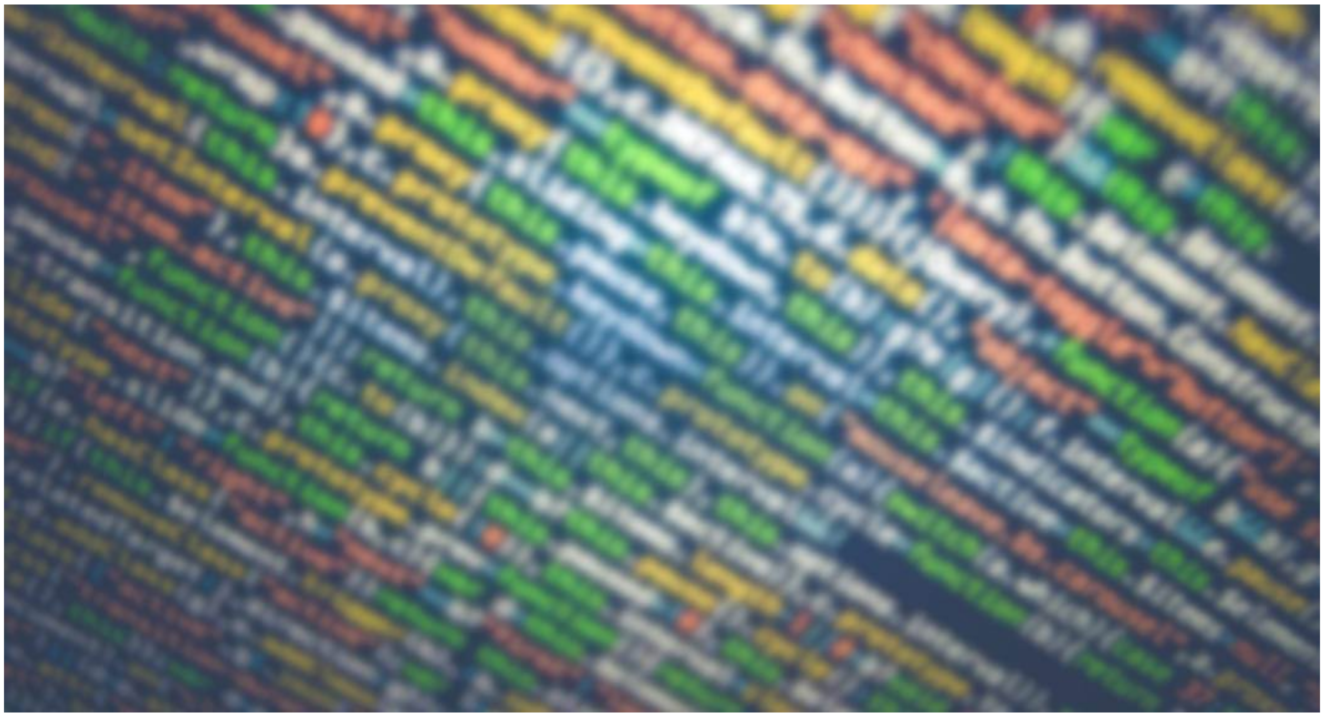
Ragnarok Stopper: development of a vaccine

tarlogic.com/blog/ragnarok-malware-stopper-vaccine/

May 4, 2020



The field of reverse engineering and specifically malware analysis within the Compromise Assessment process is of vital importance. Beyond the analysis of logs, events, network connections, alerts generated by IDS and firewalls, etc., experience tells us that a preliminary and quick analysis of a suspicious binary (whenever possible) can offer high-value intelligence, not just to get more context about an incident (TTP, C2, persistence, timeline, etc.) but to develop tools or techniques to mitigate a campaign still underway. The following case is an example of this.

In a recent incident we obtained a binary packaged with **Ragnarok**, a malware widely used in the last months in various campaigns. The aim of this post is to describe a possible **vaccine** that can stop the threat posed by some of the samples that we have analyzed so far.

Developing the vaccine

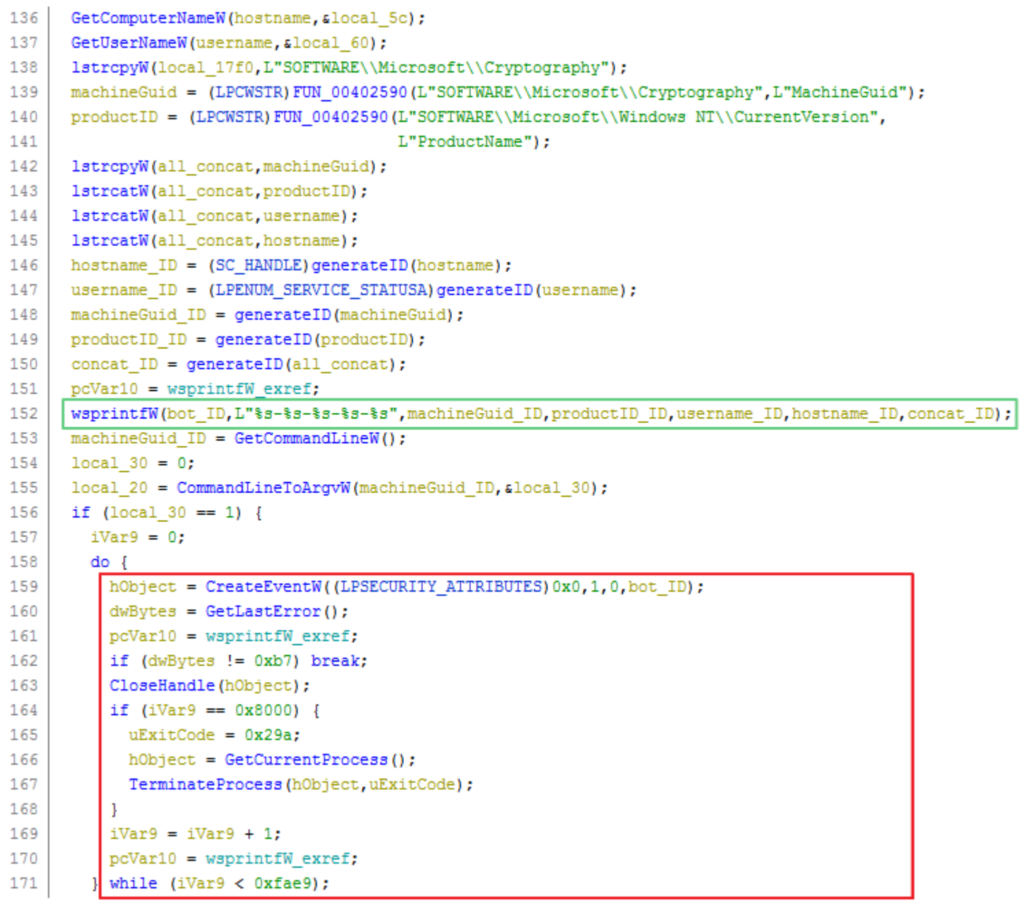
Just before harmful actions start (basically encrypting files with RSA 2048 + AES) the malware computes a fingerprint based on 5 values:

- MachineGUID
- Product Name
- Username

- Hostname
- A string concatenating the previous four elements

For each of these values, an 8-byte ID is generated, which are joined by a "-". The fingerprint obtained (a string with the following form: XXXXXXXX-XXXXXXXX-XXXXXXXX-XXXXXXXX-XXXXXXXX) is used to create an event object via `CreateEventW` with which to check if an instance is already running.

This sort of actions is very common in malware to guarantee the execution of a single process, generally with Mutex and Event objects. What is interesting is that these checks are very useful for creating **vaccines that prevent malware from running**. The following screenshot shows the logic previously described.

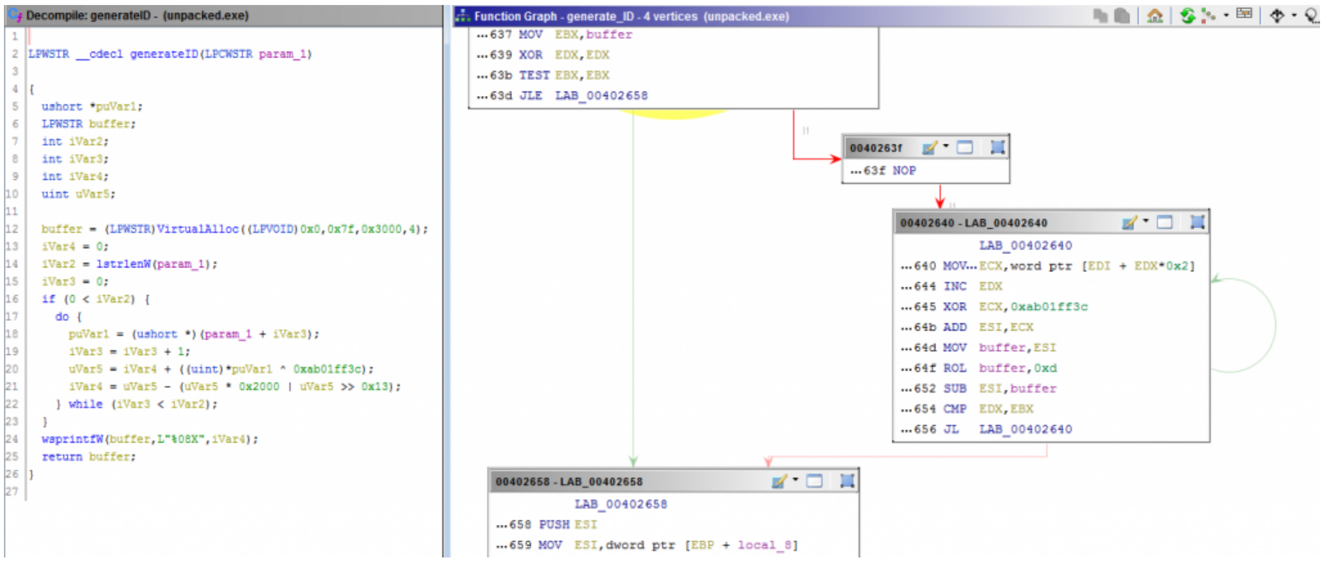


```
GetComputerNameW(hostname,&local_5c);
GetUserNameW(username,&local_60);
lstrcpyW(local_17f0,L"SOFTWARE\\Microsoft\\Cryptography");
machineGuid = (LPCWSTR)FUN_00402590(L"SOFTWARE\\Microsoft\\Cryptography",L"MachineGuid");
productID = (LPCWSTR)FUN_00402590(L"SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion",
                                  L"ProductName");
lstrcpyW(all_concat,machineGuid);
lstrcatW(all_concat,productID);
lstrcatW(all_concat,username);
lstrcatW(all_concat,hostname);
hostname_ID = (SC_HANDLE)generateID(hostname);
username_ID = (LPENUM_SERVICE_STATUSA)generateID(username);
machineGuid_ID = generateID(machineGuid);
productID_ID = generateID(productID);
concat_ID = generateID(all_concat);
pcVar10 = wsprintfW_exref;
wsprintfW(bot_ID,L"%s-%s-%s-%s-%s",machineGuid_ID,productID_ID,username_ID,hostname_ID,concat_ID);
machineGuid_ID = GetCommandLineW();
local_30 = 0;
local_20 = CommandLineToArgvW(machineGuid_ID,&local_30);
if (local_30 == 1) {
  iVar9 = 0;
  do {
    hObject = CreateEventW((LPSECURITY_ATTRIBUTES)0x0,1,0,bot_ID);
    dwBytes = GetLastError();
    pcVar10 = wsprintfW_exref;
    if (dwBytes != 0xb7) break;
    CloseHandle(hObject);
    if (iVar9 == 0x8000) {
      uExitCode = 0x29a;
      hObject = GetCurrentProcess();
      TerminateProcess(hObject,uExitCode);
    }
    iVar9 = iVar9 + 1;
    pcVar10 = wsprintfW_exref;
  } while (iVar9 < 0xfae9);
```

Ragnarok running instance check

In this case, note that once the identifier is generated, it is used to create an event object with that name. If the returned handle belongs to an existing object (`GetLastError() == ERROR_ALREADY_EXISTS`) it retries it for a total of 0x8000 times. If this counter reaches this value, `TerminateProcess()` is invoked, thus avoiding the destructive actions of the ransomware.

With this information creating a vaccine that forces the ransomware to finish its execution is trivial. It would be enough to create an event object whose name follows the format used by Ragnarok. The following function shows the logic used to generate the ID from each string.


Decompile: generateID - (unpacked.exe)

```
LPWSTR __cdecl generateID(LPCWSTR param_1)

{
  ushort *puVar1;
  LPWSTR buffer;
  int iVar2;
  int iVar3;
  int iVar4;
  uint uVar5;

  buffer = (LPWSTR)VirtualAlloc((LPVOID)0x0,0x7f,0x3000,4);
  iVar4 = 0;
  iVar2 = lstrlenW(param_1);
  iVar3 = 0;
  if (0 < iVar2) {
    do {
      puVar1 = (ushort *)(param_1 + iVar3);
      iVar3 = iVar3 + 1;
      uVar5 = iVar4 + ((uint)*puVar1 ^ 0xab01ff3c);
      iVar4 = uVar5 - (uVar5 * 0x2000 | uVar5 >> 0x13);
    } while (iVar3 < iVar2);
  }
  wsprintfW(buffer,L"%08X",iVar4);
  return buffer;
}
```

Function Graph - generate_ID - 4 vertices (unpacked.exe)

```
...637 MOV  EBX,buffer
...639 XOR  EDX,EDX
...63b TEST EBX,EBX
...63d JLE  LAB_00402658
```

0040263f

```
...63f NOP
```

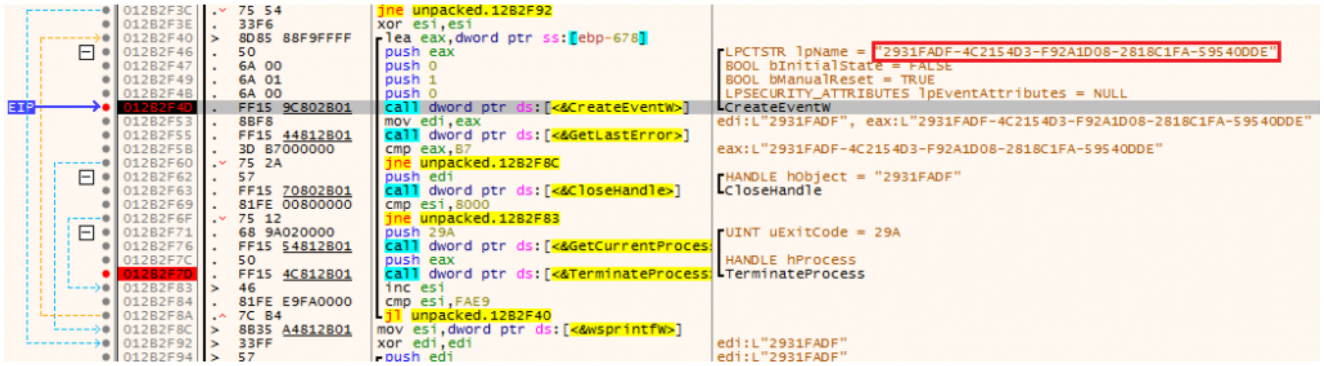
00402640 - LAB_00402640

```
             LAB_00402640
...640 MOV...ECX,word ptr [EDI + EDX*0x2]
...644 INC  EDX
...645 XOR  ECX,0xab01ff3c
...64b ADD  ESI,ECX
...64d MOV  buffer,ESI
...64f ROL  buffer,0xd
...652 SUB  ESI,buffer
...654 CMP  EDX,EBX
...656 JL   LAB_00402640
```

00402658 - LAB_00402658

```
             LAB_00402658
...658 PUSH ESI
...659 MOV  ESI,dword ptr [EBP + local_8]
```

Fingerprint generation



```
012B2F3C  .^  75 54           jne unpacked.12B2F92
012B2F3E  .   33F6            xor esi,esi
012B2F40  >   8D85 88F9FFFF   lea eax,dword ptr ss:[ebp-678]
012B2F46  .   50              push eax                                   LPCTSTR lpName = "2931FADF-4C2154D3-F92A1D08-2818C1FA-59540DDE"
012B2F47  .   6A 00           push 0                                     BOOL bInitialState = FALSE
012B2F49  .   6A 01           push 1                                     BOOL bManualReset = TRUE
012B2F4B  .   6A 00           push 0                                     LPSECURITY_ATTRIBUTES lpEventAttributes = NULL
EIP 012B2F4D  .   FF15 9C802B01   call dword ptr ds:[<&CreateEventW>]    CreateEventW
012B2F53  .   8BF8            mov edi,eax                                edi:L"2931FADF", eax:L"2931FADF-4C2154D3-F92A1D08-2818C1FA-59540DDE"
012B2F55  .   FF15 44812B01   call dword ptr ds:[<&GetLastError>]
012B2F5B  .   3D B7000000     cmp eax,B7                                 eax:L"2931FADF-4C2154D3-F92A1D08-2818C1FA-59540DDE"
012B2F60  .^  75 2A           jne unpacked.12B2F8C
012B2F62  .   57              push edi                                   HANDLE hObject = "2931FADF"
012B2F63  .   FF15 70802B01   call dword ptr ds:[<&CloseHandle>]         CloseHandle
012B2F69  .   81FE 00800000   cmp esi,8000
012B2F6F  .^  75 12           jne unpacked.12B2F83
012B2F71  .   68 9A020000     push 29A                                   UINT uExitCode = 29A
012B2F76  .   FF15 54812B01   call dword ptr ds:[<&GetCurrentProces
012B2F7C  .   50              push eax                                   HANDLE hProcess
012B2F7D  .   FF15 4C812B01   call dword ptr ds:[<&TerminateProcess      TerminateProcess
012B2F83  >   46              inc esi
012B2F84  .   81FE E9FA0000   cmp esi,FAE9
012B2F8A  .^  7C B4           jl unpacked.12B2F40
012B2F8C  >   8B35 A4812B01   mov esi,dword ptr ds:[<&wsprintfW>]
012B2F92  >   33FF            xor edi,edi                                edi:L"2931FADF"
012B2F94  >   57              push edi                                   edi:L"2931FADF"
```

Fingerprint generation

A cleaner version of the `GenerateID` function is shown below:

```
#define __ROL__(x, y) _rotl(x, y)
inline unsigned int __ROL4__(unsigned int value, int count) { return
__ROL__((unsigned int)value, count); }
inline unsigned int __ROR4__(unsigned int value, int count) { return
__ROL__((unsigned int)value, -count); }

LPWSTR GenerateID(LPCWSTR lpString)
{
    unsigned int inc;
    int len, i, acu;
    LPWSTR codeID;

    inc = 0;
    codeID = (LPWSTR)VirtualAlloc(0, 0x7Fu, MEM_RESERVE | MEM_COMMIT,
PAGE_READWRITE);
    len = lstrlenW(lpString);
    for (i = 0; i < len; inc = (acu ^ 0xAB01FF3C) + inc - __ROL4__((acu ^ 0xAB01FF3C)
+ inc, 13))
        acu = lpString[i++];
    wsprintfW(codeID, L"%08X", inc);
    return codeID;
}
```

The code of the PoC Ragnarok Stopper can be **found in our repository**. It should be noted that this is just a vaccine that applies to some of the samples that we have analyzed so far. New variants of Ragnarok may use other criteria to identify a running instance.

https://youtu.be/xUC07prttZs