There Is More Than One Way to Sleep: Dive Deep Into the Implementations of API Hammering by Various Malware Families

unit42.paloaltonetworks.com/api-hammering-malware-families/

Mark Lim, Riley Porter June 24, 2022

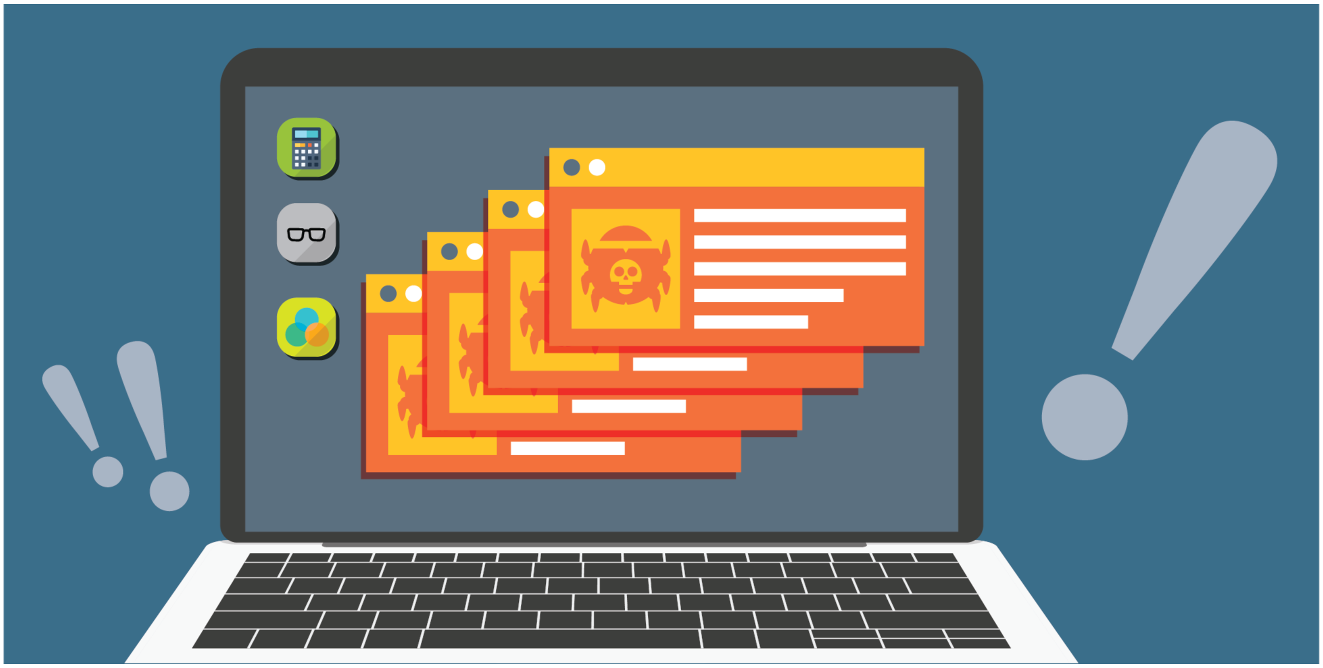
people reacted
- 6
- 5 min. read

By Mark Lim and Riley Porter

June 24, 2022 at 6:00 AM

Category: Malware, Threat Prevention

Tags: API Hammering, BazarLoader, Cortex XDR, malware, Sandbox evasion, threat prevention, WildFire, Zloader



This post is also available in: 日本語 (Japanese)

Executive Summary

Unit 42 has discovered Zloader and BazarLoader samples that had interesting implementations of a sandbox evasion technique. This blog post will go into details of the unique implementations of API Hammering in these types of malware. API Hammering involves the use of a massive number of calls to Windows APIs as a form of extended sleep to evade detection in sandbox environments.

Sandboxing is a popular technique used to detect if a sample is malicious. A sandbox analyzes the behaviors of the binary as it executes inside a controlled environment. Sandboxes have to deal with many challenges while analyzing a large number of binaries with limited computing resources. Malware sometimes abuses these challenges by "sleeping" in the sandbox before carrying out malicious procedures to hide its real intentions.

Palo Alto Networks customers receive protections from malware families using evasion techniques through Cortex XDR or the Next-Generation Firewall with WildFire and Threat Prevention security subscriptions.

Related Unit 42 Topics Malware, evasion

Table of Contents

Common Ways for Malware to Sleep

The most common way for malware to sleep is to simply call the Windows API function Sleep. A sneakier way that we often see is the Ping Sleep technique where the malware constantly sends ICMP network packets to an IP address (ping) in a loop. To send and receive such useless ping messages takes a certain amount of time, thus the malware indirectly sleeps. However, all these methods are easily detected by many sandboxes.

What Is API Hammering?

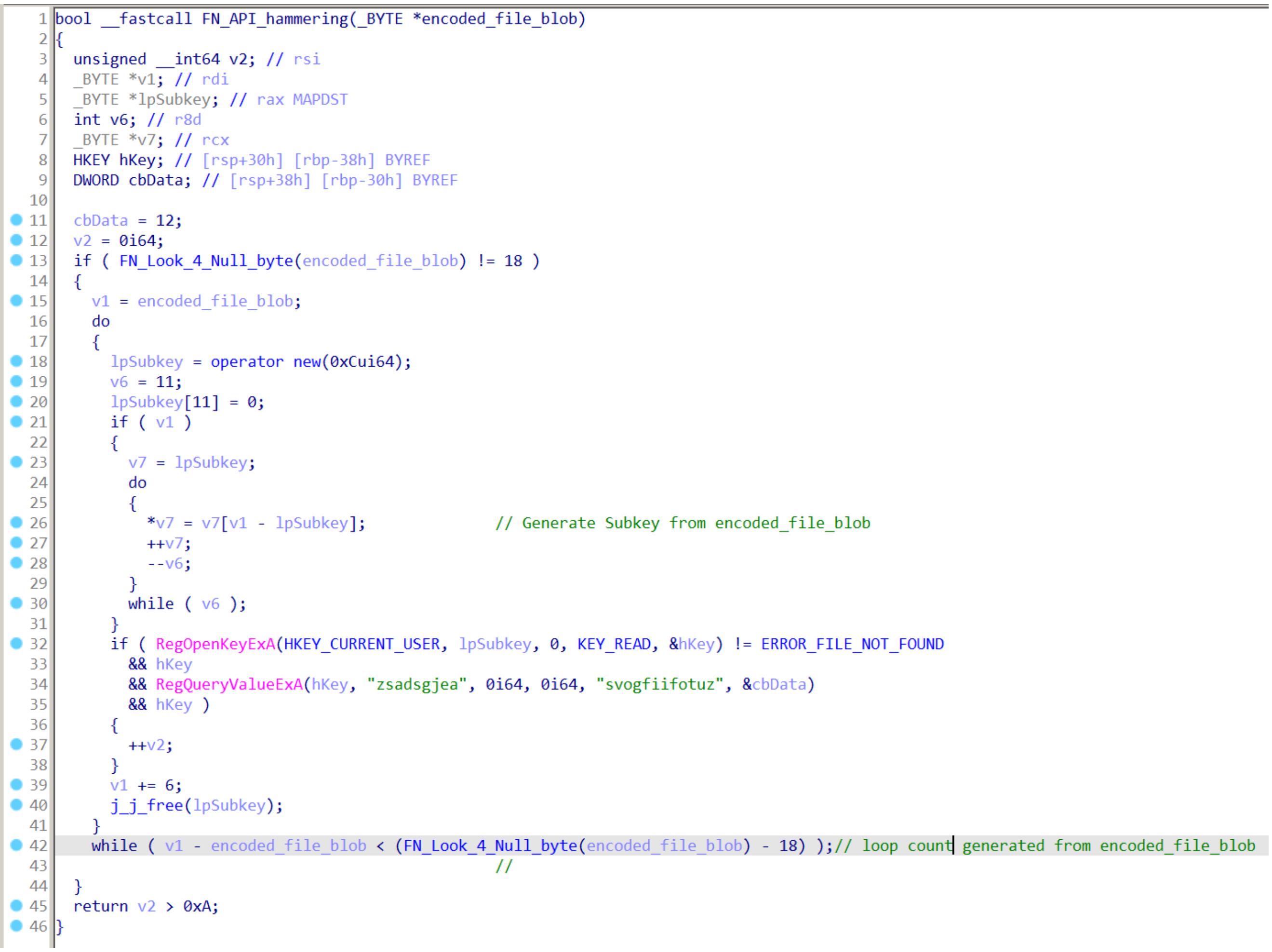
API Hammering has been a known sandbox bypass technique that is sometimes used by malware authors to evade sandboxes. We've recently observed Zloader – a dropper for multiple types of malware – and the backdoor BazarLoader using new and unique implementations of API Hammering to remain stealthy.

API Hammering consists of a large number of garbage Windows API function calls. The execution time of these calls delays the execution of the real malicious routines of the malware. This allows the malware to indirectly sleep during the sandbox analysis process.

API Hammering in BazarLoader

An older variant of BazarLoader made use of a fixed number (1550) of printf function calls to time out malware analysis. While analyzing a newer version of BazarLoader, we found a new and more complex implementation of API Hammering.

The following decompiled procedure shows how this new variant is implemented in the BazarLoader sample we analyzed. It makes use of a huge loop with a random count that repeatedly accesses a list of random registry keys in Windows.

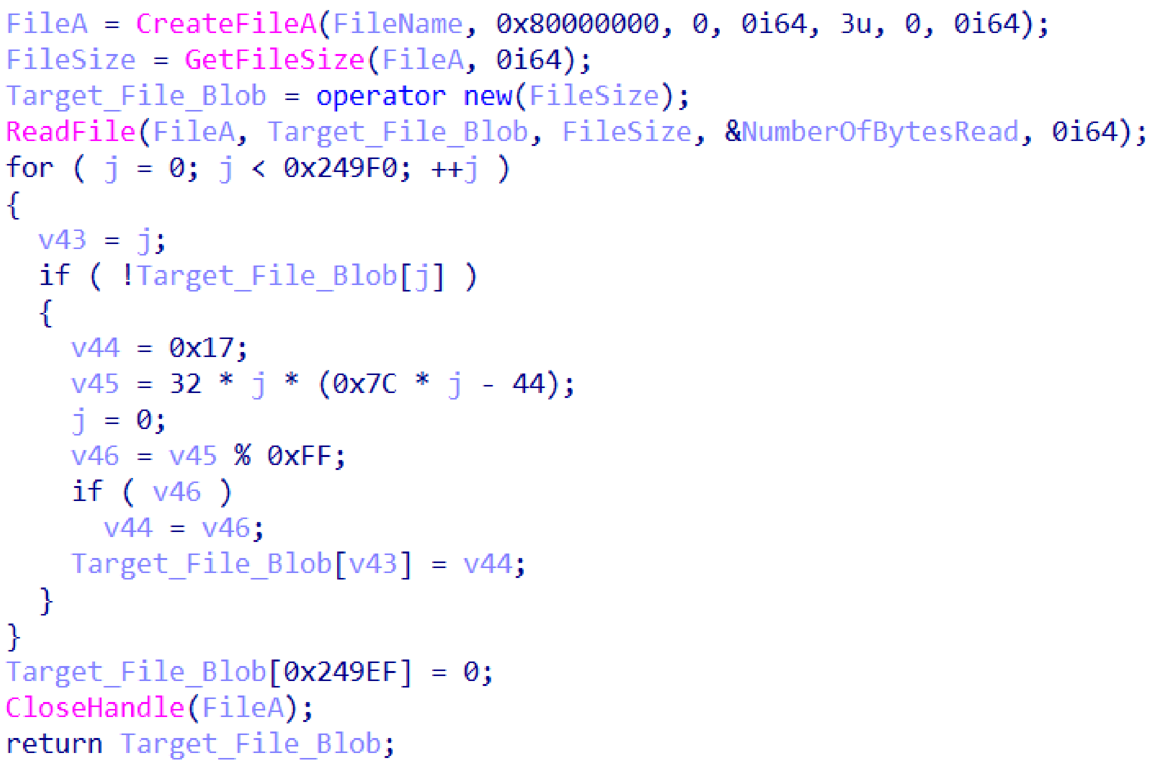


```
bool __fastcall FN_API_hammering(_BYTE *encoded_file_blob)
{
  unsigned __int64 v2; // rsi
  _BYTE *v1; // rdi
  _BYTE *lpSubkey; // rax MAPDST
  int v6; // r8d
  _BYTE *v7; // rcx
  HKEY hKey; // [rsp+30h] [rbp-38h] BYREF
  DWORD cbData; // [rsp+38h] [rbp-30h] BYREF

  cbData = 12;
  v2 = 0i64;
  if ( FN_Look_4_Null_byte(encoded_file_blob) != 18 )
  {
    v1 = encoded_file_blob;
    do
    {
      lpSubkey = operator new(0xCui64);
      v6 = 11;
      lpSubkey[11] = 0;
      if ( v1 )
      {
        v7 = lpSubkey;
        do
        {
          *v7 = v7[v1 - lpSubkey];              // Generate Subkey from encoded_file_blob
          ++v7;
          --v6;
        }
        while ( v6 );
      }
      if ( RegOpenKeyExA(HKEY_CURRENT_USER, lpSubkey, 0, KEY_READ, &hKey) != ERROR_FILE_NOT_FOUND
        && hKey
        && RegQueryValueExA(hKey, "zsadsgjea", 0i64, 0i64, "svogfiifotuz", &cbData)
        && hKey )
      {
        ++v2;
      }
      v1 += 6;
      j_j_free(lpSubkey);
    }
    while ( v1 - encoded_file_blob < (FN_Look_4_Null_byte(encoded_file_blob) - 18) );// loop count generated from encoded_file_blob
                                                //
  }
  return v2 > 0xA;
}
```

Figure 1. API Hammering loop in BazarLoader.

To generate the random loop count and list of registry keys, the sample reads the first file from the System32 directory that matches a defined size. This file is then encoded (see Figure 2) to remove most of its null bytes. The random count is then computed based on the offset of the first null byte in that file. The list of random registry keys are generated from fixed length chunks from the encoded file.

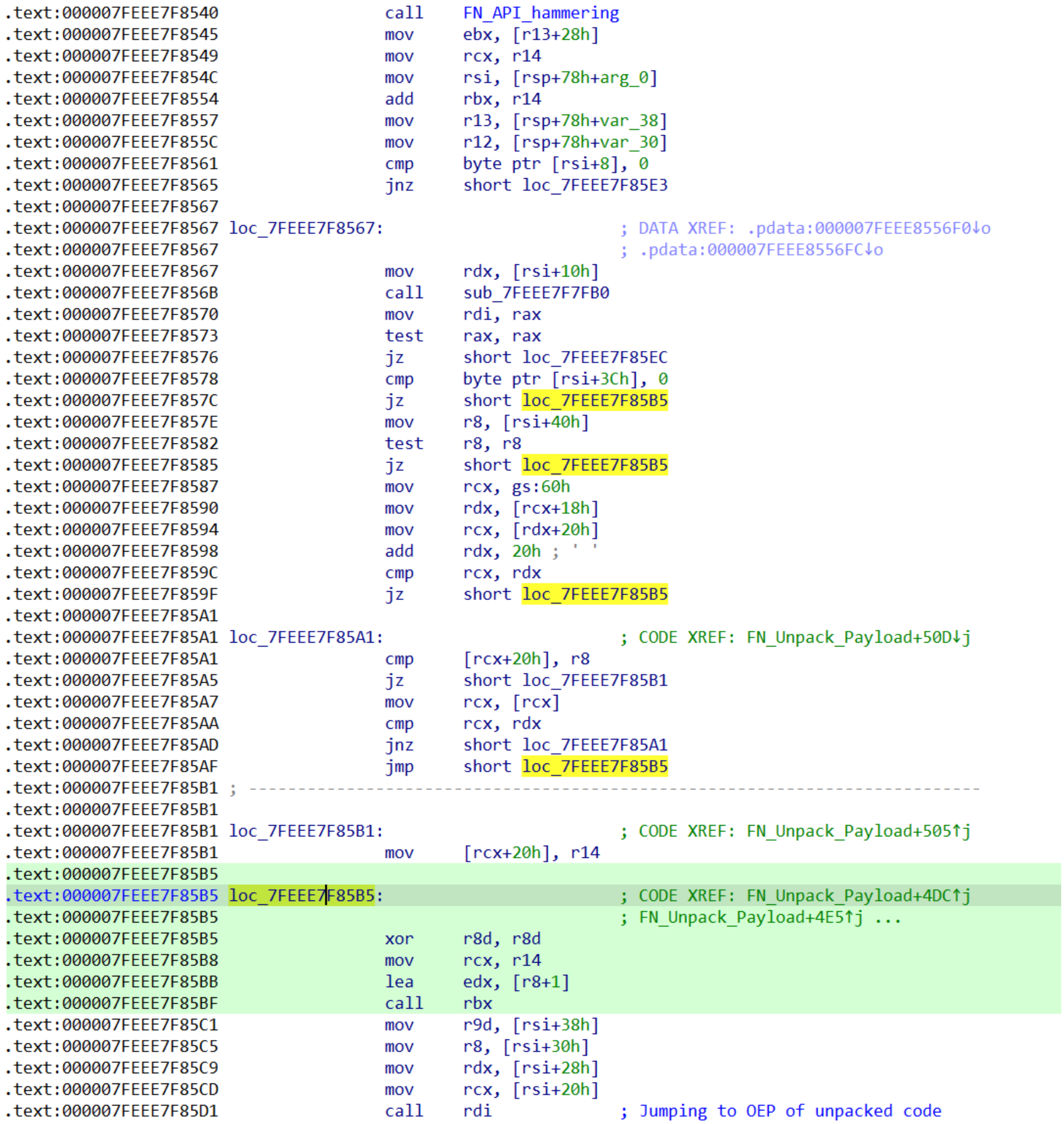


```
FileA = CreateFileA(FileName, 0x80000000, 0, 0i64, 3u, 0, 0i64);
FileSize = GetFileSize(FileA, 0i64);
Target_File_Blob = operator new(FileSize);
ReadFile(FileA, Target_File_Blob, FileSize, &NumberOfBytesRead, 0i64);
for ( j = 0; j < 0x249F0; ++j )
{
  v43 = j;
  if ( !Target_File_Blob[j] )
  {
    v44 = 0x17;
    v45 = 32 * j * (0x7C * j - 44);
    j = 0;
    v46 = v45 % 0xFF;
    if ( v46 )
      v44 = v46;
    Target_File_Blob[v43] = v44;
  }
}
Target_File_Blob[0x249EF] = 0;
CloseHandle(FileA);
return Target_File_Blob;
```

Figure 2. Encoding the selected file in BazarLoader.

With a different Windows version (Windows 7, 8, etc.) and a different set of applied updates, there is also a different set of files in the System32 directory. This results in a varying loop count and list registry keys used by BazarLoader when executed in different machines.

The API Hammering function is located in the packer of the BazarLoader sample (see Figure 3). It delays the payload unpacking process to evade detection of the aforementioned. Without completing the unpacking process, the BazarLoader sample would appear to be just accessing random registry keys, a behavior that can be also seen in many legitimate types of software.

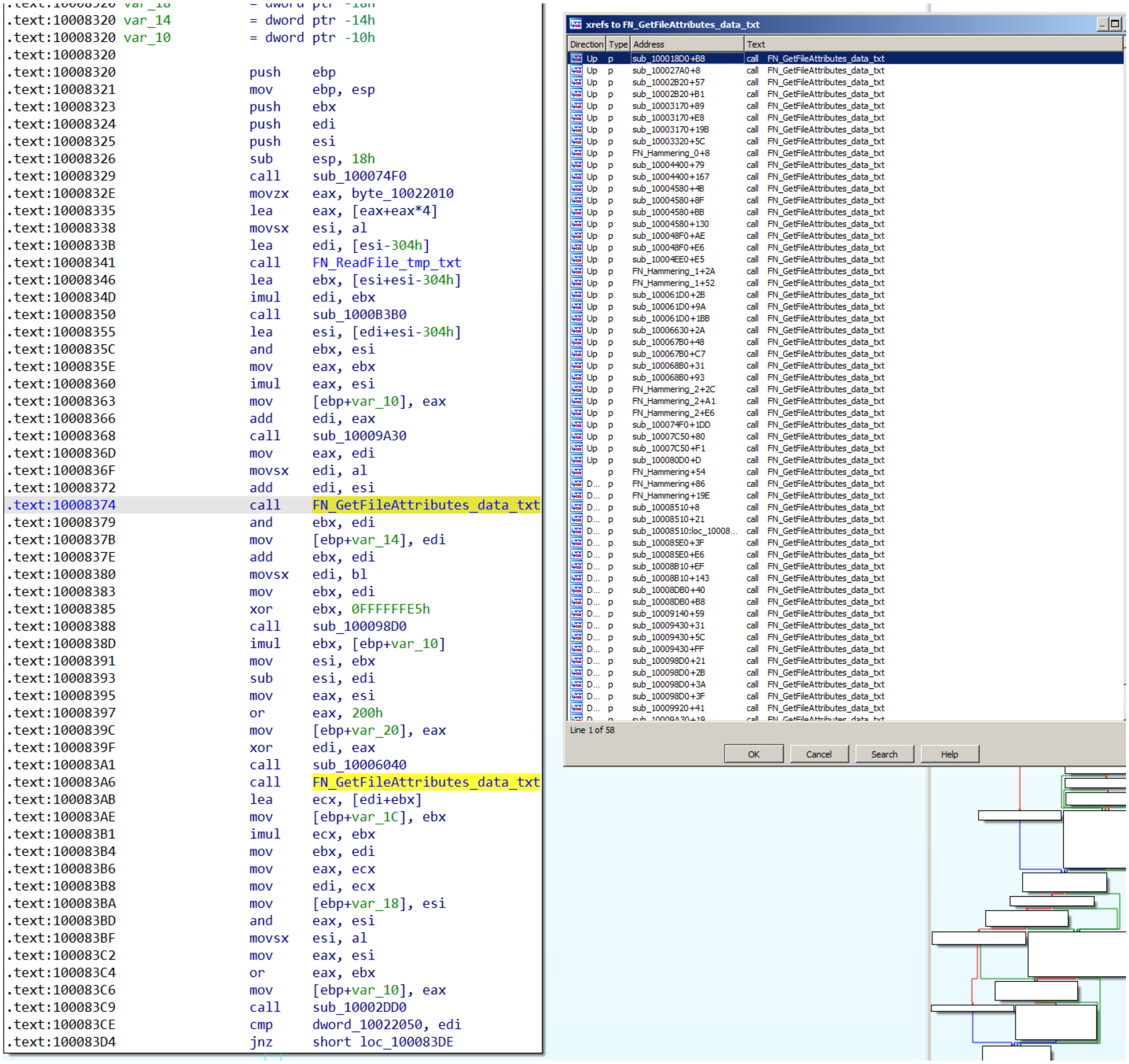


```
.text:000007FEEE7F8540                 call    FN_API_hammering
.text:000007FEEE7F8545                 mov     ebx, [r13+28h]
.text:000007FEEE7F8549                 mov     rcx, r14
.text:000007FEEE7F854C                 mov     rsi, [rsp+78h+arg_0]
.text:000007FEEE7F8554                 add     rbx, r14
.text:000007FEEE7F8557                 mov     r13, [rsp+78h+var_38]
.text:000007FEEE7F855C                 mov     r12, [rsp+78h+var_30]
.text:000007FEEE7F8561                 cmp     byte ptr [rsi+8], 0
.text:000007FEEE7F8565                 jnz     short loc_7FEEE7F85E3
.text:000007FEEE7F8567
.text:000007FEEE7F8567 loc_7FEEE7F8567:                        ; DATA XREF: .pdata:000007FEEE8556F0↓o
.text:000007FEEE7F8567                                         ; .pdata:000007FEEE8556FC↓o
.text:000007FEEE7F8567                 mov     rdx, [rsi+10h]
.text:000007FEEE7F856B                 call    sub_7FEEE7F7FB0
.text:000007FEEE7F8570                 mov     rdi, rax
.text:000007FEEE7F8573                 test    rax, rax
.text:000007FEEE7F8576                 jz      short loc_7FEEE7F85EC
.text:000007FEEE7F8578                 cmp     byte ptr [rsi+3Ch], 0
.text:000007FEEE7F857C                 jz      short loc_7FEEE7F85B5
.text:000007FEEE7F857E                 mov     r8, [rsi+40h]
.text:000007FEEE7F8582                 test    r8, r8
.text:000007FEEE7F8585                 jz      short loc_7FEEE7F85B5
.text:000007FEEE7F8587                 mov     rcx, gs:60h
.text:000007FEEE7F8590                 mov     rdx, [rcx+18h]
.text:000007FEEE7F8594                 mov     rcx, [rdx+20h]
.text:000007FEEE7F8598                 add     rdx, 20h ; ' '
.text:000007FEEE7F859C                 cmp     rcx, rdx
.text:000007FEEE7F859F                 jz      short loc_7FEEE7F85B5
.text:000007FEEE7F85A1
.text:000007FEEE7F85A1 loc_7FEEE7F85A1:                        ; CODE XREF: FN_Unpack_Payload+50D↓j
.text:000007FEEE7F85A1                 cmp     [rcx+20h], r8
.text:000007FEEE7F85A5                 jz      short loc_7FEEE7F85B1
.text:000007FEEE7F85A7                 mov     rcx, [rcx]
.text:000007FEEE7F85AA                 cmp     rcx, rdx
.text:000007FEEE7F85AD                 jnz     short loc_7FEEE7F85A1
.text:000007FEEE7F85AF                 jmp     short loc_7FEEE7F85B5
.text:000007FEEE7F85B1 ; ---------------------------------------------------------------------------
.text:000007FEEE7F85B1
.text:000007FEEE7F85B1 loc_7FEEE7F85B1:                        ; CODE XREF: FN_Unpack_Payload+505↑j
.text:000007FEEE7F85B1                 mov     [rcx+20h], r14
.text:000007FEEE7F85B5
.text:000007FEEE7F85B5 loc_7FEEE7F85B5:                        ; CODE XREF: FN_Unpack_Payload+4DC↑j
.text:000007FEEE7F85B5                                         ; FN_Unpack_Payload+4E5↑j ...
.text:000007FEEE7F85B5                 xor     r8d, r8d
.text:000007FEEE7F85B8                 mov     rcx, r14
.text:000007FEEE7F85BB                 lea     edx, [r8+1]
.text:000007FEEE7F85BF                 call    rbx
.text:000007FEEE7F85C1                 mov     r9d, [rsi+38h]
.text:000007FEEE7F85C5                 mov     r8, [rsi+30h]
.text:000007FEEE7F85C9                 mov     rdx, [rsi+28h]
.text:000007FEEE7F85CD                 mov     rcx, [rsi+20h]
.text:000007FEEE7F85D1                 call    rdi             ; Jumping to OEP of unpacked code
```

Figure 3. API Hammering delaying unpacking process in BazarLoader.

API Hammering in Zloader

While the BazarLoader sample relied on a loop to carry out API Hammering, Zloader uses a different approach. It does not require a huge loop, but instead consists of 4 large functions which contain nested calls to multiple other smaller functions (see Figure 4).



```
.text:10008320 var_18          = dword ptr -18h
.text:10008320 var_14          = dword ptr -14h
.text:10008320 var_10          = dword ptr -10h
.text:10008320
.text:10008320                 push    ebp
.text:10008321                 mov     ebp, esp
.text:10008323                 push    ebx
.text:10008324                 push    edi
.text:10008325                 push    esi
.text:10008326                 sub     esp, 18h
.text:10008329                 call    sub_100074F0
.text:1000832E                 movzx   eax, byte_10022010
.text:10008335                 lea     eax, [eax+eax*4]
.text:10008338                 movsx   esi, al
.text:1000833B                 lea     edi, [esi-304h]
.text:10008341                 call    FN_ReadFile_tmp_txt
.text:10008346                 lea     ebx, [esi+esi-304h]
.text:1000834D                 imul    edi, ebx
.text:10008350                 call    sub_1000B3B0
.text:10008355                 lea     esi, [edi+esi-304h]
.text:1000835C                 and     ebx, esi
.text:1000835E                 mov     eax, ebx
.text:10008360                 imul    eax, esi
.text:10008363                 mov     [ebp+var_10], eax
.text:10008366                 add     edi, eax
.text:10008368                 call    sub_10009A30
.text:1000836D                 mov     eax, edi
.text:1000836F                 movsx   edi, al
.text:10008372                 add     edi, esi
.text:10008374                 call    FN_GetFileAttributes_data_txt
.text:10008379                 and     ebx, edi
.text:1000837B                 mov     [ebp+var_14], edi
.text:1000837E                 add     ebx, edi
.text:10008380                 movsx   edi, bl
.text:10008383                 mov     ebx, edi
.text:10008385                 xor     ebx, 0FFFFFFE5h
.text:10008388                 call    sub_100098D0
.text:1000838D                 imul    ebx, [ebp+var_10]
.text:10008391                 mov     esi, ebx
.text:10008393                 sub     esi, edi
.text:10008395                 mov     eax, esi
.text:10008397                 or      eax, 200h
.text:1000839C                 mov     [ebp+var_20], eax
.text:1000839F                 xor     edi, eax
.text:100083A1                 call    sub_10006040
.text:100083A6                 call    FN_GetFileAttributes_data_txt
.text:100083AB                 lea     ecx, [edi+ebx]
.text:100083AE                 mov     [ebp+var_1C], ebx
.text:100083B1                 imul    ecx, ebx
.text:100083B4                 mov     ebx, edi
.text:100083B6                 mov     eax, ecx
.text:100083B8                 mov     edi, ecx
.text:100083BA                 mov     [ebp+var_18], esi
.text:100083BD                 and     eax, esi
.text:100083BF                 movsx   esi, al
.text:100083C2                 mov     eax, esi
.text:100083C4                 or      eax, ebx
.text:100083C6                 mov     [ebp+var_10], eax
.text:100083C9                 call    sub_10002DD0
.text:100083CE                 cmp     dword_10022050, edi
.text:100083D4                 jnz     short loc_100083DE
```

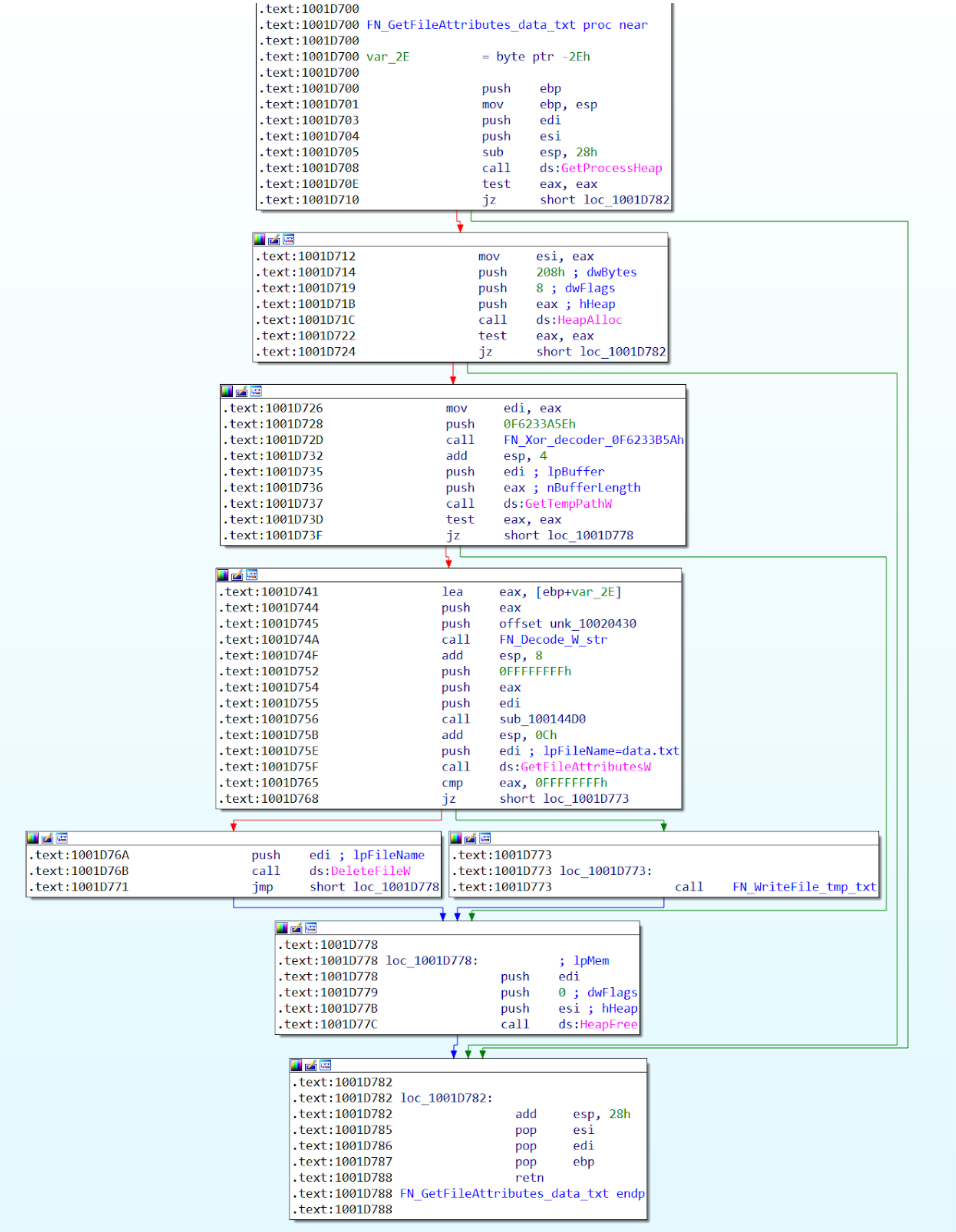
xrefs to FN_GetFileAttributes_data_txt

Direction	Type	Address	Text
Up	p	sub_100018D0+B8	call FN_GetFileAttributes_data_txt
Up	p	sub_100027A0+8	call FN_GetFileAttributes_data_txt
Up	p	sub_10002B20+57	call FN_GetFileAttributes_data_txt
Up	p	sub_10002B20+B1	call FN_GetFileAttributes_data_txt
Up	p	sub_10003170+89	call FN_GetFileAttributes_data_txt
Up	p	sub_10003170+E8	call FN_GetFileAttributes_data_txt
Up	p	sub_10003170+19B	call FN_GetFileAttributes_data_txt
Up	p	sub_10003320+5C	call FN_GetFileAttributes_data_txt
Up	p	FN_Hammering_0+8	call FN_GetFileAttributes_data_txt
Up	p	sub_10004400+79	call FN_GetFileAttributes_data_txt
Up	p	sub_10004400+167	call FN_GetFileAttributes_data_txt
Up	p	sub_10004580+4B	call FN_GetFileAttributes_data_txt
Up	p	sub_10004580+8F	call FN_GetFileAttributes_data_txt
Up	p	sub_10004580+BB	call FN_GetFileAttributes_data_txt
Up	p	sub_10004580+130	call FN_GetFileAttributes_data_txt
Up	p	sub_100048F0+AE	call FN_GetFileAttributes_data_txt
Up	p	sub_100048F0+E6	call FN_GetFileAttributes_data_txt
Up	p	sub_10004EE0+E5	call FN_GetFileAttributes_data_txt
Up	p	FN_Hammering_1+2A	call FN_GetFileAttributes_data_txt
Up	p	FN_Hammering_1+52	call FN_GetFileAttributes_data_txt
Up	p	sub_100061D0+2B	call FN_GetFileAttributes_data_txt
Up	p	sub_100061D0+9A	call FN_GetFileAttributes_data_txt
Up	p	sub_100061D0+1BB	call FN_GetFileAttributes_data_txt
Up	p	sub_10006630+2A	call FN_GetFileAttributes_data_txt
Up	p	sub_100067B0+48	call FN_GetFileAttributes_data_txt
Up	p	sub_100067B0+C7	call FN_GetFileAttributes_data_txt
Up	p	sub_100068B0+31	call FN_GetFileAttributes_data_txt
Up	p	sub_100068B0+93	call FN_GetFileAttributes_data_txt
Up	p	FN_Hammering_2+2C	call FN_GetFileAttributes_data_txt
Up	p	FN_Hammering_2+A1	call FN_GetFileAttributes_data_txt
Up	p	FN_Hammering_2+E6	call FN_GetFileAttributes_data_txt
Up	p	sub_100074F0+1DD	call FN_GetFileAttributes_data_txt
Up	p	sub_10007C50+80	call FN_GetFileAttributes_data_txt
Up	p	sub_10007C50+F1	call FN_GetFileAttributes_data_txt
Up	p	sub_100080D0+D	call FN_GetFileAttributes_data_txt
	p	FN_Hammering+54	call FN_GetFileAttributes_data_txt
D...	p	FN_Hammering+86	call FN_GetFileAttributes_data_txt
D...	p	FN_Hammering+19E	call FN_GetFileAttributes_data_txt
D...	p	sub_10008510+8	call FN_GetFileAttributes_data_txt
D...	p	sub_10008510+21	call FN_GetFileAttributes_data_txt
D...	p	sub_10008510:loc_10008...	call FN_GetFileAttributes_data_txt
D...	p	sub_100085E0+3F	call FN_GetFileAttributes_data_txt
D...	p	sub_100085E0+E6	call FN_GetFileAttributes_data_txt
D...	p	sub_10008B10+EF	call FN_GetFileAttributes_data_txt
D...	p	sub_10008B10+143	call FN_GetFileAttributes_data_txt
D...	p	sub_10008DB0+40	call FN_GetFileAttributes_data_txt
D...	p	sub_10008DB0+B8	call FN_GetFileAttributes_data_txt
D...	p	sub_10009140+59	call FN_GetFileAttributes_data_txt
D...	p	sub_10009430+31	call FN_GetFileAttributes_data_txt
D...	p	sub_10009430+5C	call FN_GetFileAttributes_data_txt
D...	p	sub_10009430+FF	call FN_GetFileAttributes_data_txt
D...	p	sub_100098D0+21	call FN_GetFileAttributes_data_txt
D...	p	sub_100098D0+2B	call FN_GetFileAttributes_data_txt
D...	p	sub_100098D0+3A	call FN_GetFileAttributes_data_txt
D...	p	sub_100098D0+3F	call FN_GetFileAttributes_data_txt
D...	p	sub_10009920+41	call FN_GetFileAttributes_data_txt
D...	p	sub_10009A30+19	call FN_GetFileAttributes_data_txt

Line 1 of 58

Figure 4. One of the large functions responsible for API Hammering in ZLoader.

Inside each of these small procedures are four API function calls related to file I/O. The functions are GetFileAttributesW, ReadFile, CreateFileW and WriteFile (see Figure 5).



```
.text:1001D700
.text:1001D700 FN_GetFileAttributes_data_txt proc near
.text:1001D700
.text:1001D700 var_2E          = byte ptr -2Eh
.text:1001D700
.text:1001D700                 push    ebp
.text:1001D701                 mov     ebp, esp
.text:1001D703                 push    edi
.text:1001D704                 push    esi
.text:1001D705                 sub     esp, 28h
.text:1001D708                 call    ds:GetProcessHeap
.text:1001D70E                 test    eax, eax
.text:1001D710                 jz      short loc_1001D782
```

```
.text:1001D712                 mov     esi, eax
.text:1001D714                 push    208h ; dwBytes
.text:1001D719                 push    8 ; dwFlags
.text:1001D71B                 push    eax ; hHeap
.text:1001D71C                 call    ds:HeapAlloc
.text:1001D722                 test    eax, eax
.text:1001D724                 jz      short loc_1001D782
```

```
.text:1001D726                 mov     edi, eax
.text:1001D728                 push    0F6233A5Eh
.text:1001D72D                 call    FN_Xor_decoder_0F6233B5Ah
.text:1001D732                 add     esp, 4
.text:1001D735                 push    edi ; lpBuffer
.text:1001D736                 push    eax ; nBufferLength
.text:1001D737                 call    ds:GetTempPathW
.text:1001D73D                 test    eax, eax
.text:1001D73F                 jz      short loc_1001D778
```

```
.text:1001D741                 lea     eax, [ebp+var_2E]
.text:1001D744                 push    eax
.text:1001D745                 push    offset unk_10020430
.text:1001D74A                 call    FN_Decode_W_str
.text:1001D74F                 add     esp, 8
.text:1001D752                 push    0FFFFFFFFh
.text:1001D754                 push    eax
.text:1001D755                 push    edi
.text:1001D756                 call    sub_100144D0
.text:1001D75B                 add     esp, 0Ch
.text:1001D75E                 push    edi ; lpFileName=data.txt
.text:1001D75F                 call    ds:GetFileAttributesW
.text:1001D765                 cmp     eax, 0FFFFFFFFh
.text:1001D768                 jz      short loc_1001D773
```

```
.text:1001D76A                 push    edi ; lpFileName
.text:1001D76B                 call    ds:DeleteFileW
.text:1001D771                 jmp     short loc_1001D778
```

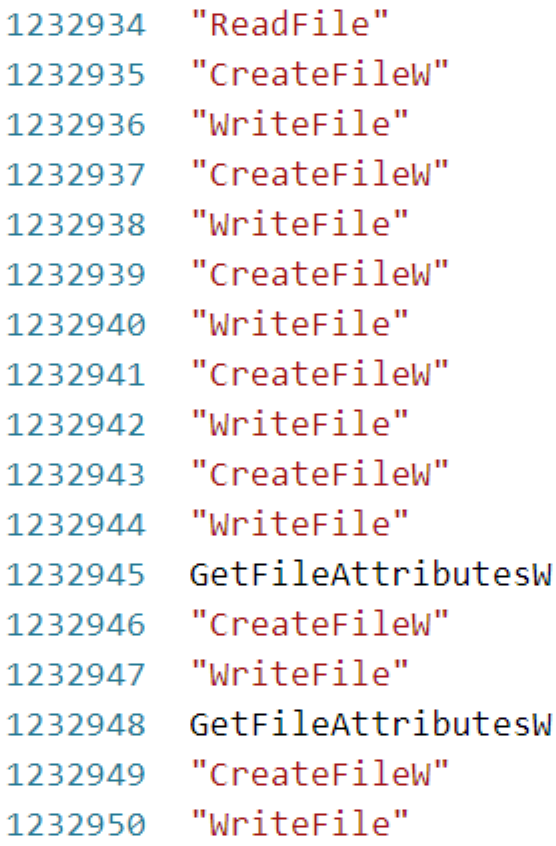
```
.text:1001D773
.text:1001D773 loc_1001D773:
.text:1001D773                 call    FN_WriteFile_tmp_txt
```

```
.text:1001D778
.text:1001D778 loc_1001D778:           ; lpMem
.text:1001D778                 push    edi
.text:1001D779                 push    0 ; dwFlags
.text:1001D77B                 push    esi ; hHeap
.text:1001D77C                 call    ds:HeapFree
```

```
.text:1001D782
.text:1001D782 loc_1001D782:
.text:1001D782                 add     esp, 28h
.text:1001D785                 pop     esi
.text:1001D786                 pop     edi
.text:1001D787                 pop     ebp
.text:1001D788                 retn
.text:1001D788 FN_GetFileAttributes_data_txt endp
.text:1001D788
```

Figure 5. One of the small functions responsible for API Hammering in ZLoader.

By using a debugger, we could figure out the number of calls made to four file I/O functions (see Figure 6). The large and smaller functions together generate more than a million function calls in total, without the use of a single large loop as seen in BazarLoader.



```
1232934  "ReadFile"
1232935  "CreateFileW"
1232936  "WriteFile"
1232937  "CreateFileW"
1232938  "WriteFile"
1232939  "CreateFileW"
1232940  "WriteFile"
1232941  "CreateFileW"
1232942  "WriteFile"
1232943  "CreateFileW"
1232944  "WriteFile"
1232945  GetFileAttributesW
1232946  "CreateFileW"
1232947  "WriteFile"
1232948  GetFileAttributesW
1232949  "CreateFileW"
1232950  "WriteFile"
```

Figure 6. Debugger log for APIs responsible for API Hammering in ZLoader.

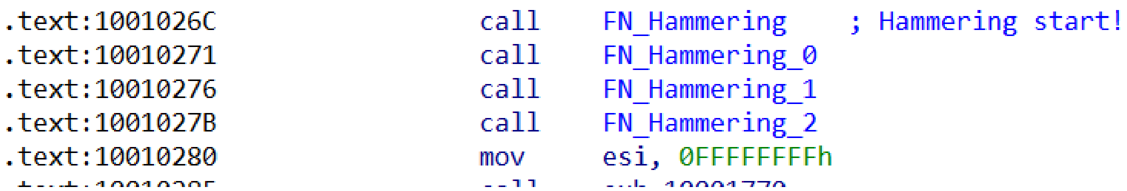
The following table shows the API function call counts made during our analysis process:

I/O API function	Total Call Count
ReadFile	278,850
WriteFile	280,921
GetFileAttributesW	113,389
CreateFileW	559,771

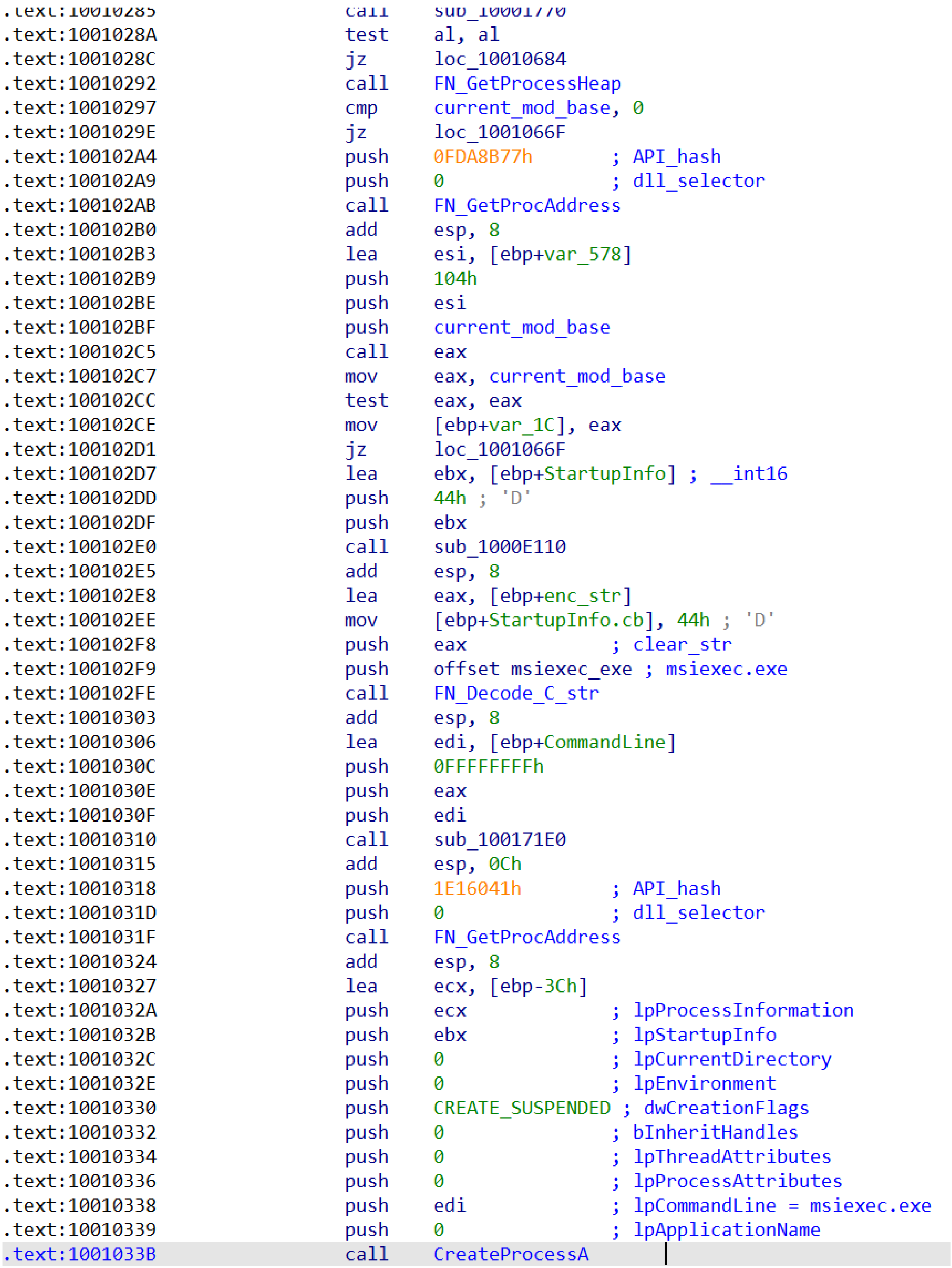
Table 1. API function call counts.

The execution time of the four large functions delays the injection of the Zloader payload. Without complete execution of these functions, the sample would appear to be a benign sample just carrying out file I/O operations.

The following disassembled code shows the four API hammering procedures followed by the injection procedures:



```
.text:1001026C                 call    FN_Hammering     ; Hammering start!
.text:10010271                 call    FN_Hammering_0
.text:10010276                 call    FN_Hammering_1
.text:1001027B                 call    FN_Hammering_2
.text:10010280                 mov     esi, 0FFFFFFFFh
.text:10010285                 call    sub_10001770
```



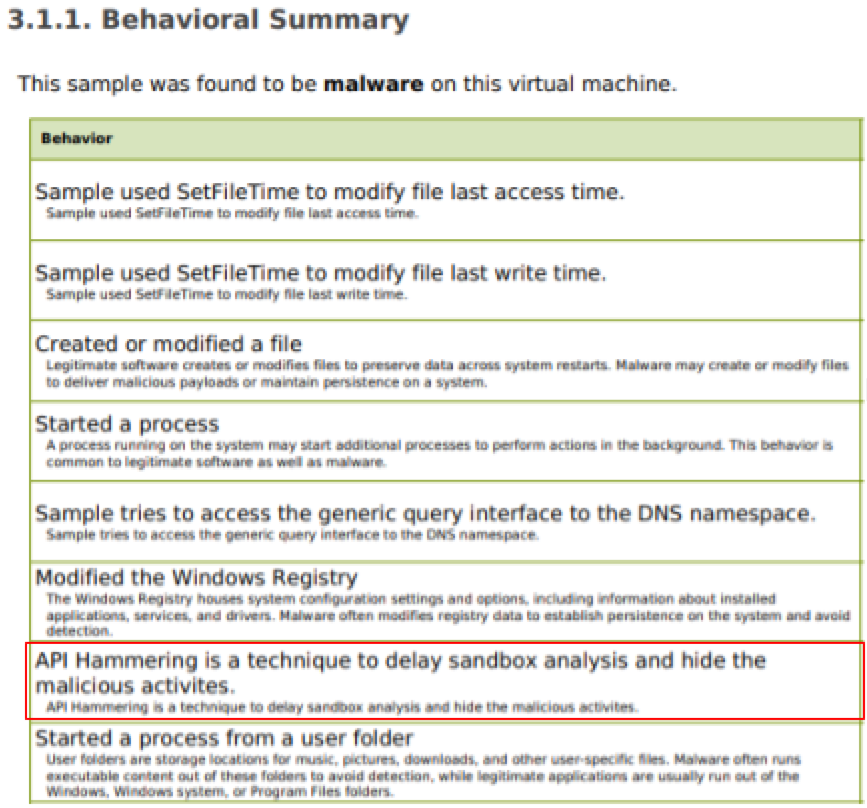
```
.text:10010285                 call    sub_10001770
.text:1001028A                 test    al, al
.text:1001028C                 jz      loc_10010684
.text:10010292                 call    FN_GetProcessHeap
.text:10010297                 cmp     current_mod_base, 0
.text:1001029E                 jz      loc_1001066F
.text:100102A4                 push    0FDA8B77h       ; API_hash
.text:100102A9                 push    0               ; dll_selector
.text:100102AB                 call    FN_GetProcAddress
.text:100102B0                 add     esp, 8
.text:100102B3                 lea     esi, [ebp+var_578]
.text:100102B9                 push    104h
.text:100102BE                 push    esi
.text:100102BF                 push    current_mod_base
.text:100102C5                 call    eax
.text:100102C7                 mov     eax, current_mod_base
.text:100102CC                 test    eax, eax
.text:100102CE                 mov     [ebp+var_1C], eax
.text:100102D1                 jz      loc_1001066F
.text:100102D7                 lea     ebx, [ebp+StartupInfo] ; __int16
.text:100102DD                 push    44h ; 'D'
.text:100102DF                 push    ebx
.text:100102E0                 call    sub_1000E110
.text:100102E5                 add     esp, 8
.text:100102E8                 lea     eax, [ebp+enc_str]
.text:100102EE                 mov     [ebp+StartupInfo.cb], 44h ; 'D'
.text:100102F8                 push    eax             ; clear_str
.text:100102F9                 push    offset msiexec_exe ; msiexec.exe
.text:100102FE                 call    FN_Decode_C_str
.text:10010303                 add     esp, 8
.text:10010306                 lea     edi, [ebp+CommandLine]
.text:1001030C                 push    0FFFFFFFFh
.text:1001030E                 push    eax
.text:1001030F                 push    edi
.text:10010310                 call    sub_100171E0
.text:10010315                 add     esp, 0Ch
.text:10010318                 push    1E16041h        ; API_hash
.text:1001031D                 push    0               ; dll_selector
.text:1001031F                 call    FN_GetProcAddress
.text:10010324                 add     esp, 8
.text:10010327                 lea     ecx, [ebp-3Ch]
.text:1001032A                 push    ecx             ; lpProcessInformation
.text:1001032B                 push    ebx             ; lpStartupInfo
.text:1001032C                 push    0               ; lpCurrentDirectory
.text:1001032E                 push    0               ; lpEnvironment
.text:10010330                 push    CREATE_SUSPENDED ; dwCreationFlags
.text:10010332                 push    0               ; bInheritHandles
.text:10010334                 push    0               ; lpThreadAttributes
.text:10010336                 push    0               ; lpProcessAttributes
.text:10010338                 push    edi             ; lpCommandLine = msiexec.exe
.text:10010339                 push    0               ; lpApplicationName
.text:1001033B                 call    CreateProcessA
```

Figure 6. API Hammering before payload injection in ZLoader.

Conclusion: WildFire vs API Hammering

Results from analyzing various implementations of API Hammering enabled the detection of malware samples using API Hammering for sandbox evasion in WildFire. WildFire detects the use of API Hammering by BazarLoader, Zloader, and other malware families.

The following excerpt from the WildFire report of our BazarLoader sample shows the detected entry for API Hammering.



3.1.1. Behavioral Summary

This sample was found to be **malware** on this virtual machine.

Behavior
Sample used SetFileTime to modify file last access time. Sample used SetFileTime to modify file last access time.
Sample used SetFileTime to modify file last write time. Sample used SetFileTime to modify file last write time.
Created or modified a file Legitimate software creates or modifies files to preserve data across system restarts. Malware may create or modify files to deliver malicious payloads or maintain persistence on a system.
Started a process A process running on the system may start additional processes to perform actions in the background. This behavior is common to legitimate software as well as malware.
Sample tries to access the generic query interface to the DNS namespace. Sample tries to access the generic query interface to the DNS namespace.
Modified the Windows Registry The Windows Registry houses system configuration settings and options, including information about installed applications, services, and drivers. Malware often modifies registry data to establish persistence on the system and avoid detection.
API Hammering is a technique to delay sandbox analysis and hide the malicious activites. API Hammering is a technique to delay sandbox analysis and hide the malicious activites.
Started a process from a user folder User folders are storage locations for music, pictures, downloads, and other user-specific files. Malware often runs executable content out of these folders to avoid detection, while legitimate applications are usually run out of the Windows, Windows system, or Program Files folders.

Figure 7. WildFire detected API Hammering along with other behaviors in a Bazarloader sample.

Palo Alto Networks customers receive further protections against other malware families using similar sandbox evasion techniques through Cortex XDR or our Next-Generation Firewall with WildFire and Threat Prevention security subscriptions.

Indicators of Compromise

BazarLoader Sample
ce5ee2fd8aa4acda24baf6221b5de66220172da0eb312705936adc5b164cc052

Zloader Sample
44ede6e1b9be1c013f13d82645f7a9cff7d92b267778f19b46aa5c1f7fa3c10b

Get updates from Palo Alto Networks!

Sign up to receive the latest news, cyber threat intelligence and research from us