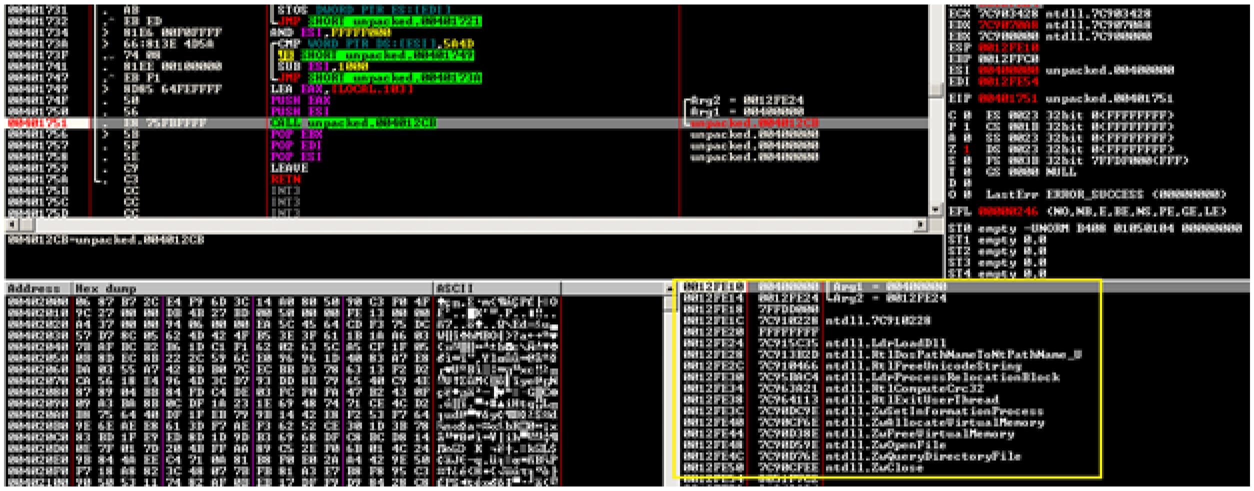
# Andromeda Bot Analysis part 2

resources.infosecinstitute.com/andromeda-bot-analysis-part-two/

Malware analysis
September 28, 2015 by **Ayoub Faouzi**

## Bot Analysis:

Now, you get the original Andromeda build file. Load the unpacked sample at OllyDBG. As before, after the stack frame at the EP, you see that the malware is looking to load API's address using the PEB_LDR_DATA structure, but this time instead of kernel32.dll; the malware try to find ntdll.dll base address, then, it will parse the EAT, hash each APIs then make comparison to find the needed APIs :



```
00401731   .  AB             STOS DWORD PTR ES:[EDI]
00401732   .^ EB ED          JMP SHORT unpacked.00401721
00401734   >  81E6 00F0FFFF  AND ESI,FFFFF000
0040173A   >  66:813E 4D5A   CMP WORD PTR DS:[ESI],5A4D
0040173F   .. 74 08          JE SHORT unpacked.00401749
00401741   .  81EE 00100000  SUB ESI,1000
00401747   .^ EB F1          JMP SHORT unpacked.0040173A
00401749   >  8D85 64FEFFFF  LEA EAX,[LOCAL.103]
0040174F   .  50             PUSH EAX                     ┌Arg2 = 0012FE24
00401750   .  56             PUSH ESI                     │Arg1 = 00400000
00401751   .  E8 75FBFFFF    CALL unpacked.004012CB       └unpacked.004012CB
00401756   >  5B             POP EBX                       unpacked.00400000
00401757   .  5F             POP EDI                       unpacked.00400000
00401758   .  5E             POP ESI                       unpacked.00400000
00401759   .  C9             LEAVE
0040175A   .  C3             RETN
0040175B      CC             INT3
0040175C      CC             INT3
0040175D      CC             INT3
004012CB=unpacked.004012CB
```

```
0012FE10   00400000  ┌Arg1 = 00400000
0012FE14   0012FE24  └Arg2 = 0012FE24
0012FE18   7FFD0000
0012FE1C   7C910228  ntdll.7C910228
0012FE20   FFFFFFFF
0012FE24   7C915C35  ntdll.LdrLoadDll
0012FE28   7C913B2D  ntdll.RtlDosPathNameToNtPathName_U
0012FE2C   7C910466  ntdll.RtlFreeUnicodeString
0012FE30   7C95B0C4  ntdll.LdrProcessRelocationBlock
0012FE34   7C963021  ntdll.RtlComputeCrc32
0012FE38   7C964113  ntdll.RtlExitUserThread
0012FE3C   7C90DC9E  ntdll.ZwSetInformationProcess
0012FE40   7C90CF6E  ntdll.ZwAllocateVirtualMemory
0012FE44   7C90D38E  ntdll.ZwFreeVirtualMemory
0012FE48   7C90D59E  ntdll.ZwOpenFile
0012FE4C   7C90D76E  ntdll.ZwQueryDirectoryFile
0012FE50   7C90CFEE  ntdll.ZwClose
```

After getting inside the CALL, it will calculate the hash of a buffer located at 00402028:

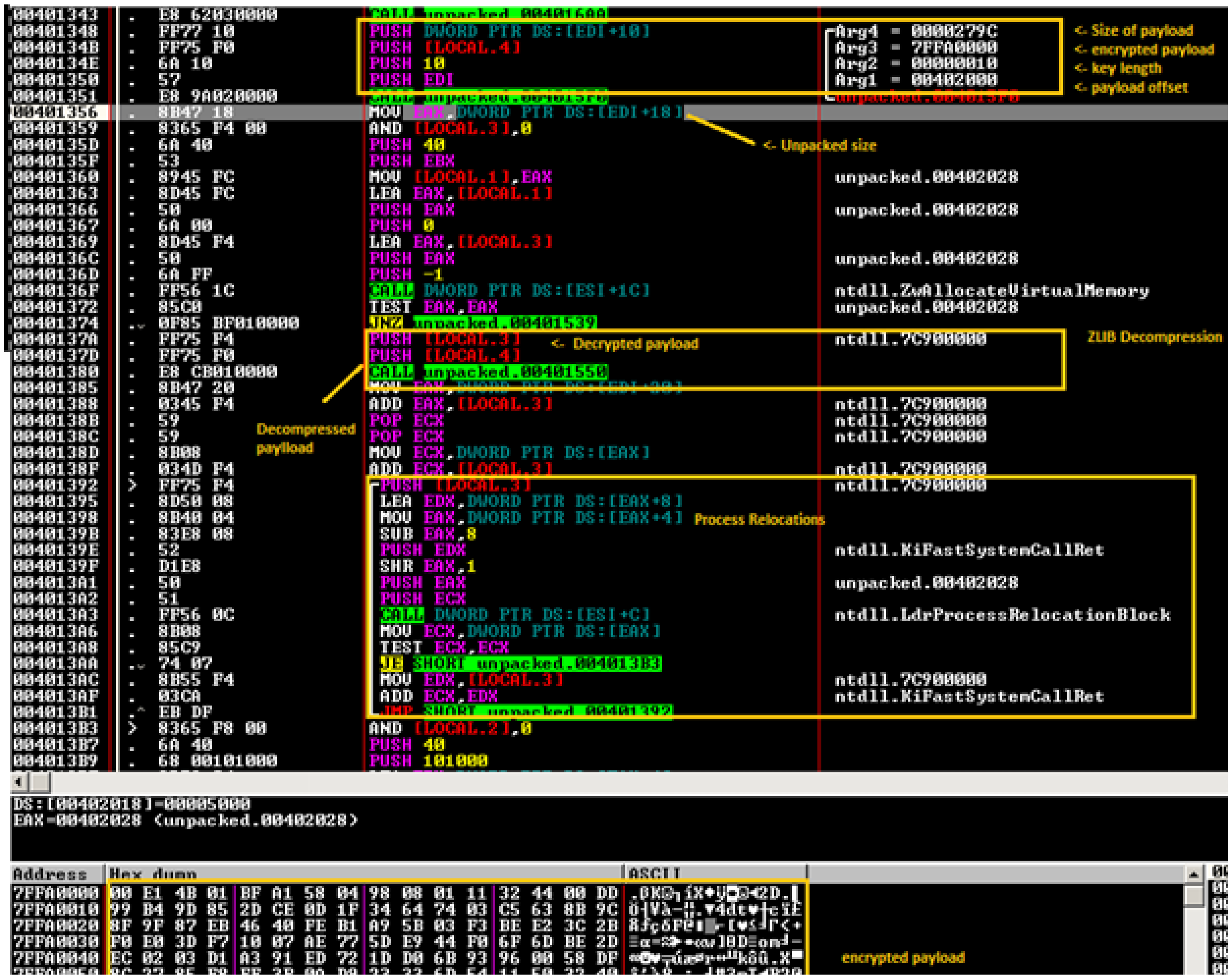
def calc_hash(string):

return binascii.crc32(string) & 0xffffffff

Then, it compared it with 0xBD274BDB, if not matches, it calls RtlExitUserThread, we will figure out soon what kind of data has been hashed. Afterwards, ZwAllocateVirtualMemory will be called and return for me 7FFA0000 then the CALL at 00401343 will copy the whole buffer to the allocated space. Next, we see the CALL at VA 00401351 which takes 4 arguments and one of them is a pointer to our unknown buffer. This routine is actually preforming a RC4 encryption, you could recognize that by looking at the constants, this is how basically some cryptographic algorithm is detected; RC4 have loops that go till 256 which is 0x100 in hexadecimal.

Using ZwAllocateVirtualMemory again, we allocate a new memory space to the partially decrypted payload so it is finally decompressed using the aPLib-library. The code that follows is responsible for processing relocations and fixing imports. For instance, from NTDLL Andromeda is importing these APIs:

LdrLoadDll, RtlDosPathNameToNtPathName_U, RtlFreeUnicodeString, LdrProcessRelocationBlock, RtlComputeCrc32, RtlWalkHeap, RtlImageNtHeader, RtlImageDirectoryEntryToData, RtlExitUserThread, ZwSetInformationProcess, ZwUnmapViewOfSection, ZwAllocateVirtualMemory, ZwMapViewOfSection, ZwFreeVirtualMemory, ZwOpenFile, ZwQueryDirectoryFile, ZwClose, ZwQueryInformationProcess.



```
00401343  .   E8 62030000     CALL unpacked.004016AA
00401348  .   FF77 10         PUSH DWORD PTR DS:[EDI+10]              ┌Arg4 = 0000279C
0040134B  .   FF75 F0         PUSH [LOCAL.4]                          │Arg3 = 7FFA0000
0040134E  .   6A 10           PUSH 10                                 │Arg2 = 00000010
00401350  .   57              PUSH EDI                                │Arg1 = 00402000
00401351  .   E8 9A020000     CALL unpacked.004015F0                  └unpacked.004015F0
00401356  .   8B47 18         MOV EAX,DWORD PTR DS:[EDI+18]
00401359  .   8365 F4 00      AND [LOCAL.3],0
0040135D  .   6A 40           PUSH 40
0040135F  .   53              PUSH EBX
00401360  .   8945 FC         MOV [LOCAL.1],EAX                       unpacked.00402028
00401363  .   8D45 FC         LEA EAX,[LOCAL.1]
00401366  .   50              PUSH EAX                                unpacked.00402028
00401367  .   6A 00           PUSH 0
00401369  .   8D45 F4         LEA EAX,[LOCAL.3]
0040136C  .   50              PUSH EAX                                unpacked.00402028
0040136D  .   6A FF           PUSH -1
0040136F  .   FF56 1C         CALL DWORD PTR DS:[ESI+1C]              ntdll.ZwAllocateVirtualMemory
00401372  .   85C0            TEST EAX,EAX                            unpacked.00402028
00401374  .v  0F85 BF010000   JNZ unpacked.00401539
0040137A  .   FF75 F4         PUSH [LOCAL.3]                          ntdll.7C900000
0040137D  .   FF75 F0         PUSH [LOCAL.4]
00401380  .   E8 CB010000     CALL unpacked.00401550
00401385  .   8B47 20         MOV EAX,DWORD PTR DS:[EDI+20]
00401388  .   0345 F4         ADD EAX,[LOCAL.3]                       ntdll.7C900000
0040138B  .   59              POP ECX                                 ntdll.7C900000
0040138C  .   59              POP ECX                                 ntdll.7C900000
0040138D  .   8B08            MOV ECX,DWORD PTR DS:[EAX]
0040138F  .   034D F4         ADD ECX,[LOCAL.3]                       ntdll.7C900000
00401392  >   FF75 F4         ┌PUSH [LOCAL.3]                         ntdll.7C900000
00401395  .   8D50 08         │LEA EDX,DWORD PTR DS:[EAX+8]
00401398  .   8B40 04         │MOV EAX,DWORD PTR DS:[EAX+4]
0040139B  .   83E8 08         │SUB EAX,8
0040139E  .   52              │PUSH EDX                               ntdll.KiFastSystemCallRet
0040139F  .   D1E8            │SHR EAX,1
004013A1  .   50              │PUSH EAX                               unpacked.00402028
004013A2  .   51              │PUSH ECX
004013A3  .   FF56 0C         │CALL DWORD PTR DS:[ESI+C]              ntdll.LdrProcessRelocationBlock
004013A6  .   8B08            │MOV ECX,DWORD PTR DS:[EAX]
004013A8  .   85C9            │TEST ECX,ECX
004013AA  .v  74 07           │JE SHORT unpacked.004013B3
004013AC  .   8B55 F4         │MOV EDX,[LOCAL.3]                      ntdll.7C900000
004013AF  .   03CA            │ADD ECX,EDX                            ntdll.KiFastSystemCallRet
004013B1  .^  EB DF           └JMP SHORT unpacked.00401392
004013B3  >   8365 F8 00      AND [LOCAL.2],0
004013B7  .   6A 40           PUSH 40
004013B9  .   68 00101000     PUSH 101000

DS:[00402018]=00005000
EAX=00402028 (unpacked.00402028)

Address   Hex dump
7FFA0000  00 E1 4B 01 BF A1 58 04 98 08 01 11 32 44 00 DD
7FFA0010  99 B4 9D 85 2D CE 0D 1F 34 64 74 03 C5 63 8B 9C
7FFA0020  8F 9F 87 EB 46 40 FE B1 A9 5B 03 F3 BE E2 3C 2B
7FFA0030  F0 E0 3D F7 10 07 AE 77 5D E9 44 F0 6F 6D BE 2D
7FFA0040  EC 02 03 D1 A3 91 ED 72 1D D0 6B 93 96 00 58 DF
```

You can find a script here of an old version of Andromeda thanks to **0xEBFE**. You still need to make some minor changes on it to get it works correctly particularly the APIs and Imports, which changed a bit:

[download]

from idaapi import *

from idautils import *

```
from aplib import decompress

import binascii

import struct

# hardcoding sucks 🙂

IMPORTS = { ‘ntdll.dll’ : (‘ZwResumeThread’, ‘ZwQueryInformationProcess’,
‘ZwMapViewOfSection’, ‘ZwCreateSection’, ‘ZwClose’, ‘ZwUnmapViewOfSection’,
‘NtQueryInformationProcess’, ‘RtlAllocateHeap’, ‘RtlExitUserThread’, ‘RtlFreeHeap’,
‘RtlRandom’,’RtlReAllocateHeap’, ‘RtlSizeHeap’, ‘ZwQuerySection’, ‘RtlWalkHeap’,
‘NtDelayExecution’),

‘kernel32.dll’ : (‘GetModuleFileNameW’, ‘GetThreadContext’, ‘GetWindowsDirectoryW’,
‘GetModuleFileNameA’, ‘CopyFileA’, ‘CreateProcessA’, ‘ExpandEnvironmentStringsA’,
‘CreateProcessW’, ‘CreateThread’, ‘CreateToolhelp32Snapshot’,
‘DeleteFileW’,’DisconnectNamedPipe’, ‘ExitProcess’, ‘ExitThread’,
‘ExpandEnvironmentStringsW’, ‘FindCloseChangeNotification’,
‘FindFirstChangeNotificationW,FlushInstructionCache’, ‘FreeLibrary’,
‘GetCurrentProcessId’, ‘GetEnvironmentVariableA’, ‘GetEnvironmentVariableW’,
‘GetExitCodeProcess’, ‘GetFileSize’, ‘GetFileTime’, ‘GetModuleHandleA’,
‘GetModuleHandleW’, ‘GetProcAddress’, ‘GetProcessHeap’, ‘CreateNamedPipeA’,
‘GetSystemDirectoryW’, ‘GetTickCount’, ‘GetVersionExA’, ‘GetVolumeInformationA’,
‘GlobalLock’, ‘GlobalSize’, ‘GlobalUnlock’, ‘LoadLibraryA’, ‘LoadLibraryW’, ‘LocalFree’,
‘MultiByteToWideChar’, ‘OpenProcess’, ‘OpenThread’, ‘QueueUserAPC’, ‘ReadFile’,
‘ResumeThread’, ‘SetCurrentDirectoryW’, ‘SetEnvironmentVariableA’,
‘SetEnvironmentVariableW’, ‘SetErrorMode’, ‘SetFileAttributesW’, ‘SetFileTime’,
‘SuspendThread’, ‘TerminateProcess’, ‘Thread32First’, ‘Thread32Next’, ‘VirtualAlloc’,
‘VirtualFree’, ‘VirtualProtect’, ‘VirtualQuery’, ‘WaitForSingleObject’, ‘WriteFile’, ‘lstrcatA’,
‘lstrcatW’, ‘lstrcmpiW’, ‘lstrcpyA’, ‘lstrcpyW’, ‘lstrlenA’, ‘lstrlenW’, ‘CreateFileW’,
‘CreateFileA’, ‘ConnectNamedPipe’, ‘CloseHandle’, ‘GetShortPathNameW’),

‘advapi32.dll’ : (‘CheckTokenMembership’, ‘RegCloseKey’, ‘ConvertStringSidToSidA’,
‘ConvertStringSecurityDescriptorToSecurityDescriptorA’, ‘RegOpenKeyExA’,
‘RegSetValueExW’, ‘RegSetValueExA’, ‘RegSetKeySecurity’, ‘RegQueryValueExW’,
‘RegQueryValueExA’, ‘RegOpenKeyExW’, ‘RegNotifyChangeKeyValue’, ‘RegFlushKey’,
‘RegEnumValueW’, ‘RegEnumValueA’, ‘RegDeleteValueW’, ‘RegDeleteValueA’,
‘RegCreateKeyExW’, ‘RegCreateKeyExA’),

‘ws2_32.dll’ : (‘connect’, ‘shutdown’, ‘WSACreateEvent’, ‘closesocket’, ‘WSAStartup’,
‘WSAEventSelect’, ‘socket’, ‘sendto’, ‘recvfrom’, ‘getsockname’, ‘gethostbyname’, ‘listen’,
‘accept’, ‘WSASocketA’, ‘bind’, ‘htons’),

‘user32.dll’ : (‘wsprintfW’, ‘wsprintfA’),
‘ole32.dll’ : (‘CoInitialize’),

‘dnsapi.dll’ : (‘DnsWriteQuestionToBuffer_W’, ‘DnsRecordListFree’,
‘DnsExtractRecordsFromMessage_W’)}

def calc_hash(string):

return binascii.crc32(string) & 0xffffffff

def rc4crypt(data, key):

x = 0

box = bytearray(range(256))

for i in range(256):

x = (x + box[i] + key[i % len(key)]) % 256

box[i], box[x] = box[x], box[i]

x,y = 0, 0

out = bytearray()

for byte in data:

x = (x + 1) % 256

y = (y + box[x]) % 256

box[x], box[y] = box[y], box[x]

out += bytearray([byte ^ box[(box[x] + box[y]) % 256]])

return out

def fix_payload_relocs_and_import(segment, relocs_offset):

current_offset = 0

# processing relocations

while True:

base = Dword(segment + relocs_offset + current_offset)

size = Dword(segment + relocs_offset + current_offset + 4)
```

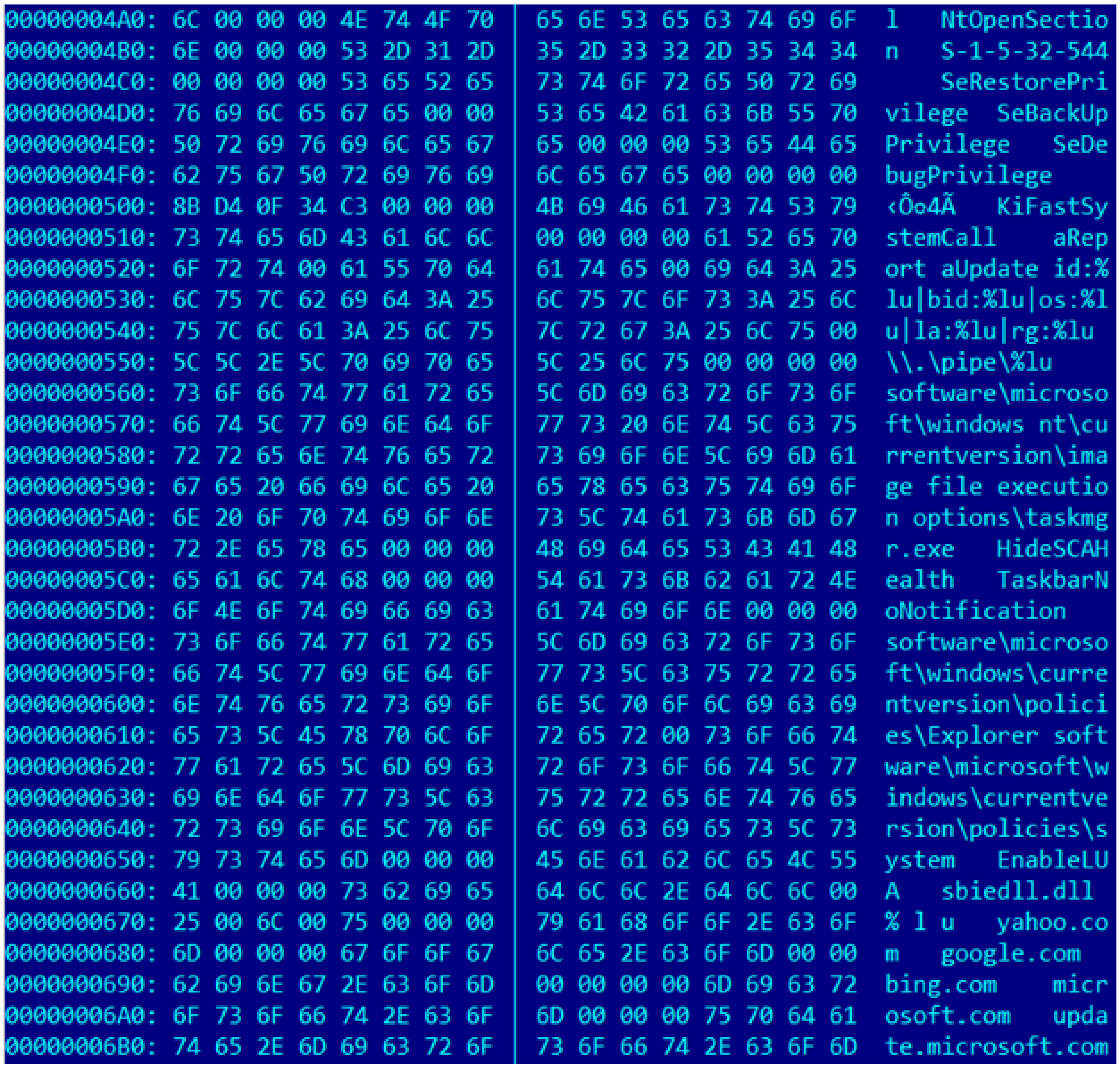
```
if (base == 0 and current_offset != 0) or size == 0:

current_offset += 4

break

current_offset += 8

size = (size – 8) // 2

for i in range(size):

reloc = Word(segment + relocs_offset + current_offset)

if reloc & 0x3000:

reloc = reloc & 0xFFF

PatchDword(segment + base + reloc, Dword(segment + base + reloc) + segment)

SetFixup(segment + base + reloc, idaapi.FIXUP_OFF32 or idaapi.FIXUP_CREATED, 0,
Dword(segment + base + reloc) + segment, 0)

current_offset += 2

# processing imports

while True:

module_hash = Dword(segment + relocs_offset + current_offset)

import_offset = Dword(segment + relocs_offset + current_offset + 4)

current_offset += 8

if module_hash == 0 or import_offset == 0:

break

module = None

for library in iter(IMPORTS):

if module_hash == calc_hash(library.lower()):

module = library

while True:
```

```
func_hash = Dword(segment + relocs_offset + current_offset)

current_offset += 4

if func_hash == 0:

break

if module is not None:

for function in iter(IMPORTS[module]):

if func_hash == calc_hash(function):

MakeDword(segment + import_offset)

MakeName(segment + import_offset, SegName(segment) + ‘_’ + module.split(‘.’)[0] + ‘_’ + function)

else:

print(‘Import not found: module = 0x{0:08X}, function = 0x{1:08X}’.format(module_hash, func_hash))

import_offset += 4

return

def decrypt_payload(encrypted_addr, rc4key, encrypted_size, unpacked_size, entry_point, relocs, relocs_size):

buffer = bytearray(encrypted_size)

for i in range(len(buffer)):

buffer[i] = Byte(encrypted_addr + i)

decrypted = rc4crypt(buffer, rc4key)

unpacked = decompress(str(decrypted)).do()

# checking for free segment address

seg_start = 0x10000000

while SegName(seg_start) != ”:

seg_start += 0x10000000
```

```
AddSeg(seg_start, seg_start + unpacked_size, 0, 1, idaapi.saRelPara, idaapi.scPub)

# copying data to new segment

data = unpacked[0]

for i in range(len(data)):

PatchByte(seg_start + i, ord(data[i]))

fix_payload_relocs_and_import(seg_start, relocs)

MakeFunction(seg_start + entry_point)

return

def main():

payload_addr = AskAddr(ScreenEA(), “Enter address of andromeda payload”)

if payload_addr != idaapi.BADADDR and payload_addr is not None:

payload = bytearray(0x28)

for i in range(len(payload)):

payload[i] = Byte(payload_addr + i)

dwords = struct.unpack_from(‘<LLLLLL’, bytes(payload), 0x10)

decrypt_payload(payload_addr + 0x28, payload[:16], dwords[0], dwords[2], dwords[3],
dwords[4], dwords[5])

if __name__ == ‘__main__’:

main()
```

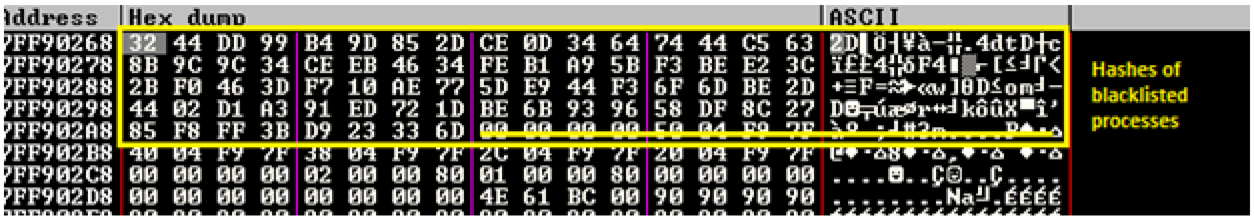
At the end, you see the call to: 00401532 |. FFD0 CALL EAX

This will transfer the control to the payload. Here is a screenshot about the payload decrypted.



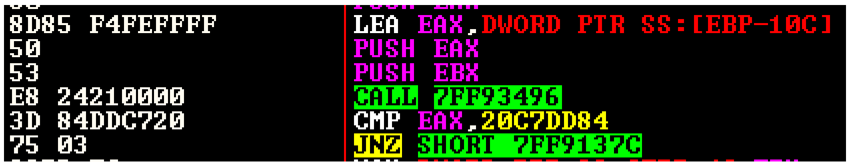
```
00000004A0: 6C 00 00 00 4E 74 4F 70  65 6E 53 65 63 74 69 6F  l   NtOpenSectio
00000004B0: 6E 00 00 00 53 2D 31 2D  35 2D 33 32 2D 35 34 34  n   S-1-5-32-544
00000004C0: 00 00 00 00 53 65 52 65  73 74 6F 72 65 50 72 69      SeRestorePri
00000004D0: 76 69 6C 65 67 65 00 00  53 65 42 61 63 6B 55 70  vilege  SeBackUp
00000004E0: 50 72 69 76 69 6C 65 67  65 00 00 00 53 65 44 65  Privilege   SeDe
00000004F0: 62 75 67 50 72 69 76 69  6C 65 67 65 00 00 00 00  bugPrivilege
0000000500: 8B D4 0F 34 C3 00 00 00  4B 69 46 61 73 74 53 79  ‹Ô☼4Ã   KiFastSy
0000000510: 73 74 65 6D 43 61 6C 6C  00 00 00 00 61 52 65 70  stemCall    aRep
0000000520: 6F 72 74 00 61 55 70 64  61 74 65 00 69 64 3A 25  ort aUpdate id:%
0000000530: 6C 75 7C 62 69 64 3A 25  6C 75 7C 6F 73 3A 25 6C  lu|bid:%lu|os:%l
0000000540: 75 7C 6C 61 3A 25 6C 75  7C 72 67 3A 25 6C 75 00  u|la:%lu|rg:%lu
0000000550: 5C 5C 2E 5C 70 69 70 65  5C 25 6C 75 00 00 00 00  \\.\pipe\%lu
0000000560: 73 6F 66 74 77 61 72 65  5C 6D 69 63 72 6F 73 6F  software\microso
0000000570: 66 74 5C 77 69 6E 64 6F  77 73 20 6E 74 5C 63 75  ft\windows nt\cu
0000000580: 72 72 65 6E 74 76 65 72  73 69 6F 6E 5C 69 6D 61  rrentversion\ima
0000000590: 67 65 20 66 69 6C 65 20  65 78 65 63 75 74 69 6F  ge file executio
00000005A0: 6E 20 6F 70 74 69 6F 6E  73 5C 74 61 73 6B 6D 67  n options\taskmg
00000005B0: 72 2E 65 78 65 00 00 00  48 69 64 65 53 43 41 48  r.exe   HideSCAH
00000005C0: 65 61 6C 74 68 00 00 00  54 61 73 6B 62 61 72 4E  ealth   TaskbarN
00000005D0: 6F 4E 6F 74 69 66 69 63  61 74 69 6F 6E 00 00 00  oNotification
00000005E0: 73 6F 66 74 77 61 72 65  5C 6D 69 63 72 6F 73 6F  software\microso
00000005F0: 66 74 5C 77 69 6E 64 6F  77 73 5C 63 75 72 72 65  ft\windows\curre
0000000600: 6E 74 76 65 72 73 69 6F  6E 5C 70 6F 6C 69 63 69  ntversion\polici
0000000610: 65 73 5C 45 78 70 6C 6F  72 65 72 00 73 6F 66 74  es\Explorer soft
0000000620: 77 61 72 65 5C 6D 69 63  72 6F 73 6F 66 74 5C 77  ware\microsoft\w
0000000630: 69 6E 64 6F 77 73 5C 63  75 72 72 65 6E 74 76 65  indows\currentve
0000000640: 72 73 69 6F 6E 5C 70 6F  6C 69 63 69 65 73 5C 73  rsion\policies\s
0000000650: 79 73 74 65 6D 00 00 00  45 6E 61 62 6C 65 4C 55  ystem   EnableLU
0000000660: 41 00 00 00 73 62 69 65  64 6C 6C 2E 64 6C 6C 00  A   sbiedll.dll
0000000670: 25 00 6C 00 75 00 00 00  79 61 68 6F 6F 2E 63 6F  % l u   yahoo.co
0000000680: 6D 00 00 00 67 6F 6F 67  6C 65 2E 63 6F 6D 00 00  m   google.com
0000000690: 62 69 6E 67 2E 63 6F 6D  00 00 00 00 6D 69 63 72  bing.com    micr
00000006A0: 6F 73 6F 66 74 2E 63 6F  6D 00 00 00 75 70 64 61  osoft.com   upda
00000006B0: 74 65 2E 6D 69 63 72 6F  73 6F 66 74 2E 63 6F 6D  te.microsoft.com
```

The next step shows anti-analysis tricks that are employed. The call at VA 7FF91408 is iterating through process names and computing their CRC32 hash values: if a hash value matches any of those on a list of hash values of VM processes and monitoring tools like wireshark.exe, etc., this indicates that the debugging process is inside a sandbox environment or being monitored.



```
Address  Hex dump                                         ASCII
7FF90268 32 44 DD 99|B4 9D 85 2D|CE 0D 34 64|74 44 C5 63  2D▌Ö┤¥à─╬.4dtD┼c
7FF90278 8B 9C 9C 34|CE EB 46 34|FE B1 A9 5B|F3 BE E2 3C  ï££4╬δF4■▒⌐[≤╛Γ<
7FF90288 2B F0 46 3D|F7 10 AE 77|5D E9 44 F3|6F 6D BE 2D  +≡F=≈►«w]ΘD≤om╛-
7FF90298 44 02 D1 A3|91 ED 72 1D|BE 6B 93 96|58 DF 8C 27  D☻╤úæφr↔╛kôûX▀î'
7FF902A8 85 F8 FF 3B|D9 23 33 6D|00 00 00 00|50 04 F9 7F  à° ;┘#3m....P♦∙⌂
7FF902B8 40 04 F9 7F|38 04 F9 7F|2C 04 F9 7F|20 04 F9 7F  @♦∙⌂8♦∙⌂,♦∙⌂ ♦∙⌂
7FF902C8 00 00 00 00|02 00 00 80|01 00 00 80|00 00 00 00  ....☻..Ç☺..Ç....
7FF902D8 00 00 00 00|00 00 00 00|4E 61 BC 00|90 90 90 90  ........Na╝.ÉÉÉÉ
```

Furthermore, this trick is not changed. As in version 2.07 and 2.08, the 2.09 version continues to calculate the CRC32 hash of the volume name of drive C:, which is then compared with the hardcoded value 0x20C7DD84. If you get caught, you will run in infinite loop that call ZwDelayExecution ! just patch the JNZ after the call or put RET in ZwDelayExecution.



```
8D85 F4FEFFFF      LEA EAX,DWORD PTR SS:[EBP-10C]
50                 PUSH EAX
53                 PUSH EBX
E8 24210000        CALL 7FF93496
3D 84DDC720        CMP EAX,20C7DD84
75 03              JNZ SHORT 7FF9137C
```

After that I think that the CALL at VA 7FF91420 is trying to setup a KiFastSystemCall hook, this API is the lowest level API available in the “usermode” layer aka Ring3, all application’ calls pass from KiFastSystemCall, which redirects all those controls onto the Windows Kernel via an instruction called SYSENTER.

Next, because processes run by the user can’t do everything like writing in explorer.exe memory, the malware is trying to use SeDebugPrivilege and calling ZwAdjustTokenPrivilege to escalate to System privileges. It calls the SetEnvironmentVariableW API to save the original bot’s full path to the environment variable. Afterwards, it comes the injection process, depending if you have a 32 or a 64 bits operating system, the malware will launch a hallowed version of msiexe.exe and inject its code there:

## Code Injection:

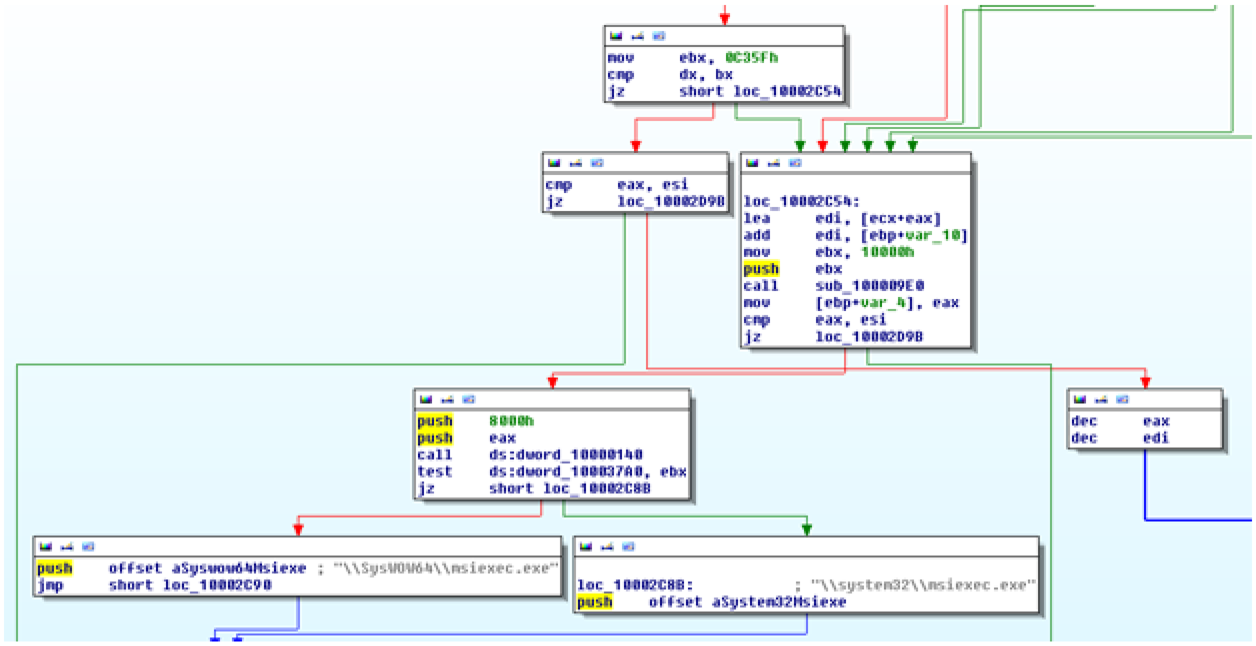
The injection process involves several steps:

As with the previous versions, the malware calls CreateFile to get the handle of the file it wants to inject. It then gets its section handle by calling ZwCreateSection, which is used by ZwMapViewOfSection to get the image of the file in memory. From this image, it extracts the size of image and the address of the entry point from the PE header.

A memory address with the same size as that of the image of the file that it wants to inject is created with PAGE_EXECUTE_READWRITE access. Then the image of the file is copied over to this memory address.

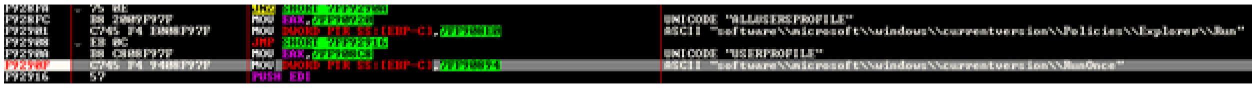
Another memory address is created with the same size as that of the image of the original bot file, also with PAGE_EXECUTE_READWRITE access. The original file is then copied over to this new memory address.

A suspended process of the file to be injected is created. The memory address containing the original file is unmapped. ZwMapViewOfSection is called with the bot’s file handle and the process handle (acquired from creating the suspended file process). So now the injected file’s process handle has a map view of the botnet file. The final step is the call to ZwResumeThread, which resume the process.



```
mov     ebx, 0C35Fh
cmp     dx, bx
jz      short loc_10002C54

cmp     eax, esi
jz      loc_10002D90

loc_10002C54:
lea     edi, [ecx+eax]
add     edi, [ebp+var_10]
mov     ebx, 10000h
push    ebx
call    sub_100009E0
mov     [ebp+var_4], eax
cmp     eax, esi
jz      loc_10002D9B

push    8000h
push    eax
call    ds:dword_10000140
test    ds:dword_100037A0, ebx
jz      short loc_10002C8B

dec     eax
dec     edi

push    offset aSyswow64Msiexe ; "\\SysWOW64\\msiexec.exe"
jmp     short loc_10002C90

loc_10002C8B:                   ; "\\system32\\msiexec.exe"
push    offset aSystem32Msiexe
```

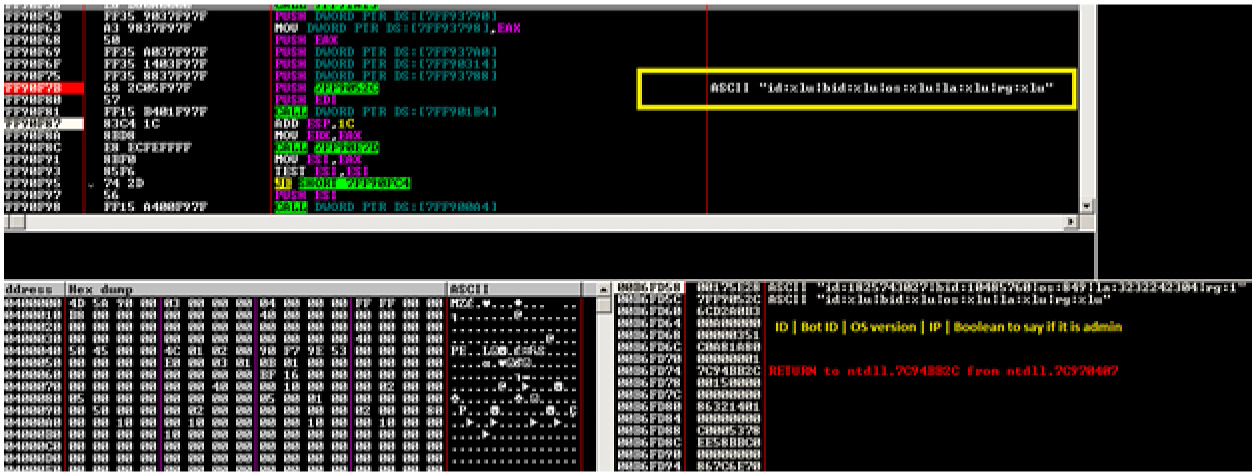
If the User is an admin, it checks that with CheckTokenMemberShip, it installes into "%ALLUSERPROFILE%" and autostarts using an uncommon Registry Path "softwaremicrosoftwindowscurrentversionPoliciesExplorerRun" – with a random Key name. If not it only installs into "%USERPROFILE%".



```
F928FA  . 75 0E          JNZ SHORT 7FF92908                UNICODE "ALLUSERSPROFILE"
F928FC    B8 2009F97F    MOV EAX,7FF90920                  ASCII "software\\microsoft\\windows\\currentversion\\Policies\\Explorer\\Run"
F92901    C745 F4 1008F97F MOV DWORD PTR SS:[EBP-C],7FF90810
F92908  . EB 0C          JMP SHORT 7FF92916                UNICODE "USERPROFILE"
F9290A    B8 C808F97F    MOV EAX,7FF908C8                  ASCII "software\\microsoft\\windows\\currentversion\\RunOnce"
F9290F    C745 F4 9408F97F MOV DWORD PTR SS:[EBP-C],7FF90894
F92916    57             PUSH EDI
```

## CnC Communication:

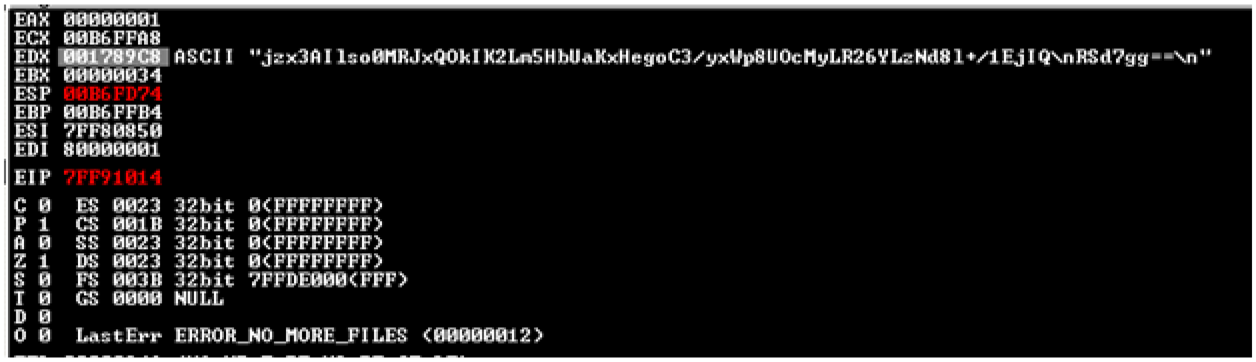
Before establishing a connection, the bot prepares the message to be sent to the C&C server. It uses the following format: **id:%lu|bid:%lu|os:%lu|la:%lu|rg:%lu**



```
FF90F5D   FF35 9037F97F   PUSH DWORD PTR DS:[7FF93790]
FF90F63   A3 9837F97F     MOV DWORD PTR DS:[7FF93798],EAX
FF90F68   50              PUSH EAX
FF90F69   FF35 A037F97F   PUSH DWORD PTR DS:[7FF937A0]
FF90F6F   FF35 1403F97F   PUSH DWORD PTR DS:[7FF90314]
FF90F75   FF35 8837F97F   PUSH DWORD PTR DS:[7FF93788]
FF90F7B   68 2C05F97F     PUSH 7FF9052C                 ASCII "id:%lu|bid:%lu|os:%lu|la:%lu|rg:%lu"
FF90F80   57              PUSH EDI
FF90F81   FF15 B401F97F   CALL DWORD PTR DS:[7FF901B4]
FF90F87   83C4 1C         ADD ESP,1C
FF90F8A   8BD8            MOV EBX,EAX
FF90F8C   E8 ECFEFFFF     CALL 7FF90E7D
FF90F91   8BF0            MOV ESI,EAX
FF90F93   85F6            TEST ESI,ESI
FF90F95   74 2D           JE SHORT 7FF90FC4
FF90F97   56              PUSH ESI
FF90F98   FF15 A400F97F   CALL DWORD PTR DS:[7FF900A4]
```

```
0006FD58  0012512B  ASCII "id:1025743027|bid:1040576010|os:847|la:3232242304|rg:1"
0006FD5C  7FF9052C  ASCII "id:%lu|bid:%lu|os:%lu|la:%lu|rg:%lu"
0006FD60  6CD2A0D3
0006FD64  00000000      ID | Bot ID | OS version | IP | Boolean to say if it is admin
0006FD68  00000351
0006FD6C  C0001800
0006FD70  00000001
0006FD74  7C94882C  RETURN to ntdll.7C94882C from ntdll.7C970407
0006FD78  00150000
0006FD7C  00000000
0006FD80  86321401
0006FD84  00000000
0006FD88  C0005378
0006FD8C  EE58BBC0
0006FD90  00000000
0006FD94  8672C6F20
```

This string is encrypted using RC4 with a hard-coded key of length 0x20 and is further encoded using base64. The message is then sent to the server. Once a message is received, the bot calculates the CRC32 hash of the message without including the first DWORD. If the calculated hash matches the first DWORD, the message is valid. Later it is decrypted using RC4 with the VolumeSerialNumber as the key. After the RC4 decryption the message is in the format gn([base64-encoded string]). This used to be just the base64-encoded string, but for some reason the author decided not to make the server backward compatible with the older bot versions. Then it decodes the base64 string inside the brackets to get the message in plain text.



```
EAX 00000001
ECX 00B6FFA8
EDX 001789C8 ASCII "jzx3AIlso0MRJxQOkIK2Lm5HbUaKxHegoC3/yxWp8UOcMyLR26YLzNd8l+/1EjIQ\nRSd7gg==\n"
EBX 00000034
ESP 00B6FD74
EBP 00B6FFB4
ESI 7FF80850
EDI 80000001

EIP 7FF91014

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_NO_MORE_FILES (00000012)
```

The first DWORD of the message is used as a multiplier to multiply a value in a fixed offset. The DWORD in that offset is used as an interval to delay calling the thread again to establish another connection. The next byte indicates what action to carry out – there are seven options:

- Case 1 (download EXE): Connect to the domain decrypted from the message to download an EXE file. Save the file to the %tmp% location with a random name and run the process.
- Case 2 (load plug-ins): Connect to the domain decrypted from the message, install and load plug-ins. The plug-ins are decrypted by RC4 using the same key of length 0x20h.
- Case 3 (update case): Connect to the domain to get the update EXE file. If a file name of VolumeSerialNumber is present in theregistry, then save the PE file to the %tmp% location with a random name; else save it to the current location with the name of the file as VolumeSerialNumber. The file in %tmp% is run, while the current process terminates. It also sends the message 'kill' xor'ed by VolumeSerialNumber to terminate the older process.
- Case 4 (download DLL): Connect to the domain and save the DLL file to the %alluserprofile% location. The file is saved as a .dat file with a random name and loaded from a specified export function. The registry is modified so it can be auto-loaded by the bot.
- Case 5 (delete DLLs): Delete and uninstall all the DLLs loaded and installed in Case 4.
- Case 6 (delete plug-ins): Uninstall all the plug-ins loaded in Case 3.
- Case 7 (uninstall bot): Suspend all threads and uninstall the bot.
- After executing the action based on which instruction it received, another message is sent to the server to notify it that the action has been completed:

**id:%lu|tid:%lu|res:%lu**

- **id** is the VolumeSerialNumber
- **tid** is the next byte (task id) after the byte displaying the case number in the message received
- **res** is the result of whether or not the task was carried out successfully.

Once the message has been sent, the thread exits and waits for the delay interval period to pass before it reconnects to the server to receive additional instructions.

## Conclusion:

Andromeda's current version 2.09 increased the barriers that it has set up for security researchers. The new features raise additional difficulty for analysis, but are still easy to skip.

We anticipate that the Andromeda botnet will keep on evolving. Our botnet monitoring system is continuing to track its activities and we will respond immediately when it enters its next generation.

## Credits and References:

https://blog.fortinet.com/post/andromeda-2-7-features

Posted: September 28, 2015

Author

**Ayoub Faouzi**

**VIEW PROFILE**
Ayoub Faouzi is interested to computer viruses and reverse engineering, In the first hand, he likes to study PE packers and protectors, and write security tools. In the other hand, he enjoys coding in python and assembly.