Bandios

Alföldi Mátyás

June 24, 2020

Contents

Introduction

File: Onlineinstaller.bin
VT: https://www.virustotal.com/gui/file/59c662a5207c6806046205348b22ee45da3f685fe022556716dbbd6643e61834

VT report

All the sections have an entropy of 6+, and the .rsrc section is close to 8, so there will definitely be some unpacking.
There is also a zip file, which is related to plenty of malware, but isn't detected, I would say that it is most likely password protected, like in the case of WannaCry.
It also has quite a few exports, which is rather strange for an exe.
Interesting Imports:

- GetAdaptersInfo
- Http* functions
- Internet* functions
- _TrackMouseEvent (Anti automatic sandbox analysis?)
- CreateEvent,CreateThread,CreateProcess,WaitForSingleObject combo
- Sleep (Anti vm?)
- SetUnhandledExceptionFilter+UnhandledExceptionFilter combo
- LoadLibrary* functions + GetProcAddress
- Various file operations Copy/Move/Read/Write
- Heap related functions
- Tls functions
- System information gathering
- Resource handling functions
- GetCurrentDirectory/TempPath + SetCurrentDirectory
- DuplicateHandle (possibly used as a defense mechanism)
- QueryPerformanceCounter
- IsProcessorFeaturePresent
- GetTickCount
- Service related functions
- Registry related functions
- CreateStreamOnHGlobal

- Com related functions + Variant
- File time gathering/modification

Network related:

- DNS resolution for iostream.system.band
- HTTP request to iostream.system.band/dump/io/time.php

File readings:

- Internet cache/history/cookies
- autoexec.bat
- On some systems it opens \\.\PIPE\lsarpc and \\.\PIPE\ROUTER

File modifications:

- <sys32>\spoolsr.exe
- <sys32>\MS.dat
- <sys32>\KeyHook32.dll
- <sys32>\KH.dat
- <sys32>\usp20.dll
- <sys32>\UP.dat
- <sys32>\drivers\iaStorE.sys
- <tmp>\996E.tmp

Based on the above spoolsr.exe-MS.dat KeyHook32.dll-KH.dat usp20.dll-UP.dat belong together.
The first is related to printers, and possibly data sent out for printing will be stored in MS.dat.
The second is related to keylogging.
And the driver is a defense mechanism.
Registry modifications:

- Deletes proxy related keys.
- Sets the SavedLegacySettings key

Services related:

- Creates the iaStorE service (the above mentioned driver) and also opens RASMAN

Mutexes:

- RasPbFile
- ZonesCounterMutex
- ZonesCacheCounterMutex
- ZonesLockedCacheCounterMutex

Static Analysis

One has to start here from the entry point, since it looks similar to a normal msvc built app, but there are some differences.
Also CRC32,zinflate possibly for the zip file in the .rsrc, and rijndael_td and _te constants can be found, the later signaling that a rijndael implementation is built into this malware.
Note: Functions/Variables named by me, will be in italic.

0.1 entry

- It starts with the usual ___security_init_cookie()
- *GetShowWindow* call, which is sort of junk code, since the return value isn't used anywhere.
- In the next function call a global variable is set to 2
- *TryGetProcessHeap* is called and if it fails we call _fast_error_exit with 0x1c
- Next comes *Init* which initializes proc addresses, fls, and tls

0.2 GetShowWindow

GetStartupInfoW is called and based on if dwFlags contains STARTF_USESSHOWWINDOW it either returns wShowWindow or 10 for SW_SHOWDEFAULT.

0.3 TryGetProcessHeap

Stores in a global variable (*hProcessHeap*) the return value of GetProcessHeap, and returns *hProcessHeap* != 0

0.4 Init

- First subcall is *InitProcAddresses*
  - First it calls EncodePointer with 0 and calls some setter functions for some pointers
  - In between these setter calls, the address of the terminate function gets stored too in a global variable after run through EncodePointer
  - Next we call GetModuleHandleW on kernel32.dll
  - Finally we use GetProcAddress and store the returned value xor BB40E64E in global variables

- Next is *InitCritSections* where we iterate over what seems like an array of LPCRITICAL_SECTIONS and we call either InitializeCriticalSectionEx (if *InitProcAddresses* managed to find it), or InitializeCriticalSectionAndSpinCount on it.
- Next a function pointer is called, and some further initialization occurs
- Finally we call main

0.5 main

- It gets a handle to the current process, opens its token, and tries to give it SeDebugPrivilege, and SeLoadDriverPrivilege.
- Afterwards it checks with the help of GetModuleFileNameW and a self written/statically linked sprintf variation and an overly complicated compare if the -install flag was given. Lets see first where we go if the install flag isn't given. (First start)
    - A Copy happens to the Tmp dir and a new process gets created most likely with the -install flag
    - Next an unnamed event is created with bManualReset set to TRUE, and starts a Thread at 4079c0 (passed the event handle), and waits for its completion.
    - In the started thread it seems like information is first gathered and afterwards it connects to iostream.system.band.
    - It does some other stuff, which I'll have to look into more, but in the end it calls SetEvent on the eventHandle, which was passed during thread creation.
    - based on if it is (x86 or Intel Itanium based processor) or something else it will do one of the following:
        - Disable Wow64FsRedirection
        - Files get created in the SystemDirectory (In memory Xor-ing with 0xdd reveals the files.)
        - Resources get loaded and a new file is created, based on it. (The files are zlib compressed)
        - Revers Wow64FsRedirection
        - Do the same as above without Wow64FsRedirection
        - Create and start a service
    - In the end it installs a Crash Dump Filter driver through registry modifications.

0.6 PE files hidden in the .rsrc

0.6.1 104:PE32+,DLL

Seems relatively short, in the first function(InitVars) it sets to variables, one to the value 2b992ddfa232 and the negated value into another. If the first parameter to the entry is 0 it calls a function(FUNCTION2) and sets the return value to 0xc0000009a (Insufficient system resources?) FUNCTION2: Inside the function some xor-ing happens in the .data section first, which based on the following RtlInitUnicodeString are strings. These are then passed further down to a function(FUNCTION3) in pairs. Next 2 function calls happen to FUNCTION4, with the first value being an address to a variable, the second a 0, and the third what seems like a size value FUNCTION3:

0.6.2 106:PE32,DLL

0.6.3 108:PE32

iaStorE.sys (compared hash to the dropped file in an any.run analysis)

0.6.4 110:PE32+

0.6.5 111:PE32

Todo: look into the In memory Xored + zlib compressed PE files.