```