Application of radare2 illustrated by Shylock/Caphaw.D and Snakso.A analysis

Anton Kochkov

PHDays IV, May 21, 2014

Intro

radare2

Please use radare2 from git

Warnings

- There is a nasty bug in r2 for now, please bear with us
- This is a quick writeup

snakso.A

- md5: 52852ac955ba03e4ebb012c55550dca3
- Linux 64bit rootkit
- Lame

Shylock/Caphaw.D

- md5: dcc876357354acaf2b61ee3e839154ad
- Windows 32bit Financial trojan
- Many modifications

Shylock/Caphaw.D

File hash

```
$ rahash2 -a sha256 shylock_d.exe
shylock_d.exe: 0x00000000-0x00049000
sha256: 35ccf0e051fb0ff61e52ff4130eb3 \
    8521f954f90ad9c97be59af1e7901974557
$
```

Mitigations

```
$ rabin2 -k '*' shylock_d.exe
archs=0:0:x86:32
pe.seh=true
$
```

Shylock/Caphaw.D Sections

```
$rabin2 -S shylock_d.exe
[Sections]
idx=00 addr=0x00001000 ... name=.text
idx=01 addr=0x00009000 ... name=.rdata
idx=02 addr=0x0000d000 ... name=.data
idx=03 addr=0x0000e000 ... name=.debug1
idx=04 addr=0x0000f000 ... name=E1
idx=05 addr=0x00012000 ... name=E2
idx=06 addr=0x00016000 ... name=B_0
idx=07 addr=0x00046000 ... name=.rsrc
idx=08 addr=0x00047000 ... name=.reloc

9 sections
$
```

Yara

```
$ r2 shylock_d.exe
```

Radare2 opens PE and automatically jumped to the entrypoint.

Lets run YARA on it:

```
> yara scan
    Microsoft_Visual_C___6_0_DLL
    Microsoft_Visual_C___6_0
    Microsoft_Visual_C___7_0
    dUP_v2_x_Patcher
    Microsoft_Visual_C__v7_0___Basic__NET
```

Shylock/Caphaw.D Imports

```
> il
[Linked libraries]
KERNEL32.dll
GDI32.dll
USER32.dll
ADVAPI32.dll
WINMM.dll
WinSCard.dll
ole32.dll

7 libraries
```

Shylock/Caphaw.D Imports (cont.)

Interesting functions:

```
> ii
    ...
- WinSCard.dll_SCardAccessStartedEvent
- KERNEL32.dll_VirtualProtect
- KERNEL32.dll_VirtualAlloc
- KERNEL32.dll_VirtualQuery
```

Disassembly

Command line functions

- ‘pd’
- ‘pi’

Visual mode: ‘Vp’

Autoanalysis of the whole file: ‘aa’

GetProcessHeap

```
[0x004044b0 255 shylock_d.exe]> pd $r @ entry0
 sub esp, 0x150
 push edi
 lea eax, [esp+0x8]
 push eax
 call dword [reloc.KERNEL32.dll_GetStartupInfoA] ;[1]
 mov edi, [reloc.KERNEL32.dll_GetProcessHeap]
 call edi

 test eax, eax
 je 0x40462f ;[2]
 push esi
 push 0x1000 ;  0x00001000
 push 0x8 ;  0x00000008
 push eax
 call dword [reloc.KERNEL32.dll_HeapAlloc] ;[3]
 call dword [reloc.KERNEL32.dll_GetCommandLineA] ;[4]
```

Lets press ‘d’ and then choose ‘f’ = ‘df’ - create function, and go to the je 0x40462f (just press [2])

```
push 0x0
call dword [reloc.KERNEL32.dll_ExitProcess] ;[1]
pop esi
pop edi
add esp, 0x150
ret
```

This is just ExitProccess on fail of getting handle to the default heap of calling process

VirtualProtect

GetModuleHandleA to get the base address of the calling process and changin permissions of the committed memory via VirtualProtect

```
push 0x0
call dword [reloc.KERNEL32.dll_GetModuleHandleA] ;[1]
mov ecx, [eax+0x3c]
lea edx, [esp+0x8]
push edx
add ecx, eax
mov ecx, [ecx+0x50]
push 0x40 ; "@" ;  0x00000040
push ecx
push eax
call dword [reloc.KERNEL32.dll_VirtualProtect] ;[2]
```

WineDbg as gdbserver + radare2

Lets start winedbg in gdb-proxy mode:

```
$ winedbg --gdb --no-start shylock_d.exe
001e:001f: create process 'Z:\\home\\xvilka\\shylock_d.exe
001e:001f: create thread I @0x502b5a
target remote localhost:33563
```

In the output of this command you see line with gdbserver listening port, like “target remote localhost:33563” in our example.
“–no-start” option stop program at the start.

```
r2 -a x86 -b 32 -D gdb://localhost:33563
```

Snakso.A

Wat.

[0x000062db]> i~stri strip false

Strings

```
iz~? 332
iz~[7]|sort|less
```

Strings (cont.)

```
iz | grep -E '.*([0-9]{1,3}[\.]){3}[0-9]{1,3}*'
```

```
string=188.40.102.11
string=127.0.0.1
string=91.123.100.207
string=149.20.4.69
string=149.20.20.133
string=192.168.1.40
string=149.20.4.69
string=149.20.4.69
string=64.189.125.254
string=10.0.2.15
string=10.0.2.14
string=192.168.1.1
string=192.168.1.33
string=192.168.1.38
```

Strings (cont.)

- Some HTTP error codes
- Apache
- nginx
- KERNEL_VERSION_XXX
- Inject

Likely one of those low-level httpd injector

Interesting functions

`is~?hide` 51 `is~?test` 19

Time to reverse the funny ones!

Persistence

```
[0x00006130]> pdf@sym.formation_module_startup_command
movsxd rsi, esi
sub rsp, 0x10
xor eax, eax
cmp rsi, 0x3f
mov rdx, rdi
jbe loc.00002e63
mov rsi, 0x20646f6d736e690a ;  0x20646f6d736e690a
mov ecx, 0x29 ; ")" ;  0x00000029
mov eax, 0x1 ;  0x00000001
mov [rdi], rsi
lea rdi, [rdi+0x8]
mov rsi, str._lib_modules_2.6.32_5_amd64_kernel_sound_
rep movsb
lea rdi, [rdx+0x31]
mov rsi, str.module_init_ko
...
```

```
[0x00006130]> !rax2 -s 0x20646f6d736e690a
domsni
[0x00006130]> !rax2 -s 0x20646f6d736e690a | rev
insmod
```

It builds the string `insmod /lib/modules/2.6.32-5-amd64/kernel/sound/module_init.ko`

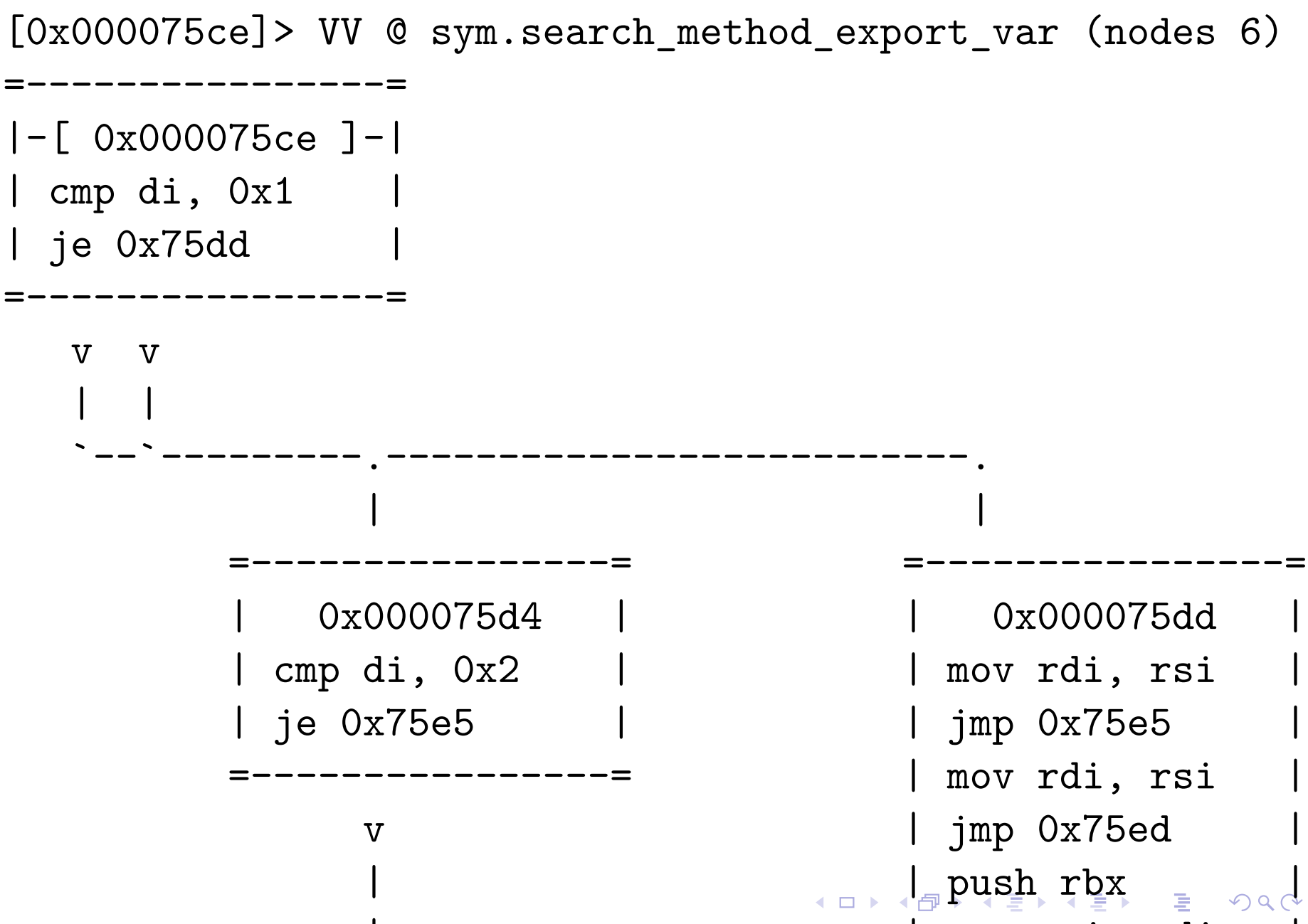
This function is called from sym.write_startup_module_command_in_file

Let's be lazy clever:

```
[0x00006130]> pdf@sym.write_startup_module_command_in_file~
str.etc_rc_local
```

Super-lame persistence system.

Symbols resolving



The graph is not-super exact, because this function is doggy, but you get the idea.

This is (should, since the malware is wrongly coded) use a first method to get symbols, and a second one as fallback.

```
[0x0000717c]> pdf@sym.search_method_find_in_file
```

A stupid grep in System.map

```
[0x00006130]> pdf@sym.search_method_exec_command
```

Equivalend to ‘cat /proc/kallsyms > /.kallsyms_tmp

Learn to UNIX

```
[0x00006130]> s sym.execute_command
[0x00006130]> pdf~XREF
; UNKNOWN XREF from 0x00006118 (fcn.000060fc)
; JMP XREF from 0x000061c0 (fcn.00006189)
; CALL XREF from 0x00006184 (fcn.00006189)
; CALL XREF from 0x00006196 (fcn.00006189)
; CALL XREF from 0x000061a4 (fcn.00006189)
; JMP XREF from 0x0000618f (fcn.00006189)
; JMP XREF from 0x0000619d (fcn.00006189)
; CALL XREF from 0x000061b7 (fcn.00006189)
; JMP XREF from 0x000061ae (fcn.00006189)
```

```
[0x00006130]> pdf@sym.execute_command~str
          str._bin_bash
[0x00006130]> pdf@sym.execute_command~call
call 0x6189 ; (sym.execute_command)
call 0x619b ; (sym.execute_command)
call 0x61a9 ; (sym.execute_command)
call 0x61bc ; (sym.execute_command)
```

This function is a wrapper to /bin/bash -c

```
[0x00000064]> pdf@sym.get_kernel_version~str
[0x00000064]> pdf@sym.get_kernel_version~"
mov r10, 0x722d20656d616e75 ; "uname -r" ;  0x722d20656d616
mov word [rbp+0x8], 0x3e20 ; " >" ;  0x00003e20
[0x00000064]>
```

Patching!

The rootkit hooks some functions:

```
0x0000a3db  lea rax, [rbp+0x1]
0x0000a3df  mov byte [rbp], 0xe9 ;  0xffffffffffffffe9
0x0000a3e3  lea rsi, [rsp+0x20]
0x0000a3e8  mov ecx, 0x13 ;  0x00000013
0x0000a3ed  mov rdi, rax
0x0000a3f0  rep movsb
0x0000a3f2  mov rdi, rax
0x0000a3f5  mov esi, 0x14 ;  0x00000014
0x0000a3fa  call fcn.0000a3ff
```

Not that obvious, eh? Actually, it puts 0xe9 in the prologue.

Cross-references

```
af 0x60cc
```

- search_method_exec_command (736e)
- search_method_find_in_file (747b)

Decryption function

- get_task()
- Static password:
  str.GL7mCfcoW5wIobokBAkia7kmqy3KDcN3GFIeG
  iO3f9GtES09ZyAAGvM9pi787mYsIHSVOUQWGyYW7B
  DI8mAChgFwT5deL3N7WXyIccsaiC90MkSE5w5dGIQu0GcMSec

Counter measures

- search_engines_ip_array

References

- CrowdStrike - IFrame injecting rootkit
- Kaspersky
- Trusteer - Evading Shylock's new trick
- Shylock in depth malware analysis
- BAE Systems - Shylock Whitepaper
- Quequero - Shylock in depth analysis

Credits

- pancake
- jvoisin
- dso