

ANDROMEDA BOT ANALYSIS - PART 2

Andromeda Bot Analysis part 2

JUMP TO

SELECT POST SECTION ▼


35

 Tweet

2

 Share

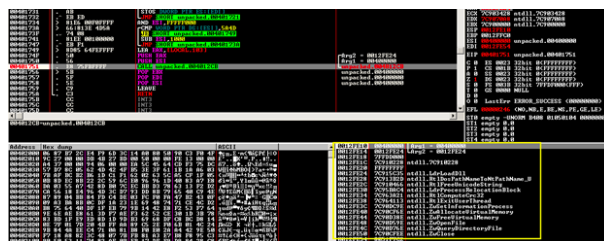
6

 Share submit
reddit 0 Like

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 :



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

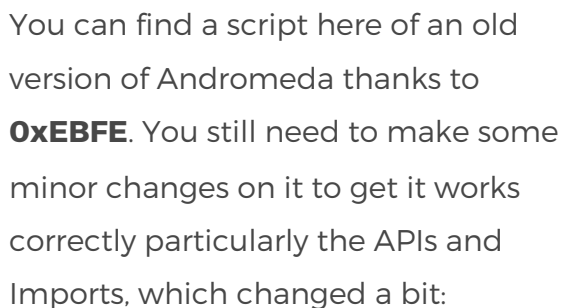
```
def calc_hash(string):
```

```
    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 performing 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

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



```
from idutils 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', 'lstrcpwW',  
'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
|. FFDO CALL EAX

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

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.

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.

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

INFOSEC INSTITUTE

INTENSE SCHOOL

CERTIFICATION TRACKER

Try out our new FREE Phishing Simulator**Phish Your Friends!**

inject its code there:

Code Injection:



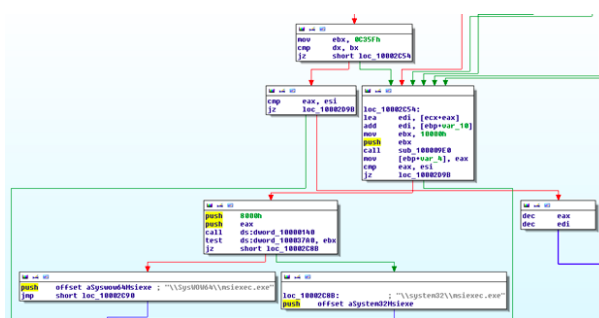
The injection process involves several steps:

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.



If the User is an admin, it checks that with CheckTokenMembership, it installes into "%ALLUSERPROFILE%" and autostarts using an uncommon Registry Path

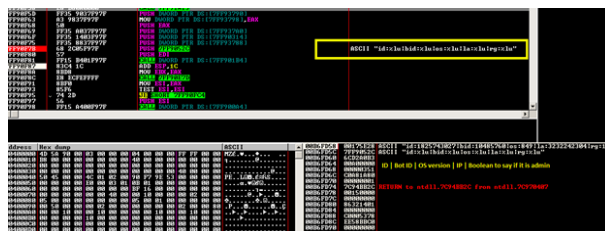
"software\microsoft\windows\currentversion\Policies\Explorer\Run" – with a random Key name. If not it only installes into "%USERPROFILE%".



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



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.



ETHICAL HACKING TRAINING

– RESOURCES (INFOSEC)

Want to learn more? The InfoSec Institute [Ethical Hacking course](#) goes in-depth into the techniques used by malicious, black hat hackers with attention getting lectures and hands-on lab exercises. You leave with the ability to quantitatively assess and measure threats to information assets; and discover where your organization is most vulnerable to black hat hackers. Some features of this course include:

- Dual Certification - CEH and [CPT](#)
- 5 days of Intensive Hands-On Labs
- CTF exercises in the evening

FIRST NAME *

LAST NAME *

COMPANY

EMAIL *

PHONE *

JOB TITLE *

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 the registry, 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>

PREV: AND...



AUTHOR

Ayoub
Faouzi

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.



Challenges Faced
By CISOs:
Balancing
Security...



What's Worse:
APT's or Spear
Phishing?



Andromeda Bot
Analysis part 1



Exploiting
Corporate Printers



0 Comments

InfoSec Institute Resources

 Login ▾ Recommend Share

Sort by Best ▾



Start the discussion...

Be the first to comment.

ALSO ON INFOSEC INSTITUTE RESOURCES

WHAT'S THIS?

Security+ Study Guide: Domain 1.0 – Network Security

1 comment • 3 months ago

**Infolookup** — very nice writeup.**How Hackers Violate Privacy and Security of the Smart Home**

1 comment • 18 days ago

**Anne Cerrato** — Follow+ success of many who are making profit monthly by doing an online job... Learn more on my~profile**Introducing the InfoSec Institute Job Board**

1 comment • 2 months ago

**Monica Lieberman** — Follow path of thousands!who are earning cash each month by freelancing online... Get informed**Establishing a Secure IPSec Connection to the Cloud Server**

1 comment • 14 days ago

**Anne Cerrato** — Follow^ success of many who are making profit monthly by doing an online job... Learn more on my~profile Subscribe Add Disqus to your site Privacy

DISQUS

About InfoSec

InfoSec Institute is the best source for high quality [information security training](#). We have been training Information Security and IT Professionals since 1998 with a diverse lineup of relevant training courses. In the past 16 years, over 50,000

Connect with us

Stay up to date with [InfoSec Institute](#) and [Intense School](#) - at info@infosecinstitute.com

 Like 494 Follow @infosecedu

Join our newsletter

Get the latest news, updates & offers straight to your inbox.

ENTER YOUR EMAIL

SUBSCRIBE

individuals have trusted
InfoSec Institute for their
professional development
needs!

© INFOSEC RESOURCES 2015