# SFG: Furtim's Derivative

## By Joseph Landry and Udi Shamir

The Labs team at SentinelOne recently discovered a sophisticated malware campaign specifically targeting at least one European energy company. Upon discovery, the team reverse engineered the code and believes that based on the nature, behavior and sophistication of the malware and the extreme measures it takes to evade detection, it likely points to a nation-state sponsored initiative, potentially originating in Eastern Europe.

The malware is most likely a dropper tool being used to gain access to carefully targeted network users, which is then used either to introduce the payload, which could either work to extract data or insert the malware to potentially shut down an energy grid. The exploit affects all versions of Microsoft Windows and has been developed to bypass traditional antivirus solutions, next-generation firewalls, and even more recent endpoint solutions that use sandboxing techniques to detect advanced malware. (biometric readers are non-relevant to the bypass / detection techniques, the malware will stop executing if it detects the presence of specific biometric vendor software).

We believe the malware was released in May of this year and is still active. It exhibits traits seen in previous nation-state Rootkits, and appears to have been designed by multiple developers with high-level skills and access to considerable resources.

We validated this malware campaign against SentinelOne and confirmed the steps outlined below were detected by our Dynamic Behavior Tracking (DBT) engine.

## Malware Synopsis

This sample was written in a manner to evade static and behavioral detection. Many anti-sandboxing techniques are utilized. Analysts relying solely on sandbox solutions such as GFI and Joe Sandbox to analyze this malware will miss full functionality of the sample.

Two known exploits (CVE-2014-4113 and CVE-2015-1701) were found in the sample, as well as one UAC bypass.

The sample appears to be targeting facilities that not only have software security in place, but physical security as well. ZKTeco (http://www.zkteco.com/) is a global manufacturer of access control systems including facial recognition, fingerprint scanners, and RFID. If the sample is run on a workstation with ZKTeco's ZKAccess software installed, the process will prematurely terminate. These systems would be heavily scrutinized by their administrators, and an infection on one of these machines would likely not go unnoticed.

Two hard coded MAC addresses are checked for by the sample. A MAC address is unique 6-byte number that is burned into the chips of all network cards. The sample will prematurely terminate if the machine it is running on has one of these two MAC addresses.

Use of low-level API ( `Nt* and Rtl*` ) and direct system calls ( `INT 2Eh` and `CALL ntdll!KiFastSystemCall` )
were used to bypass user-space hooks used by antivirus software and sandboxes. This also demonstrates the expertise of the author. Many of these low-level APIs and system calls are undocumented/under-documented and can change between different versions of Windows. To gain an understanding of these functions, one has to be familiar with the Windows Driver Development Kit (DDK), and also reverse-engineered portions of the Windows operating system.

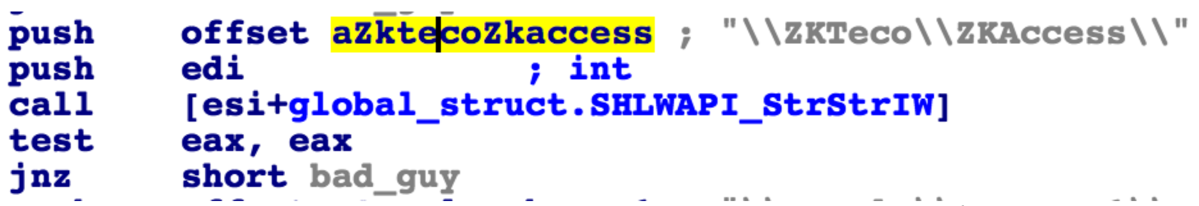
The use of indirect subroutine calls make manual static analysis nearly impossible, and manual dynamic analysis painful and slow. The author took special care to keep this sample undetected for as long as possible.

The main goal of the sample analyzed is to run its final payload after silently removing a number of antivirus products.

## Overview of Execution

The sample starts by rigorously checking its environment. If in a sandbox or under manual inspection by an analyst, the sample will prematurely terminate. If the sample finds specific antivirus software installed, it will carefully enable and disable specific functionality to evade behavioral detection.

In many situations, the sample will distance itself from malicious behaviors by invoking cmd.exe to do its dirty work. For example, modifying sensitive registry values are done by invoking cmd.exe /c reg.exe …. Unfortunately for this sample, SentinelOne tracks the full context of processes to determine the root cause of malicious behavior.



```
push    offset aZktecoZkaccess ; "\\ZKTeco\\ZKAccess\\"
push    edi             ; int
call    [esi+global_struct.SHLWAPI_StrStrIW]
test    eax, eax
jnz     short bad_guy
```

From this point on, the sample's goal is to remove any antivirus software before running its final payload. To accomplish this goal, the sample must be run as administrator. Two known local privilege escalation exploits are included in the sample (CVE-2014- 4113 and CVE-2015-1701), as well as one UAC bypass, which are used to acquire administrator access. As a last resort, the sample will use a UAC prompt to try and elevate itself to administrator. Once the sample is running as administrator, it will add the current user to the local Administrator group, allowing it to maintain administrator access in the future.

The sample now writes its Native Application binary to disk. Unlike regular application code, this binary can only link to `ntdll.dll`. It will run at a point in the boot-up process where some Windows subsystems are not yet initialized, and therefore can not call into normal dlls like `kernel32.dll` and `user32.dll`. This Native Application is hidden in an NTFS Alternative Data Stream (ADS) at the path `C:\Windows\Temp:1`. By using ADS, the file will not be visible by normal file browsers, like `explorer.exe`. The Native Application is registered to run on boot-up altering the values `SetupExecute` and `BootExecute` in the registry key `HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\\`.

To ensure the success of the Native Application, the sample will remove all filter drivers from running after reboot by removing their associated registry entries. Filter drivers are use by anti-virus software to intercept file and network access to run static detection on the contents of the traffic. These drivers are loaded early in the boot process, and could interfere with the execution of the Native Application.

The system is now forced to reboot, allowing the Native Application to run. The Native Application also has similar checks to tell if it is running in a sandbox, and will terminate prematurely when one is detected.

The Native Application's goal is to remove any anti-virus software that is installed on the system and drop its final payload. By running during the boot process, and after the preperation that was done in the previous stage, the Native Application has full control over the system. Removing any anti-virus is trivial at this point because the anti-virus software is not running. The Native Binary writes the final payload of the sample to disk under the filename `rdpinst.exe` and registers it to be run later in the boot process by creating a registry value in `\Registry\Machine\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce`.

## Architecture

There is one large structure that is allocated on the heap. This structure contains mostly function pointers to external libraries and internal function pointers. This creates a problem for static analysis. There are many indirect calls (e.g. CALL EAX) obscuring the program flow for static analysis. This structure is passed as the first parameter to almost every function in the binary.

A large chunk of the .data region is encrypted using RC4. This encrypted region contains the string literals for the sample, creating another problem for static analysis and static detection. Before the process is terminated, this region is re-encrypted, possibly to deter an analyst from recovering the unencrypted contents by using memory dumps.

Included in this encrypted region are three binary blobs that are also encrypted and compressed: final payload, a Windows Native API application; A DLL with a UAC bypass; and a 64-bit executable an exploit for CVE-2014-4113.

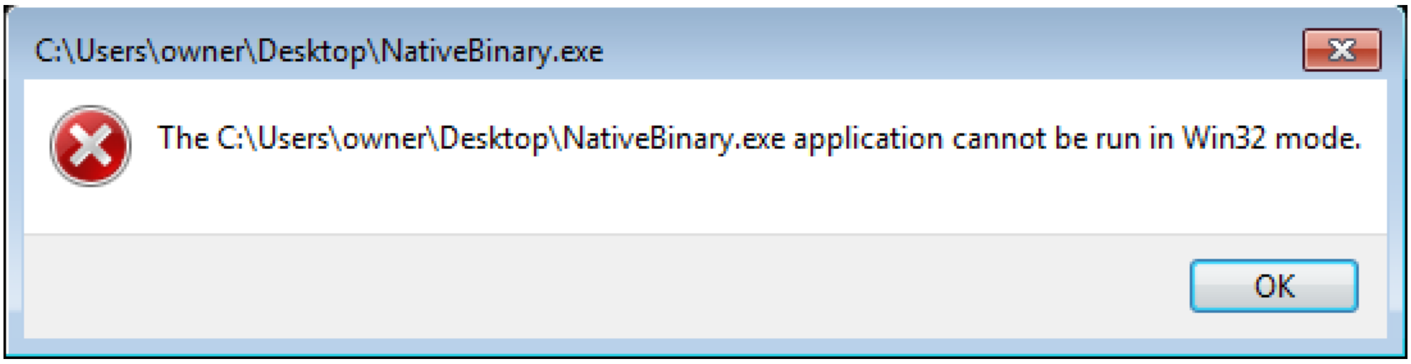
## Reversing Techniques Used

To reverse the main sample, I developed a python script to patch out the blacklist and NOP out some test code. By placing a breakpoint on the function that gets called to prematurely terminate the process, we were able to identify checks that failed by inspecting the return address on the call stack.

Zeroing out the relocation size in the PE Data Directory also made jumping between IDA and OllyDBG easier because the base address of the executable was not randomized.

Noting the destination address of indirect jumps in IDA comments made reviewing after debugging much simpler.

To debug the Native Application binary, I patched the PE Optional Header field `Subsystem` field from 1 to 2. This changed the subsystem used by the binary from `Native` to `WindowsGUI`. This will let the binary run after bootup is finished, instead of getting this error message:


C:\Users\owner\Desktop\NativeBinary.exe

The C:\Users\owner\Desktop\NativeBinary.exe application cannot be run in Win32 mode.

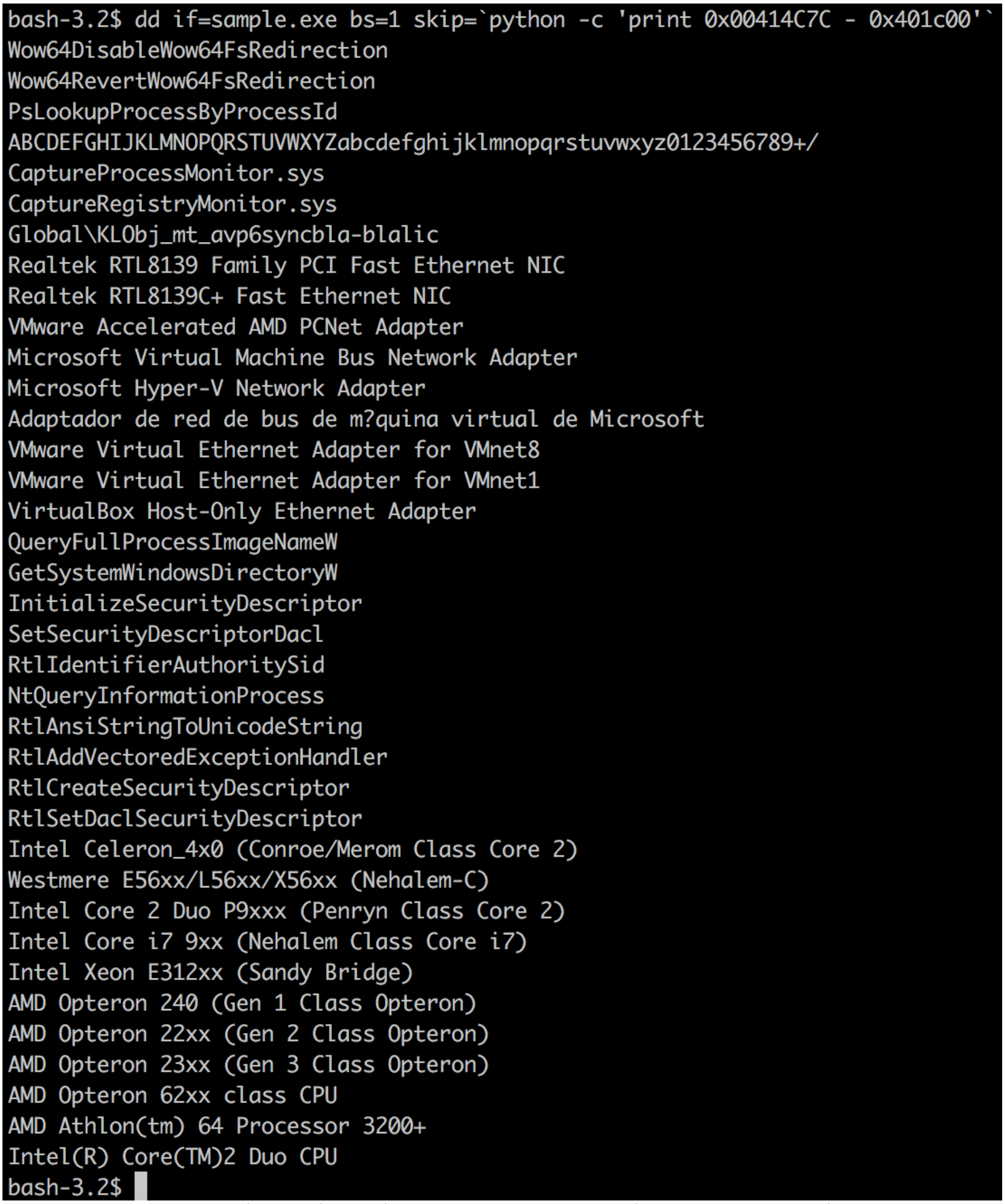
## "Packing"

The code of the main executable (.text segment) isn't packed, but a region in the .data section is encrypted using RC4 with the password "dqrChZonUF". The RC4 implementation looks like a direct copy of the code found in the FreeBSD and XNU kernel:

- https://github.com/freebsd/freebsd/blob/master/sys/crypto/rc4/rc4.c
- http://opensource.apple.com//source/xnu/xnu-1456.1.26/bsd/crypto/rc4/rc4.c

The only modification to the BSD RC4 implementation is the pointer to the global struct containing the function pointers to the RC4 subroutines.

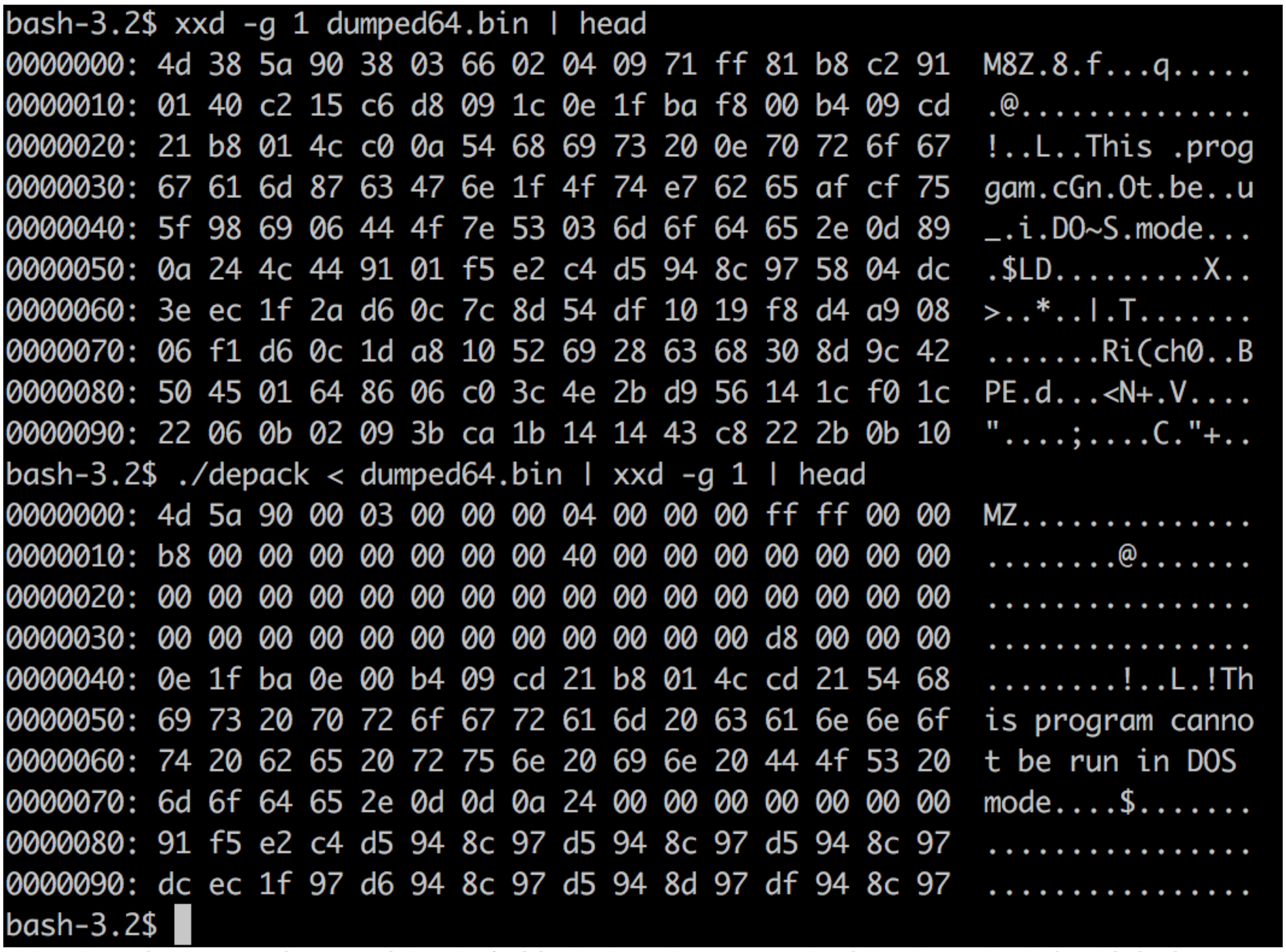
After decrypting this large section, all the string literals are uncovered.



```
bash-3.2$ dd if=sample.exe bs=1 skip=`python -c 'print 0x00414C7C - 0x401c00'`
Wow64DisableWow64FsRedirection
Wow64RevertWow64FsRedirection
PsLookupProcessByProcessId
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
CaptureProcessMonitor.sys
CaptureRegistryMonitor.sys
Global\KLObj_mt_avp6syncbla-blalic
Realtek RTL8139 Family PCI Fast Ethernet NIC
Realtek RTL8139C+ Fast Ethernet NIC
VMware Accelerated AMD PCNet Adapter
Microsoft Virtual Machine Bus Network Adapter
Microsoft Hyper-V Network Adapter
Adaptador de red de bus de m?quina virtual de Microsoft
VMware Virtual Ethernet Adapter for VMnet8
VMware Virtual Ethernet Adapter for VMnet1
VirtualBox Host-Only Ethernet Adapter
QueryFullProcessImageNameW
GetSystemWindowsDirectoryW
InitializeSecurityDescriptor
SetSecurityDescriptorDacl
RtlIdentifierAuthoritySid
NtQueryInformationProcess
RtlAnsiStringToUnicodeString
RtlAddVectoredExceptionHandler
RtlCreateSecurityDescriptor
RtlSetDaclSecurityDescriptor
Intel Celeron_4x0 (Conroe/Merom Class Core 2)
Westmere E56xx/L56xx/X56xx (Nehalem-C)
Intel Core 2 Duo P9xxx (Penryn Class Core 2)
Intel Core i7 9xx (Nehalem Class Core i7)
Intel Xeon E312xx (Sandy Bridge)
AMD Opteron 240 (Gen 1 Class Opteron)
AMD Opteron 22xx (Gen 2 Class Opteron)
AMD Opteron 23xx (Gen 3 Class Opteron)
AMD Opteron 62xx class CPU
AMD Athlon(tm) 64 Processor 3200+
Intel(R) Core(TM)2 Duo CPU
bash-3.2$
```

Also, there are other regions inside of this decrypted region containing more encrypted blobs, like a Matryoshka doll. These contain a Native executable, the UAC bypass DLL, and the 64-bit implementation of the exploit for CVE-2014-4113. Furthermore, the Native binary contains another binary blob, that is the compressed and encrypted final payload.

Team member, Caleb Fenton, correctly identified the compressed stream format used for these blobs as aPLib.



```
bash-3.2$ xxd -g 1 dumped64.bin | head
0000000: 4d 38 5a 90 38 03 66 02 04 09 71 ff 81 b8 c2 91  M8Z.8.f...q.....
0000010: 01 40 c2 15 c6 d8 09 1c 0e 1f ba f8 00 b4 09 cd  .@..............
0000020: 21 b8 01 4c c0 0a 54 68 69 73 20 0e 70 72 6f 67  !..L..This .prog
0000030: 67 61 6d 87 63 47 6e 1f 4f 74 e7 62 65 af cf 75  gam.cGn.Ot.be..u
0000040: 5f 98 69 06 44 4f 7e 53 03 6d 6f 64 65 2e 0d 89  _.i.DO~S.mode...
0000050: 0a 24 4c 44 91 01 f5 e2 c4 d5 94 8c 97 58 04 dc  .$LD.........X..
0000060: 3e ec 1f 2a d6 0c 7c 8d 54 df 10 19 f8 d4 a9 08  >..*..|.T.......
0000070: 06 f1 d6 0c 1d a8 10 52 69 28 63 68 30 8d 9c 42  .......Ri(ch0..B
0000080: 50 45 01 64 86 06 c0 3c 4e 2b d9 56 14 1c f0 1c  PE.d...<N+.V....
0000090: 22 06 0b 02 09 3b ca 1b 14 14 43 c8 22 2b 0b 10  "....;....C."+..
bash-3.2$ ./depack < dumped64.bin | xxd -g 1 | head
0000000: 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00  MZ..............
0000010: b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ........@.......
0000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000030: 00 00 00 00 00 00 00 00 00 00 00 00 d8 00 00 00  ................
0000040: 0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68  ........!..L.!Th
0000050: 69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f  is program canno
0000060: 74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20  t be run in DOS 
0000070: 6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00  mode....$.......
0000080: 91 f5 e2 c4 d5 94 8c 97 d5 94 8c 97 d5 94 8c 97  ................
0000090: dc ec 1f 97 d6 94 8c 97 d5 94 8d 97 df 94 8c 97  ................
bash-3.2$ 
```

Although RC4 isn't an esoteric stream cipher, the decision by the author to use such a cipher shows a level of sophistication not seen in typical crimeware.

## Anti-Debug, Anti-Sandbox, Anti-AV

The sample has an overwhelming number of checks to determine if it is in a sandbox, or if an antivirus application is installed. But why would the author go through so much trouble to evade sandboxes and AV products?

The strategy used by the author seems to be this:

- If we are running in a virtual machine, sandbox, or under manual inspection by an analyst, encrypt the .data section and terminate prematurely.
- If we are in an environment with Anti-Virus products installed, carefully enable and disable behavoirs of the infection to avoid behavoiral detection.

The folowing is a list of checks the sample performs in the order that they are executed.

### CPUID Check

This test is the least invasive of all the tests performed. It also would be hard for a virtualization-based sandbox to detect, because the CPUID instruction would be run on the physical CPU, and can't be hooked.By running this test first, it will insure that the sandbox log would not show any evidence of the process trying to inspect its environment. An analyst might dismiss the sample, because it doesn't appear to be trying to detect the sandbox or virtual machine.The x86 instruction CPUID will report back features of the CPU. This instruction is normally used to check what features are supported by the CPU to avoid an "Invalid Instruction" exception before executing feature specific code. The sample uses this instruction to find artifacts of a virtual machine.When the CPUID instruction is executed and the register EAX set to 0x80000002, 0x80000003, or 0x80000004, the CPU fills registers EAX, EBX, ECX, and EDX with the "Product Brand String." If the brand string is found in the sample's blacklist, the process will prematurely terminate.

Strings check by CPUID where EAX=0x8000000x:

```
Intel(R) Xeon(R) CPU
Common KVM processor
Common 32-bit KVM
Virtual CPU
Intel Celeron_4x0 (Conroe/Merom Class Core 2)
Westmere E56xx/L56xx/X56xx (Nehalem-C)
Intel Core 2 Duo P9xxx (Penryn Class Core 2)
Intel Core i7 9xx (Nehalem Class Core i7)
Intel Xeon E312xx (Sandy Bridge)
AMD Opteron 240 (Gen 1 Class Opteron)
AMD Opteron 22xx (Gen 2 Class Opteron)
AMD Opteron 23xx (Gen 3 Class Opteron)
AMD Opteron 62xx class CPU
Intel CPU version
```

Many of these CPU strings look legitimate, but are the exact strings used by KVM and QEMU.

```
# kvm -cpu ?
x86           qemu64  QEMU Virtual CPU version 2.4.0
x86           phenom  AMD Phenom(tm) 9550 Quad-Core Processor
x86         core2duo  Intel(R) Core(TM)2 Duo CPU     T7700  @ 2.40GHz
x86            kvm64  Common KVM processor
x86           qemu32  QEMU Virtual CPU version 2.4.0
x86            kvm32  Common 32-bit KVM processor
x86          coreduo  Genuine Intel(R) CPU           T2600  @ 2.16GHz
x86              486
x86          pentium
x86         pentium2
x86         pentium3
x86           athlon  QEMU Virtual CPU version 2.4.0
x86             n270  Intel(R) Atom(TM) CPU N270   @ 1.60GHz
x86           Conroe  Intel Celeron_4x0 (Conroe/Merom Class Core 2)
x86           Penryn  Intel Core 2 Duo P9xxx (Penryn Class Core 2)
x86          Nehalem  Intel Core i7 9xx (Nehalem Class Core i7)
x86         Westmere  Westmere E56xx/L56xx/X56xx (Nehalem-C)
x86      SandyBridge  Intel Xeon E312xx (Sandy Bridge)
x86        IvyBridge  Intel Xeon E3-12xx v2 (Ivy Bridge)
x86     Haswell-noTSX  Intel Core Processor (Haswell, no TSX)
x86          Haswell  Intel Core Processor (Haswell)
x86  Broadwell-noTSX  Intel Core Processor (Broadwell, no TSX)
x86        Broadwell  Intel Core Processor (Broadwell)
x86       Opteron_G1  AMD Opteron 240 (Gen 1 Class Opteron)
x86       Opteron_G2  AMD Opteron 22xx (Gen 2 Class Opteron)
x86       Opteron_G3  AMD Opteron 23xx (Gen 3 Class Opteron)
x86       Opteron_G4  AMD Opteron 62xx class CPU
x86       Opteron_G5  AMD Opteron 63xx class CPU
x86             host  KVM processor with all supported host features (only available in KVM mode)
```

If the check passes, the string is stored in the global struct for later testing.

Furthermore, the CPUID instruction can be executed with the register EAX set to 0x40000000. This will return a string that can be set by a hypervisor.

Blacklist for CPUID where EAX=0x40000000:

```
VMwareVMware
XenVMMXenVMM
KVMKVMKVM
prl hyperv
Microsoft Hv
```

More about the CPUID can be found in the Intel Instruction Set Reference starting on page 3-179: http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-instruction-set-reference-manual-325383.pdf

## Hostname Check

The sample contains a blacklist of hostnames. In the event the result of `GetComputerNameW()` is found in the blacklist, again, the process terminates.

```
brbrb-d8fb22af1
jonathan-c561e0
avreview1-VMXP
vwinxp-maltest
avreview-VMSunbox
infected-system
```

Googling these strings brings results that suggest that they are hostnames for sandboxes and honeypots. These hostnames are also used in other malware samples as hostname blacklists.

## Filename Check

By a call to `GetModuleFileNameW()` the sample check its filename to see if it is in a location commonly used by sandboxes:
Full string case insensitive compare:

```
C:\xxx\sample.exe
C:\sample.exe
C:\Shared\dum._vxe
C:\SniferFiles\sample.exe
C:\virus\virus.exe
C:\virus.exe
c:\sampel.exe
C:\setup.exe
C:\runme.exe
c:\VMRun\Zample.exe
c:\FILE.EXE
C:\run\temp.exe
c:\taskrun\samples\rtktst.exe.exe
c:\artifact.exe
C:\manual\sunbox.exe
C:\1.exe
```

String find:

```
malware.exe
\virus\
admin\downloads\samp1e_
sample_execution
mlwr_smpl.exe
```

Any file in this format where ‘x’ is any character.

```
'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx\\xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx'
```

This format apears to be a GUID string. There must be some sandboxing technology that uses this format that the author was aware of.
Finally, it checks if a ‘Z:\’ drive is present, then checks for the file ‘Z:\VxStream’. This will detect if it is being run in the VxStream Sandbox

## Look for DLLs associated with function hooking

User-space hooking is a technique used by anti-virus to detect what could be considered malicious behavior. The technique is also used by sandboxes to record a log of runtime behaviors of a process. The most common way of hooking a process is to inject a DLL into the process. This hooking DLL will patch system DLLs like kernel32.dll and ntdll.dll in memory. When the process being hooked makes a call into these system DLLs, it will be redirected to a “detour” function inside of the injected hooking dll. If the function call is determined to be benign, the control flow is allowed to continue into the system DLLs.

Anti-Virus products that utilize this technique tend to prefer hooking system DLLs like `kernel32.dll` over `ntdll.dll`. This is because hooking `ntdll.dll` is less reliable and requires more labor to write. The interface to `ntdll.dll` could change on the whim of Microsoft, and isn’t documented well. `kernel32.dll` has a more predictable and constant interface and is well documented on MSDN.

A malicious program wanting to avoid detection at runtime by a user-space hook might have some success calling directly into `ntdll.dll` instead of `kernel32.dll` because the underlying `ntdll.dll` functions may not be hooked.

The sample will look for injected DLLs associated with user-space hooks in its process space by making a call to `ntdll!LdrGetDllHandle()` instead of the more common `kernel32!GetModuleHandle()`. By calling directly into the ntdll implementation, hooks on the kernel32 layer can be avoided. If a DLL associated with hooking is discovered, the programs behavoir can be altered to specifically avoid detection by these products.

If a DLL is found, the result is stored so that future malicious functionality can be suppressed, or specific techniques to avoid detection can be utilized.

The hooking DLL black-list:

| DLL File Name | Vendor |
|---|---|
| avcuf32.dll | BitDefender |
| BgAgent.dll | BullGuard |
| guard32.dll | COMODO |
| wl_hook.dll | Agnitum |
| QOEHook.dll | Qurb |
| a2hooks32.dll | Emsisoft |

## Looking for Sandbox Artifacts on the File System

If any of this files or directories are found, the process terminates prematurely. These files appear to be associated with sandbox software.

```
C:\agent\agent.pyw
C:\sandbox\starter.exe
c:\ipf\BDCore_U.dll
C:\cwsandbox_manager
C:\cwsandbox
C:\Stuff\odbg110
C:\gfisandbox
C:\Virus Analysis
C:\iDEFENSE\SysAnalyzer
c:\gnu\bin
C:\SandCastle\tools
C:\cuckoo\dll
C:\MDS\WinDump.exe
C:\tsl\Raptorclient.exe
C:\guest_tools\start.bat
C:\tools\aswsnx\snxcmd.exe
C:\Winap\ckmon.pyw
c:\tools\decodezeus
c:\tools\aswsnx
C:\sandbox\starter.exe
C:\Kit\procexp.exe
c:\tracer\mdare32_0.sys
C:\tool\malmon
C:\Samples\102114\Completed
c:\vmremote\VmRemoteGuest.exe
d:\sandbox_svc.exe
```

## Checking Number of CPU Cores

Remember how the CPUID "Product Brand String" was stored for later use? Here's why. If the CPU *should* have more than 1 core, but the operating system only reports 1 core, it's likely running inside a virtual machine.

A call is made to `RtlGetNativeSystemInformation(SystemBasicInformation, ...)`. This call will fill the contents of a `struct SYSTEM\_BASIC\_INFORMATION` struct. The sample checks the field `\_SYSTEM\_BASIC\_INFORMATION.NumberOfProcessors` and if the value is 1 and the CPU Product Brand String reported *should* have more than one core, the process is terminated.

CPU brand strings that are checked:

```
'Intel(R) Core(TM) i7'
'Intel(R) Core(TM) i5'
'Intel(R) Core(TM) i3'
'Intel(R) Core(TM)2 Duo CPU'
```

NtQuerySystemInformation() _SYSTEM_INFORMATION_CLASS enum in ReactOS source `RtlGetNativeSystemInformation()` seems to be similar to `NtQuerySystemInformation()` documented on MSDN here. I found an unnoficial source of the `struct SYSTEM_BASIC_INFORMATION` here: http://masm32.com/board/index.php?topic=3400.0 ReactOS struct _SYSTEM_BASIC_INFORMATION

## Yet Another DLL hooking blacklist

These DLLs are associated with software used to manually analize samples.

```
dir_watch.dll
tracer.dll
SbieDll.dll
APIOverride.dll
NtHookEngine.dll
api_log.dll
LOG_API.DLL
LOG_API32.DLL
```

`ntdll!LdrGetDllHandle()` If any of these DLLs are loaded, the process terminates.

## Kernel Driver Check

ReactOS RtlGetNativeSystemInformation is just NtQuerySystemInformation ReactOS SystemModuleInformation A call to `ntdll!RtlGetNativeSystemInformation(SystemModuleInformation, ...)` is made. This returns a list of all loaded kernel drivers.

Each kernel module is compared to a blacklist that is organized by vendor.

If any of these drivers are found, the process will terminate.

- ???
  - taskrun\bruta\kbruta.sys
  - taskrun\bruta\TBM.sys
- Sandbox, VM, and SysInternals drivers
  - vmx_svga.sys
  - vmmouse.sys
  - xennet.sys
  - CaptureProcessMonitor.sys

- CaptureRegistryMonitor.sys
- CaptureFileMonitor.sys
- CWSandboxWatchdogDri (sic)
- VBoxVideo.sys

If any of the following drivers are found, it is noted in the global struct, for later evasion techniques.

- Quick Heal (Indian)
  - bdsnm.sys
  - bdsflt.sys
  - ggc.sys
  - catflt.sys
  - wsnf.sys
  - llio.sys
  - mscank.sys
  - EMLTDI.SYS
- ZoneAlarm
  - vsdatant.sys
- Qihoo 360 (Chinese)
  - 360Box.sys
  - 360Box64.sys
  - 360Camera.sys
  - 360Camera64.sys
  - 360SelfProtection.sys
  - 360AntiHacker.sys
  - 360AntiHacker64.sys
  - 360AvFlt.sys
- PC Tools (now part of Norton Security)
  - pctNdis.sys
  - pctNdisLW64.sys
- Norton 360
  - 360AvFlt.sys
  - 360FsFlt.sys
- K7 Computing (Indian)
  - K7Sentry.sys
  - K7FWFilt.sys
  - K7TdiHlp.sys
- Trust Port (Czech Republic)
  - tpsec.sys
- Privacyware (US)
  - pwipf6.sys
- MicroWorld escan (US, India)
  - mwfsmflt.sys
  - ProcObsrvesx.sys
  - bdfsfltr.sys
  - econceal.sys
- Filseclab (Chinese)
  - ffsmon.sys
  - fildds.sys
  - filmfd.sys
  - filppd.sys
- Kaspersky
  - kl1.sys
  - klif.sys
  - kltdi.sys
  - kneps.sys
  - klkbdflt.sys
  - klmouflt.sys
- G Data (German)
  - GDBehave.sys
  - GDNdisIc.sys
  - gdwfpcd64.sys
  - gdwfpcd32.sys
- Arcabit (Polish)
  - ABFLT.sys
- Avast (Czech Republic)
  - aswMonFlt.sys
  - aswRvrt.sys
  - aswRdr2.sys

- aswVmm.sys
    - aswNdisFlt.sys
    - aswSnx.sys
    - aswSP.sys
    - aswStm.sys
- Avira (German)
    - avnetflt.sys
    - avkmgr.sys
    - avipbb.sys
    - avgntflt.sys
- ESET (Slovakia)
    - EpfwLWF.sys
    - epfwwfp.sys
    - eamonm.sys
    - ehdrv.sys
    - epfw.sys
    - eelam.sys
- Baidu (Chinese)
    - Bfilter.sys
    - Bfmon.sys
    - Bhbase.sys
- AVG (Czech Republic)
    - avgdiskx.sys
    - avgidsdriverlx.sys
    - avgtdix.sys
    - avgunivx.sys

## Anti-Process

This will only execute if something was found in the kernel module check. (???)

The process list is enumerated by calling RtlGetNativeSystemInformation(5)

ReactOS 5 == proc info MSDN SYSTEM_PROCESS_INFORMATION If a process with this filename is found, its process id is recorded, and later terminated.

This is where the author prepares an attack on the analyst's psyche. So far, the process only detects sandbox, VM, and antivirus, but this list of tools are usually run manually by an analyst. By detecting their presence but not immediately terminating them, instead delaying the termination until a later part of the process, can give some analyst nightmares.

- apispy.exe
- autoruns.exe
- autorunsc.exe
- dumpcap.exe
- emul.exe
- fortitracer.exe
- hookanaapp.exe
- hookexplorer.exe
- idag.exe
- idaq.exe
- importrec.exe
- imul.exe
- joeboxcontrol.exe
- joeboxserver.exe
- multi_pot.exe
- ollydbg.exe
- peid.exe
- petools.exe
- proc_analyzer.exe
- procexp.exe
- procmon.exe
- regmon.exe
- scktool.exe
- sniff_hit.exe
- sysanalyzer.exe
- vmsrvc.exe
- vmtoolsd.exe

- vmusrvc.exe
- vmwaretray.exe

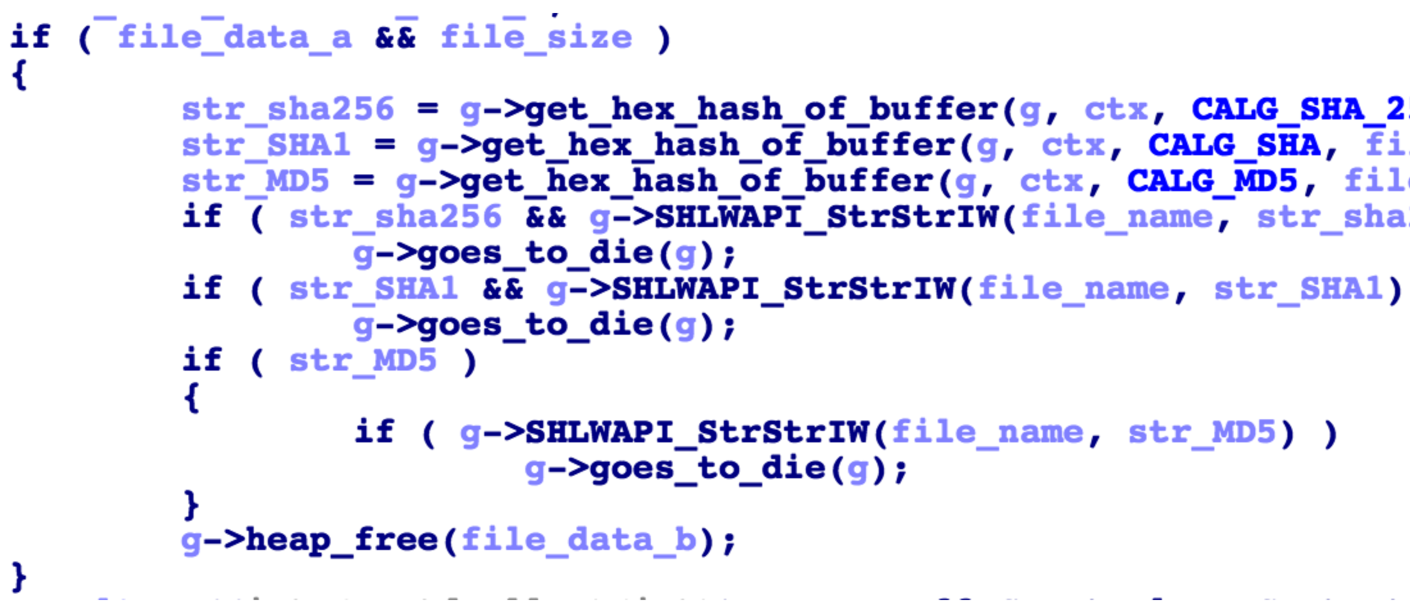
For each process, a call to `QueryFullProcessImageNameW()` is made and compared against a second blacklist. If any string in the blacklist occurs in the full image name, the process terminates.

THESE AREN'T SANDBOXES OR A/V, ARE THEY????
- \oracle\product\
- \OraHome_1\perl\
- \dbhome_1\perl\bin
- \ZKTeco\ZKAccess\
- \oracle\FRHome_1\perl\
- \Oracle\Middleware\

## Is File Name [hash].exe Check

In the Anti-Virus industry, it is extreamly common to rename a sample's filename to its hash value. This is done to easily identify a sample, and to store it with a unique filename. Typical hash algorithms that are used are MD5, SHA-1, and SHA-256.

This sample will read itself off of the disk, then calculate its checksum. Then it will see if the hex-string of the checksum is found in its filename.



```
if ( file_data_a && file_size )
{
        str_sha256 = g->get_hex_hash_of_buffer(g, ctx, CALG_SHA_2
        str_SHA1 = g->get_hex_hash_of_buffer(g, ctx, CALG_SHA, fi
        str_MD5 = g->get_hex_hash_of_buffer(g, ctx, CALG_MD5, fil
        if ( str_sha256 && g->SHLWAPI_StrStrIW(file_name, str_sha
                g->goes_to_die(g);
        if ( str_SHA1 && g->SHLWAPI_StrStrIW(file_name, str_SHA1)
                g->goes_to_die(g);
        if ( str_MD5 )
        {
                if ( g->SHLWAPI_StrStrIW(file_name, str_MD5) )
                        g->goes_to_die(g);
        }
        g->heap_free(file_data_b);
}
```

## Is VMware Tools Installed Check

Next, the sample checks if these two directories exist. If either exist, the process is terminated.
- C:\Program Files\VMware\VMware Tools
- C:\Program Files (x86)\VMware\VMware Tools

## Hard Disk Vendor Check

The children of these two registry keys are enumerated:
- \Registry\Machine\SYSTEM\CurrentControlSet\Enum\IDE
- \Registry\Machine\SYSTEM\CurrentControlSet\Enum\SCSI

The values are check against a blacklist containing virtualized hard disk vendors.
- QEMU_
- VMware
- Ven_Red_Hat&Prod_VirtIO
- DiskVBOX
- DiskVirtual

If a value is found in the blacklist, the process is terminated.

## Misc Hardware Vendors and BIOS Checks

These hardware specific registry keys are check. If they exist, the process terminates.
- \Registry\Machine\HARDWARE\ACPI\DSDT\VBOX__\VBOXBIOS
- \Registry\Machine\SYSTEM\CurrentControlSet\Enum\ACPI\Hyper_V_Gen_Counter_V1
- \Registry\Machine\SYSTEM\CurrentControlSet\Enum\ACPI\XEN0000

- \Registry\Machine\SYSTEM\CurrentControlSet\Enum\XENBUS\CLASS_VBD&REV_02

In the same function, `\Registry\Machine\HARDWARE\DESCRIPTION\System\` is queried and checked against this blacklist:

- SystemBiosVersion
  - 'BOCHS – 1'
  - 'VBOX – 1'
  - 'PRLS – 1'
- VideoBiosVersion
  - 'VirtualBox'

If any of these registry keys match, the process is terminated.

## Anti Network Interface Card (NIC) Check

This check is skipped if kernel modules associated with "Privacyware" where detected. I'm assuming "Privacyware" detects the API calls in this check as malicious.

This function check the NICs that are installed, and calls to `IPHLPAPI!GetAdaptersInfo()`

Meeting these condition will cause a premature termination.

- Realtek RTL8139 Family PCI Fast Ethernet NIC
  - also username is 'antonie'
  - also 'c:\downloads\' exists
- Realtek RTL8139C+ Fast Ethernet NIC
  - also username is 'Antony'
  - also 'c:\downloads\' exists

These network cards will get terminated out-right:

- VMware Accelerated AMD PCNet Adapter
- Microsoft Virtual Machine Bus Network Adapter
- Microsoft Hyper-V Network Adapter
- Adaptador de red de bus de m?quina virtual de Microsoft

If the network card is NOT one of these, it will check the MAC address:

- VMware Virtual Ethernet Adapter for VMnet8
- VMware Virtual Ethernet Adapter for VMnet1
- VirtualBox Host-Only Ethernet Adapter

These MAC addresses will result in a premature termination:

| MAC address | OUI Information | Notes |
|---|---|---|
| `00:01:02:03:04:xx` | 3COM | defunt, obvious bogus mac address |
| `00:03:FF:xx:xx:xx` | Microsoft Corporation | doesn't make physical hardware? |
| `00:0C:29:xx:xx:xx` | VMware, Inc. | |
| `08:00:27:xx:xx:xx` | ~~Cadmus Computer Systems~~ | VirtualBox |
| `00:07:e9:e4:ce:4d` | Intel | 0 results on google |
| `00:30:18:ab:d7:f2` | Jetway Information Co., Ltd. | 0 results on google |
| `00:ff:f2:f8:30:xx` | Dell?? VirtualBox?? | |
| `00:50:56:xx:xx:xx` | VMware, Inc. | |
| `52:54:00:12:34:56` | Realtek | copypasta QEMU startup script? |
| `00:1c:42:xx:xx:xx` | Parallels, Inc. | VMware product |
| `00:15:5d:xx:xx:xx` | Microsoft Corporation | |
| `00:1d:d8:xx:xx:xx` | Microsoft Corporation | |

I would like to know who owns the "00:07:e9:e4:ce:4d" and "00:30:18:ab:d7:f2". If they are burnt onto a physical device, it's either a development machine, or a targeted machine to be specifically avoided.

## Window Title Check

Pairs of window class names and titles are check for, and if one is found, the samples's process is terminated prematurely. These tools are used by analysts and some are used by sandboxes.

| Window Class | Window Title |
|---|---|
| PROCEXPL | sysinternals |
| PROCMON_WINDOW_CLASS | sysinternals |
| Autoruns | sysinternals |
| TCPViewClass | sysinternals |
| 0 | TCPView – Sysinternals: www.sysinternals.com |
| 0 | File Monitor – Sysinternals: www.sysinternals.com |
| 0 | Registry Monitor – Sysinternals: www.sysinternals.com |
| 0 | Process Monitor – Sysinternals: www.sysinternals.com |

| Window Class | Window Title |
|---|---|
| gdkWindowToplevel | Wireshark |
| API_TRACE_MAIN | 0 |
| 0 | Wget [100%%] http://tristan.ssdcorp.net/guid |
| 0 | C:\Program Files\Wireshark\dumpcap.exe |
| 0 | C:\wireshark\dumpcap.exe |
| 0 | C:\SandCastle\tools\FakeServer.exe |
| 0 | C:\Python27\python.exe |
| 0 | start.bat - C:\Manual\auto.bat |
| 0 | Fortinet Sunbox |
| 0 | PEiD v0.95 |
| 0 | Total Commander 7.0 - Ahnlab Inc. |
| 0 | Total Commander 6.53 - GRISOFT, s.r.o. |
| 0 | Total Commander 7.56a - Avira Soft |
| 0 | Total Commander 7.56a - ROKURA SRL |
| 0 | C:\strawberry\perl\bin\perl.exe |
| ThunderRT6FormDC | SysAnalyzer |
| TfrmMain | All-Seeing Eye |
| Afx:400000:b:10011:6:350167 | Malicious Code Monitor v1.7.6 For NT(x86) - (ariesike@naver.com) |
| TApplication | Mouse Move - by RJL Software, Inc. |
| SmartSniff | SmartSniff |
| ConsoleWindowClass | VxStream Kernel Service Manager |

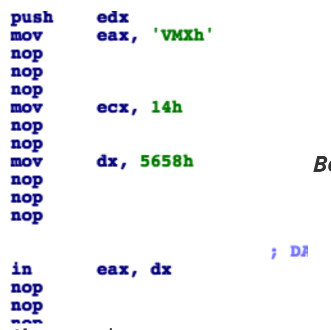
| Registry Key | Notes |
|---|---|
| \Registry\Machine\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\Iris Network Traffic Analyzer | |
| \Registry\Machine\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\InstallWatch Pro 2.5 | |
| \Registry\Machine\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\SysAnalyzer_is1 | |
| \Registry\Machine\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall{13BE68B1-7498-48AB-9D22-AD3AB6532531} | API Monitor v2 Alpha |
| \Registry\Machine\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\Oracle VM VirtualBox Guest Additions | |
| \Registry\Machine\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\Oracle VM VirtualBox Guest Additions | |
| \Registry\Machine\SYSTEM\CurrentControlSet\Enum\PCI\VEN_80EE&DEV_BEEF&SUBSYS_00000000&REV_00 | VirtualBox Graphics Device Drivers |
| \Registry\Machine\SYSTEM\CurrentControlSet\Enum\PCI\VEN_80EE&DEV_CAFE&SUBSYS_00000000&REV_00 | VirtualBox Guest Service Device Drivers |
| \Registry\Machine\SYSTEM\CurrentControlSet\Enum\PCI\VEN_5333&DEV_8811&SUBSYS_00000000&REV_00 | S3 Video Card (used by virtual machines) |
| \Registry\Machine\SYSTEM\CurrentControlSet\Enum\PCI\VEN_1AB8&DEV_4005&SUBSYS_04001AB8&REV_00 | Parallels Display WDDM Device Drivers |
| \Registry\Machine\SYSTEM\CurrentControlSet\Enum\PCI\VEN_1AB8&DEV_4000&SUBSYS_04001AB8&REV_00 | Parallels Tool Device Drivers |
| \Registry\Machine\SYSTEM\CurrentControlSet\Enum\PCI\VEN_1AB8&DEV_4006&SUBSYS_04061AB8&REV_00 | Parallels Memory Controller |
| \Registry\Machine\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall{25AD16E5-F48B-4455-83D7-849D600475A4} | Winalysis WindowexeAllkiller ? |
| \Registry\Machine\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\Iris Network Traffic Analyzer | |
| \Registry\Machine\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\SysAnalyzer_is1 | |
| \Registry\Machine\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\InstallWatch Pro 2.5 | |
| \Registry\Machine\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall{13BE68B1-7498-48AB-9D22-AD3AB6532531} | API Monitor v2 Alpha |
| \Registry\Machine\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall{25AD16E5-F48B-4455-83D7-849D600475A4} | Winalysis WindowexeAllkiller ? |

## VMXh VMWare Check

The sample checks for the presence of VMware, by trying to execute a Intel `IN` instruction with EAX set to the value 'VMXh'. The `IN` instruction is a privileged instruction, and outside of the VMware guest, would result in a General Protection Fault exception. But inside a VMWare guest, the fault is not generated. VMWare uses this as a way for its guest software to communicate to the host. It is documented here.

The specific technique to detect VMWare is here.

This is how the sample implements the check:



```
push    edx
mov     eax, 'VMXh'
nop
nop
nop
mov     ecx, 14h
nop
nop
mov     dx, 5658h
nop
nop
nop

                          ; DA
in      eax, dx
nop
nop
```

Because this portion must be written in assembly, I assume the NOPs are to thwart any static detection on the opcodes.

## Direct3D Video Card Check

Using the Direct3D interface, the sample is able to enumerate information about the installed video adapters. If the video cards are vendors are in the blacklist, the process prematurely terminates. Vendor Blacklist:

| VendorID | Vendor Name | Notes |
|---|---|---|
| 0x15ad | VMWare Inc. | |
| 0x80ee | Oracle Corp. | Virtual Box |
| 0x1013 | Cirrus Logic | Bochs and QEMU |

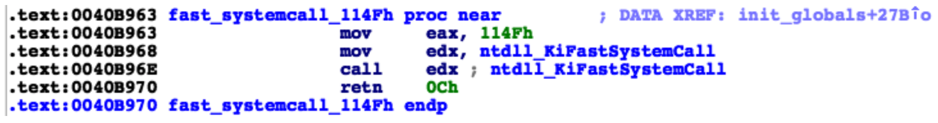
Vendor Whitelist:

| VendorID | Vendor Name |
|---|---|
| 0x8086 | Intel |
| 0x10de | Nvidia |
| 0x1002 | AMD |

Now, because there is a whitelist and a blacklist, there exists a possiblity that the device was not in either list. In the event of this case, the tie is settled by calling `GetCursorPosition()`. If there are 15 mouse movements, then the sample assumes that it is not in a sandbox, and contiues execution. If there isn't 15 mouse movements, the sample just blocks, waiting for the movements to occur. During this waiting period, the .data seciton is re-encrypted. If the sample where in a sandbox, it would be stuck in this loop, waiting. When the sandbox times-out, it might dump the contents of memory. By re-encrypteing the .data section, it will be encrypted in the data dump.

A common feature of sandboxes is a "mouse mover." The mouse mouse mover will make an application think there is a user on the workspace by moving the mouse cursor around.

The sample also utilizes direct system calls to make the call to `GetCursorPosition()`. This technique will bypass any user-space hooks that might try to move the mouse automatically.



```
.text:0040B963 fast_systemcall_114Fh proc near          ; DATA XREF: init_globals+27B↑o
.text:0040B963                 mov     eax, 114Fh
.text:0040B968                 mov     edx, ntdll_KiFastSystemCall
.text:0040B96E                 call    edx ; ntdll_KiFastSystemCall
.text:0040B970                 retn    0Ch
.text:0040B970 fast_systemcall_114Fh endp
```

# Native Application Binary

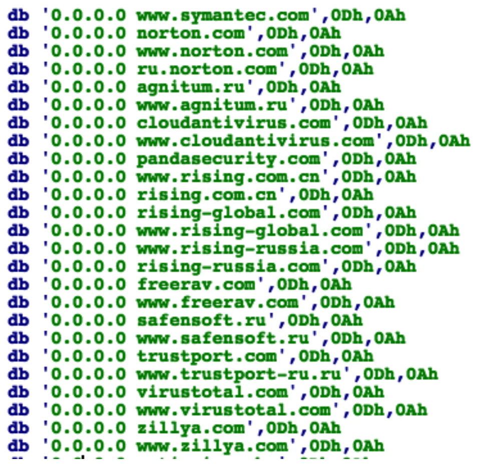
The beginning of this binary has many anti-VM and anti-sandboxing techniques. just like the previous binary. It also allocates a large struct containing function pointers like the previous binary.

This portion of the sample is encrypted and compressed in the .data segment. As mentioned before, it written to disk in a NTFS ADS and at boot time before all the windows subsystems are loaded. If you have ever upgraded to windows 2000, you will remember that the installation could upgrade the filesystem from FAT32 to NTFS. This portion of the sample runs at the same point as the file system upgrade code would run.

An example project using this technique can be found on codeproject.com

The Native binary is written in the same style as the parent. A large struct is allocated on the heap that stores function pointers. It also uses RC4 to encrypt its string literals, and contains the final paylad compressed using aplib.

The goal of this portion of the sample is to remove a large number of anti-virus software. It will also modify the `host` used by the DNS client. There are many records for anti-virus update servers that get set to `0.0.0.0`, effectivly stoping any view anti-virus installed from being able to update its definitions.

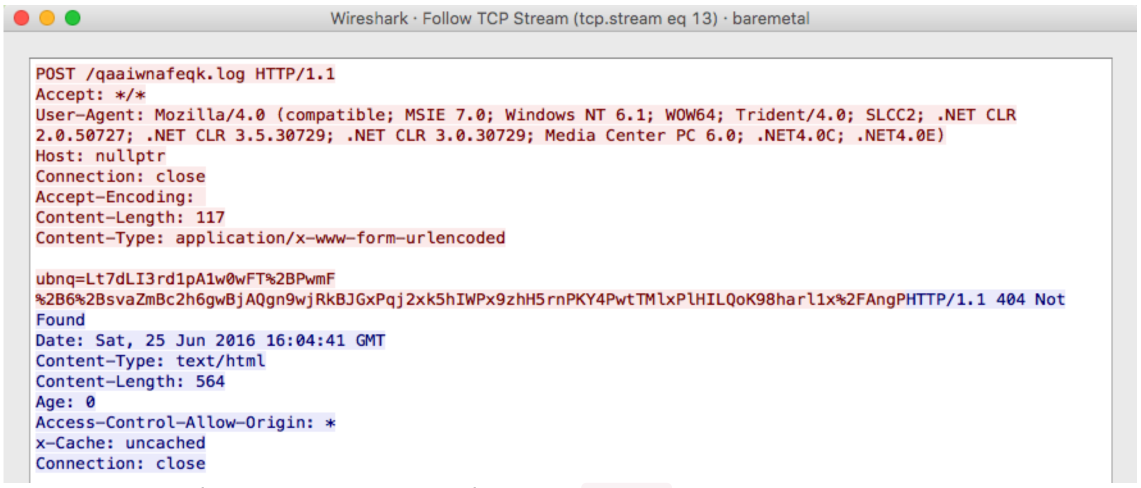

```
db '0.0.0.0 www.symantec.com',0Dh,0Ah
db '0.0.0.0 norton.com',0Dh,0Ah
db '0.0.0.0 www.norton.com',0Dh,0Ah
db '0.0.0.0 ru.norton.com',0Dh,0Ah
db '0.0.0.0 agnitum.ru',0Dh,0Ah
db '0.0.0.0 www.agnitum.ru',0Dh,0Ah
db '0.0.0.0 cloudantivirus.com',0Dh,0Ah
db '0.0.0.0 www.cloudantivirus.com',0Dh,0Ah
db '0.0.0.0 pandasecurity.com',0Dh,0Ah
db '0.0.0.0 www.rising.com.cn',0Dh,0Ah
db '0.0.0.0 rising.com.cn',0Dh,0Ah
db '0.0.0.0 rising-global.com',0Dh,0Ah
db '0.0.0.0 www.rising-global.com',0Dh,0Ah
db '0.0.0.0 www.rising-russia.com',0Dh,0Ah
db '0.0.0.0 rising-russia.com',0Dh,0Ah
db '0.0.0.0 freerav.com',0Dh,0Ah
db '0.0.0.0 www.freerav.com',0Dh,0Ah
db '0.0.0.0 safensoft.ru',0Dh,0Ah
db '0.0.0.0 www.safensoft.ru',0Dh,0Ah
db '0.0.0.0 trustport.com',0Dh,0Ah
db '0.0.0.0 www.trustport-ru.ru',0Dh,0Ah
db '0.0.0.0 virustotal.com',0Dh,0Ah
db '0.0.0.0 www.virustotal.com',0Dh,0Ah
db '0.0.0.0 zillya.com',0Dh,0Ah
db '0.0.0.0 www.zillya.com',0Dh,0Ah
```

It finishes by dropping the final payload to `%SystemRoot%\rdpinst.exe` and ensuring that it runs later in boot-up by setting a registry value in `\Registry\Machine\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce`.

## Final Payload

The payload shares some simliarities with the other binaries, but unlike the past two, it doesn't allocate a large struct and fill it with function pointers.

The final payload collects recon from the infected machine and reports back to its C2 server over HTTP.


Wireshark · Follow TCP Stream (tcp.stream eq 13) · baremetal

```
POST /qaaiwnafeqk.log HTTP/1.1
Accept: */*
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR
2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)
Host: nullptr
Connection: close
Accept-Encoding:
Content-Length: 117
Content-Type: application/x-www-form-urlencoded

ubnq=Lt7dLI3rd1pA1w0wFT%2BPwmF
%2B6%2BsvaZmBc2h6gwBjAQgn9wjRkBJGxPqj2xk5hIWPx9zhH5rnPKY4PwtTMlxPlHILQoK98harl1x%2FAngPHTTP/1.1 404 Not
Found
Date: Sat, 25 Jun 2016 16:04:41 GMT
Content-Type: text/html
Content-Length: 564
Age: 0
Access-Control-Allow-Origin: *
x-Cache: uncached
Connection: close
```

One unique feature of all the traffic collected is that the HTTP host field is always `nullptr`.

## Sample Information

Sha-256: 766e49811c0bb7cce217e72e73a6aa866c15de0ba11d7dda3bd7e9ec33ed6963
* `638d549a24bb0a28e462c70880bf3f979f137cc6`: Main Sample
  * `ce0633d8be65202870e7b916e7bec5a0218cbbbb`: Packed Native API Application binary
    * 14598af84ee2dbd88d3fff0b60aba829a412dfbe3`: Packed rdpinst.exe (payload)
  * `643b295ee6985251d771b7962f2b2fc69e36f5c2`: Packed UAC bypass dll
  * `c803eb5e8a4a4e31e8168557d82ff54d68f3832d`: Packed 64-bit CVE-2014-4113 exploit

If you work within the energy sector and would like to learn more about this specific attack and how you can prevent, contact us and we'd be happy to advise on a one-to-one basis.