Demigod: The Art of Emulating Kernel Rootkits

> Blackhat USA August 5th, 2020

NGUYEN Anh Quynh NGUYEN Hong Quang DO Minh Tuan

About us

NGUYEN Anh Quynh

- Nanyang Technological University,
 Singapore. PhD in Computer
 Science
- Operating System, Virtualization, Binary analysis, etc
- Frequent speaker of Blackhat USA/EU/Asia, Defcon, Recon, Syscan, HackInTheBox, etc
- Founder of few open source reverse engineering frameworks: Capstone, Unicorn & Keystone.
- Contact: aquynh @ gmail.com

About us

NGUYEN Hong Quang

- Security Researcher at Viettel Cyber Security (VCS)
- Interested in Fuzzing and Exploits
- I would like to thank my wife & my children. Without them, I would finish this research few months earlier ;-)
- Contact: quangnh89 @
 gmail.com

About us

DO Minh Tuan

- Security Researcher at CyStack., JSC
- Interested in Fuzzing and Exploits
- BabyPhD CTF team's member
- Speaker at Blackhat USA/Asia, T2, XCon, etc
- Contact: tuanit96 @ gmail.com

Agenda

- Background & Motivation
- Design & Implementation
 - Cross-platform framework
 - Windows
 - MacOS
 - Linux
- Demos
- More applications of Demigod
- Conclusion

Background & Motivations

Kernel Rootkit

- Computer system = Userland (ring 3) + OS Kernel (ring 0)
 - Kernel operates at lowest level, with full control on whole system
- Malware run inside kernel
 - God-mode code, with full power
 - Can fool all monitoring & defensive mechanisms
 - Very hard to detect & kill
 - Can be creative with lots of tricks to evade detection
 - Can even kill or fool any anti-malware tools
 - Mostly come in the form of kernel modules/drivers
 - Easy to build & maintain + high compatibility
 - Windows: .SYS file
 - MacOS: .KEXT file
 - Linux LKM: .KO file

Analyze Kernel Rootkit

- ➤ Goal: analyze rootkits in kernel modules/drivers
 - .SYS, .KEXT, .KO
- Dynamic analysis is troublesome
 - Need 2 machines (physical or virtual)
 - One machine to load kernel rootkits
 - Another machine to run analysis tools
 - Remote debugger
 - Crash machine
 - Low visibility when kernel run at very low level
 - Few tools built for kernel-level analysis
- Static analysis
 - Packed code?
 - Inaccurate analysis
 - Real kernel setting depends on config
 - Kernel modules are blind
 - Missing API

Better Tool for Kernel Rootkits?

Analyze ring 0 code in ring 3?

- No more machine crash
- No more headache of setting up separate machines, or virtual machines
- No more risk with kernel level malware
- Good visibility
- Existing tools can be reused
 - Monitor, trace, debug, etc
- Cross-platform-architecture analysis
 - Analyze Windows kernel rootkits on Linux and vice versa
 - Analyze Arm rootkits on X86 would be fun ;-)
- High-level language scripting is possible such as Python
 - So it is quick & efficient to build new analysis tools

Emulator Approach

- > Emulate OS kernel in software, so kernel modules/drivers can be run inside
 - Emulator runs in ring 3
 - No more machine crash
 - No more headache of setting up separate machines, or virtual machines
 - No more risk with kernel level malware
 - Existing tools can be reused
 - Excellent visibility since we can all execution code
 - Instrumentation at various levels
- Cross-platform-architecture analysis
 - Cross-platform-architecture emulator
- High-level language scripting is possible such as Python
 - Emulator as framework
 - Python-based emulator ;-)

Design of Demigod

Qiling Emulator (qiling.io)

- Emulator for userland code only
 - Cross-platforms: Windows, MacOS, Linux
 - Cross-architectures: X86, Arm, Arm64, Mips
 - Multiple executable formats: PE, MachO, ELF
 - Python-based framework
 - Instrumentation at various levels
 - Emulate system API (userland)
 - Enable debugging (with GDB server)
- > Cannot handle kernel code \rightarrow so cannot emulate kernel rootkis
 - Loaders for kernel modules are different from userland binaries
 - Different format structures
 - Do not emulate kernel components
 - Do not emulate kernel API
 - Kernel module has different execution scheme
 - Load vs Serve request from userland

Emulator for Kernel Rootkits - 1

- Build on top of Qiling!
 - Reuse the excellent frameworks with many ready features
 - Loader + little modification
 - Symbol relocation
 - Imported functions
 - Emulate kernel components
 - Emulate kernel API by hooking & emulate its semantics
 - Sometimes we can forward to kernel code (PASS-THRU)
 - Instrument memory access and code execution for dynamic analysis
 - System API hooking
 - Automatic alloc/map missing memory to enable continuous execution
 - Facility to assist dynamic analysis
 - Reuse Qiling GDB server for debugging

Emulator for Kernel Rootkits - 2

- Userland code is typically just 1 phase emulation
- Kernel driver is 2-phase scheme
 - Driver needs to be loaded first, then it stays in kernel to serve user request
 - Emulate driver entry
 - Locate entry to emulate
 - Windows: DriverEntry
 - MacOS: realmain
 - Linux: module_init
 - During emulation, extract code paths for next phase
 - Rootkit syscalls
 - IOCTL callbacks
 - Other registered callbacks
 - Emulating selected code paths
 - User choose which code path to emulate
 - Input provided by user

Windows

Windows Overview

> Windows kernel

- Kernel image: ntoskrnl.exe
 - cache manager
 - executive
 - kernel
 - security reference monitor
 - memory manager
 - Scheduler
- Driver: .SYS
 - Portable Executable (PE) format
 - Subsystem: native
- > Goal: emulate kernel rootkit in .SYS driver



Load .SYS File

- ➢ PE loader for .SYS file
 - Parse all sections of PE file and map all sections into emulator
 - Resolve Import Address Table and load all dependent DLLs into emulator.
 - Relocation

Setup CPU context

- Common registers (RSP,)
- Control registers (CR4, CR8...)
- Initialize Windows structures
 - EPROCESS, DRIVER_OBJECT, KUSER_SHARED_DATA, ..
 - Setup IMAGE_LOAD_CONFIG_DIRECTORY: SecurityCookie value
- Locate initialization function
 - NTSTATUS NTAPI DriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING RegistryPath);

Windows Internal Structures

- DRIVER_OBJECT
 - represent image of a loaded kernel-mode driver.
- ≻ IRP
 - represent an I/O request packet.
- KUSER_SHARED_DATA
 - data structure which is shared between user mode and kernel mode.
- IO_STACK_LOCATION
 - define an I/O stack location
- ≻ MDL
 - partially opaque structure that represents a memory descriptor list (MDL).

Emulating Driver Initialization

- Setup arguments of DriverEntry():
 - 1st argument: A pointer to a DRIVER_OBJECT structure (driver object)
 - 2nd argument: A pointer to a UNICODE_STRING structure (path to driver registry).
- Emulate from DriverEntry
 - Emulate Windows API functions
 - CreateloDevice
 - CreateSymbolicLink

Hook Windows Kernel APIs

- Emulate some Windows APIs necessary for driver to work
 - RtllnitUnicodeString
 - IoCreateDevice, IofCompleteRequest
 - Memory management:
 - ExAllocatePool / ExFreePool
 - ExAllocatePoolWithTag / ExFreePoolWithTag
 - MmGetSystemRoutineAddress
 - Runtime functions (strcpy, wcscpy, ...)
 - Registry functions: NtOpenKey, ...
 - Thread functions: PsCreateSystemThread
 - Object functions: ObfDereferenceObject, ObOpenObjectByPointer

<pre># void IofCompleteRequest(</pre>
PIRP Irp,
CCHAR PriorityBoost
);
@winapi(cc=STDCALL,
params={
"Irp": POINTER,
"PriorityBoost": CCHAR
30
<pre>def hook_IofCompleteRequest(ql, address, params): return None</pre>

Emulate Code Path

Userspace communicates with driver via IOCTL code (DeviceIoControl) or ReadFile/WriteFile

- Create structures in memory
 - IRP
 - IO_STACK_LOCATION, IO_STACK_LOCATION_PARAM
- Fill all structures with appropriate data
- Setup arguments for callbacks:
 - IRP_MJ_DEVICE_CONTROL
 - IRP_MJ_READ/IRP_MJ_WRITE
- Start emulate callbacks
- On return, determine NTSTATUS value & return data.
 - Retrieve IRP structure from emulator memory

Windows I/O methods

- ➢ BUFFERED_IO
 - Create a large enough buffer and assign address of buffer to irp.AssociatedIrp.SystemBuffer
- > DIRECT_IO
 - Create a buffer to store data
 - Create a MDL structure
 - MappedSystemVa == StartVa == address of buffer
 - ByteCount = size of buffer
- > NEITHER_IO
 - UserBuffer field of IRP structure points to address of buffer

Demigod Tools

- dm-unpack.py: A tool to unpack kernel mode packer
 - > Dump PE file from emulator environment to file
 - > Add new section to store Import Address Table (IAT)
 - Rebuild Import Address Table
 - Find all call/jmp to possible IAT function pointers
 - Verify pointers by checking export address table of other modules
- ✤ dm-ioctl-reverse.py: a tool to help when working with IOCTL codes
 - > Bruteforce emulation to find IOCTL from binary
 - Check and count all basic blocks

Demo: Debug Windows driver

- ➤ Setup:
 - Enable gdb mode
 - ql.debugger = True
 - ql.debugger= ":9999"
- > Execute script
- ➢ Connect IDA to port 9999 & trace
- > Demo:

https://youtu.be/5Tzc0rMfYSg

n ID	A - C:\Users\Q	2uangNH\A	ppData\Lo	cal(remp)	1001/331.10								-				
Eile	Edit Jump	Searc <u>h</u>	<u>V</u> iew D	eb <u>u</u> gger	Options	Windows	Help		f 🖂 E	רה		ลั โก	50	: 🖂 📲 🗚			
		te GDB debi	igger			: 🗊 🖬	1 . ·	4 - V		5 J	land land	ut (y	90	. 60 09 09.			2
			_			_									_		
1	ibrary function	Data	Regular	function	Unexplored	d 📕 Instru	uction E	xternal s	ymbol				i				
	Debug	View	×	[A]	Struct	ures	×		E	nums		×					
	IDA \	/iew-EIP	X		Hex	View-3	×			-				💓 General regist	ers		8×
Ì	MEMORY:0	0011420	јмр ;	100_11									-	EAX 10089880	 	MEMOR	OF C
	MEMORY:0 MEMORY:0	0011432	1nc 114	132 :					CODE	XREF:	MEM	1RY : 80	1	EDX 00000000	4	MEMOR	IF C
	MEMORY : 0	0011432	push	esi		•			,		- nerts			EBX 00000000	41	MEMOR	SF C
•	MEMORY:0	0011433	mov push	offset	aDevic	8 eHello			; "\\De	vice\\	\Hell	.0''		ESP FFFFCFD8 FBP FFFFCFFC	4		ZF 1 AF 0
	MEMORY:0	001143E	lea	eax, [ebp-OCh]								ESI 0000000	4	MEMOR	PF 1
	MEMORY : 0	0011441	0.011	can											1.1	ИГ МОЛ	
		out the	Lall	esi ;	off_100	0329D								EDI 0000000	4	HE HUR	
	MEMORY:0 MEMORY:0	0011444 0011449	push lea	esi ; offset eax, [off_100 aDosde ebp-1Ch	0329D vicesHe 1	11		; "\\Do:	sDevi	ces\'	Hello	e i	EDI 00000000 EIP 00011433 EFL 00000044	4 1 4 1	MEMOR	
	MEMORY:0 MEMORY:0 MEMORY:0 MEMORY:0	0011444 0011449 001144C	push lea push	esi ; offset eax, [eax	off_100 aDosde ebp-1Ch	0329D vicesHe] 0220D	211		; "\\Do:	sDevio 	ces\'	Hello	r i	EDI 00000000 EIP 00011433 EFL 00000044	4 4	MEMOR	01 6
	MEMORY: 0 MEMORY: 0 MEMORY: 0 MEMORY: 0 UNKNOWN 00	0011444 0011449 001144C 001144C 001144C:	push lea push call MEMORY:0	esi ; offset eax, [eax esi ; 001144C	off_100 aDosde ebp-1Ch off_100 (Synchron	0329D vicesHe] 0329D nized wi	th EIP)		; "\\Do:	sDevi 	ces\'	Hello		EDI 00000000 EIP 00011433 EFL 00000044	4 4 1	MEMOR	
0 0 0	MEMORY: 0 MEMORY: 0 MEMORY: 0 MEMORY: 0 UNKNOWN 00	0011444 0011449 001144C 001144C 001144C:	push lea push call MEMORY:0	esi ; offset eax, [eax esi ; 001144C	off_100 aDosde ebp-1Ch off_100 (Synchron	0329D vicesHe] 0329D nized wi	th EIP)		; "\\Do:	sDevi(ces\'	Hello,)' •	EDI 0000000 EIP 00011433 EFL 00000044	4 I	MEMOR	
Он	MEMORY: 9 MEMORY: 9 MEMORY: 9 MEMORY: 9 UNKNOWN 00 4 Ex View-1	0011444 0011449 001144C 001144C 001144C:	push lea push call MEMORY:0	esi ; offset eax, [eax esi ; 001144C	off_100 aDosde ebp-1Ch off_100 (Synchron	0329D vicesHe] 0329D nized wi	th EIP)		; "\\Do:	sDevid	ces\'	Hello , 7 ×	•	EDI 99999699 EIP 99911433 EFL 9999044 <	4 I		8 ×
O H	MEMORY: 9 MEMORY: 9 MEMORY: 9 MEMORY: 9 UNKNOWN 00 * Ex View-1	0011444 0011449 0011440 001144C 001144C: 001144C:	push lea push call MEMORY:0 III	esi ; offset eax, [eax esi ; 001144C	off_100 aDosde ebp-1Ch off_100 (Synchron	0329D vicesHe] 0329D nized wi	th EIP)	79 i	; "\\Do: spatch.	sDevi(ces\\ dby	Hello F ×	FEFI	EDI 00000000 EIP 00011433 EFL 00000044 <	9??	MEMOR	đ ×
0 H 3001 3001	MEMORY: 0 MEMORY: 0 MEMORY: 0 MEMORY: 0 UNKNOWN 00 C C WKW-1 1A6A 69 1A7A 65 1A8A 6F	0011444 9011449 9011440 901144C 901144C: 73 79 61 20 20 57 90 73 90	74 63 6F 72	es1 ; offset eax, [eax esi ; 001144C	off_100 aDosde ebp-1Ch off_100 (Synchron 00 CC 4 21 0A 6 76 09 6	0329D vicesHe] 0329D nized wi 07 6F 6F 00 CC 5C 09 00 63	th EIP)	79 i 86 e 86 o	; "\\Do: spatch. ,-World .s.D.e.	sDevic . Good ! \ v.i.c	ces\\ dby .D. .e.	Hello 5 ×	FEFT	EDI 999696999 EIP 99911433 EFL 99996944 <	₩ 1 ₩ 1	MEMOR	8 ×
0 H 1001 1001 1001	MEMORY: 0 MEMORY: 0 MEMORY: 0 MEMORY: 0 UNKNOWN 0 ex View-1 1A6A 69 1A7A 65 1A8A 6F 1A9A 73 1A9A 78	0011444 0011449 0011440 001144C 001144C 001144C 001144C 001144C 001144C 001144C 00050 00500 00500	The second secon	es1; offset eax, esi; 001144C 68 0A 6C 64 65 08 65 08 28 57	off_100 aDosde ebp-1Ch off_100 (Synchron 00 CC 4 21 0A 6 76 00 6 6C 00 6	0329D vicesHe] 0329D nized wi 7 6F 6F 00 CC 50 9 00 63 C 00 6F	th EIP) - 64 62 - 64 62 - 64 65 - 60 60 - 60 60	79 i 86 e 86 o 86 s 86 s	; "\\Do: spatch. ,-World .s.D.e. .\.H.e.	sDevic .;Good !;\ v.i.c 1.1.o	ces\\ dby .D. .e.	,Hello , 5 X	FEFI	EDI 999696999 EIP 99911433 EFL 99996944 III Stack view FFFFC 22222	₩ 1 ₩ 1	MEMOR	8 ×
0 H 1001 1001 1001 1001 1001	MEMORY: 9 MEMORY: 9 MEMORY: 9 MEMORY: 9 UNKNOWN 00	0011444 0011449 001144C 001144C 001144C 001144C 001144C 001144C 001144C 0005 005 005 005 005 005 005 005 005 0	call push call MEMORY:0 m 74 63 6F 72 44 00 44 00 48 00 6F 20 50 00	es1; offset eax, [eax esi;; 001144C 68 0A 6C 64 6C 64 65 00 65 00 20 57 28 00	off_100 aDosde ebp-1Ch off_100 (Synchron 00 CC 4 21 0A 0 76 00 6 6C 00 6 6F 72 6 41 00 3	0329D vicesHe] 0329D nized wi 7 6F 6F 0 CC 50 9 00 6F C 00 6F C 04 27 B 00 3E	th EIP) th EIP) 64 62 300 44 80 65 60 60 60 60 60 60 60 60 60 80	79 i. 88 e 88 o 88 s CC H 88 D	; "\\Do: spatch. .s.D.e. .\.H.e. elloW	sDevic .;Good !;\ v.i.c 1.1.o orld! A.;.;	dby .D. .e. 	,Hello	FEFI	EDI 99569699 EIP 99911433 EFL 99999644 <	₩ 1 ₩ 1		8 ×
001 001 001 001	MEMORY: 9 MEMORY: 9 MEMORY: 9 MEMORY: 9 UNKNOWN 00 C EX View-1 1A6A 69 1A7A 65 1A8A 6F 1A9A 73 1A9A 73 1A9A 48 1A9A 44 MIN 00011AA	0011444 0011449 0011440 001144C 001144C: 73 70 6 22 20 57 00 73 0 00 73 0 00 73 0 00 50 0 65 6C 60 00 3A 0 A. MEMORI	The second secon	es1; offset eax, esi; 001144C 68 0A 6C 64 65 00 29 57 28 00 World	off_100 c aDosde ebp-1Ch off_100 (Synchrot 00 CC 4 21 0A 6 76 09 6 6C 09 6 6C 90 6 6F 72 6 41 00 3	0329D vicesHe] 0329D hized wi 7 6F 6F 09 CC 5C 9 00 65 C 90 6F C 64 21 B 00 3E	<pre>th EIP) th EIP, t</pre>	79 i. 88 e 86 o 86 s CC H 86 D	<pre>spatch. , . World .s.D.e\.H.e. ello,.W .:.P.(.)</pre>	sDevic .;Good !;\ v.i.c 1.1.o orld! A.;.;	dby 	,Hello , 5 ×	, FEFF	EDI 90506060 EIP 00011433 EFL 00000044 <	??? 1 (S)	MEMOR	5 ×
H H H H H H H H H H H H H H H H H H H	MEMORY: 0 MEMORY: 0 MEMORY: 0 MEMORY: 0 UNENIOWN 0 ex View-1 1868 69 1888 6F 1888 6F 1888 6F 1888 44 1888 44 1888 44 WN 00011AA	9011444 9011440 9011440 901144C 901144C 901144C: 73 70 61 73 70 61 90144C: 90 73 90 90 73 90 90 50 90 90 50 90 90 30 90 81 4000	Call push call MEMORY:0 "" 74 63 6F 72 44 00 48 00 6F 2C 50 00 StaHelloi	es1; offset eax, esi; 001144C 68 0A 6C 64 65 00 65 00 29 57 28 00 World	off_100 : aDosde ebp-1Ch off_100 (Synchrot 21 0A 0 76 09 6 6C 00 6 6C 72 6 41 09 3	0329D vicesHe 1 0329D nized wi 7 6F 6F 0 CC 50 9 00 63 C 00 6F C 64 21 B 00 3E	<pre>th EIP) th EIP th EIP) th EIP th EIP) th EIP th EIP th EIP th EIP th EIP th EIP th EIP) th EIP th</pre>	79 i 00 e 00 o 00 s CC H 00 D	<pre>spatch. , .Worldello,.WP.(.)</pre>	sDevic .;Good !.;\ v.i.c 1.1.o orld! A.;.;	dby 	Hello 5 ×	UNKA	EDI 90506060 EIP 00011433 EFL 00506044 <	??? 1 (S)	ynchro	S ×
H 1001 1001 1001 1001 NKMC 1001 NKMC 1001 NKMC	MEMORY:0 MEMORY:0 MEMORY:0 MEMORY:0 UNENOWN OC ex View-1 1868 69 1888 65 1888 65 1888 65 1888 65 1888 73 1888 44 WN 000112A utput window initial a	8011444 9011449 9011449 9011440 9011440 9011440 9011440 9011440 9011440 9011440 9011440 9011440 9011440 9011440 9011440 9011440 9011440 9011444 9011444 9011444 9011444 9011444 9011444 9011444 9011444 9011444 9011444 9011444 9011444 9011444 9011444 9011444 9011449 9011449 9011449 9011449 9011449 9011449 9011449 9011449 9011449 9011449 9011449 9011449 9011449 9011449 9011449 9011440 901040 90176 90176 90076 90000000000	Call push call MEMORY:0 "" 74 63 74 63 74 63 6F 72 44 00 48 00 6F 2C 50 00 ::affelloi	es1; offset eax, esi; 001144C 68 0A 6C 64 65 00 65 00 65 00 80 28 57 28 00 World	off_100 :aDosde ebp-1Ch (Synchron 00 CC 4 21 0A 0 76 00 6 6F 72 6 41 00 3	03290 vicesHe] 0329D nized wi 7 6F 6F 09 CC 55 09 60 65 C 09 69 C 64 21 B 09 38	<pre>th EIP) th EIP, t</pre>	79 i 80 e 86 o 86 c 86 c 86 d 86 d	<pre>spatch. ,.World. ,.World. ,.World</pre>	sDevic .;Gooo !;\ v.i.c i.c orld! A.;.;	ces\\ dby .D. .e. 	Hello	FEFF	EDI 90060009 EIP 90911433 EFL 90969844 <	**************************************	ynchro	5 ×
H H	MEMORY:0 MEMORY:0 MEMORY:0 MEMORY:0 UNENOWN 00	8011444 9011449 9011440 901144C 901144C: 73 70 61 20 20 57 90 90 73 90 90 50 90 55 6C 60 90 3A 90 A: MEMORY uttoanal	Call push call MEMORY:0 W 74 63 6F 72 9 44 00 48 00 6F 2C 50 09 ::aHelloi	esi; offset eax, [eax esi; cooll44C 68 0A 6C 64 65 00 20 57 28 00 World s been	off_100 :aDosde ebp-1Ch (Synchron 00 CC 4 21 0A 00 76 00 6 6F 72 6 41 00 3	03290 vicesHe] 03290 hized wi 7 6F 6F 00 CC 50 9 00 63 C 00 6F C 64 21 B 00 3E	<pre>th EIP) th EIP, t</pre>	79 i. 86 e 86 o 86 s CC H 86 D	<pre>spatch. , .World .s.D.e\.H.e. ello,.W .:.P.(.)</pre>	sDevic .:Good !.:\\ v.i.c 1.1.o orld! A.;;;	dby .D. .e. 	,Hello		EDI 90060609 EIP 90911433 EFL 90969644 <		ynchro	5 ×
2 H 881 881 881 881 881 881 881 881 881 88	MEMORY:0 MEMORY:0 MEMORY:0 MEMORY:0 UNKNOWN 00	8011444 8011449 8011449 901144C 901144C: 73 70 61 20 20 57 80 73 80 80 50 01 80 50 01 80 50 01 80 30 91 80 40 30 90 80 50 50 50 50 50 50 50 50 50 50 50 50 50	Call push call MEMORY:0 74 63 6F 72 144 00 44 00 48 00 6F 2C 50 00 ::aHellol	esi; offset eax, [eax, eax esi; 001144C 68 0A 6C 64 65 00 20 57 28 00 World S been	off_100 :aDosde ebp-1Ch (Synchron 00 CC 4 21 0A 0 76 00 6 6F 72 6 41 00 3 finished	03290 vicesHe] 03290 hized wi 7 6F 6F 09 CC 50 9 00 63 C 00 6F C 00 6F C 04 21 B 00 3E	<pre>th EIP) th EIP th EIP) th EIP th EIP) th EIP th EIP</pre>	79 i. 86 e 86 o 86 s 6C H 86 D	<pre>spatch. , .World .s.D.e\.H.e. ello,.W .:.P.(.)</pre>	sDevic ! ! ! ! ! 0 ! ! ! ! ! ! ! !	dby .D. .e. .G.	, Hello		EDI 999696999 EIP 99911433 EFL 99969644 <	222 1 (S)	MEMOR ynchroc	5 ×

Sality malware

- > Sality is a polymorphic file infector, targeting Windows executable files
- Communicate over a peer-to-peer (P2P) network to form a botnet
- > Rootkit module
 - Terminate processes via NtTerminateProcess
 - Block access to some anti-virus resources
- Sample available
 - Fae6b86d04f93bcad7736d88beff90f1278c3e3d786a25fc45c9ef2ee375df0f
- Demo: <u>https://youtu.be/ID4cZpWnwkw</u>

Demo: Debug Sality rootkit

fuzz_server.tlp - root@172.16.1.2:22 - Bitvise xterm-256color - root@fuzzvm: ~/aquynh/demigod_repos _ 🗆 × CloseServiceHandle(hSCObject = 0xa0000019) = 0x1 OpenServiceA(hSCManager = 0xa0000018, lpServiceName = "amsint32", dwDesiredAccess = 0xf01ff) = 0xa000001a +] Initiate stack address at 0xfffdd000 +1 Loading samples/bin/windows/system dll/x32/Windows/SysWOW64/drivers/fddfdh.sys to 0x10000 +] PE entry point at 0x1062f +] Driver object addr is 0x6000 +] Registry path addr is 0x60a8 +] EPROCESS is is 0x60b0 +1 KI USER SHARED DATA is 0xffdf0000 Loading samples/bin/windows/system dll/x32/Windows/SysWOW64/ntoskrnl.exe to 0x10000000 Done with loading samples/bin/windows/system dll/x32/Windows/SysWOW64/ntoskrnl.exe +] Loading samples/bin/windows/system dll/x32/Windows/SysWOW64/ntdll.dll to 0x10228000 +] Done with loading samples/bin/windows/system dll/x32/Windows/SysWOW64/ntdll.dll +] Loading samples/bin/windows/system dll/x32/Windows/SysWOW64/kernel32.dll to 0x10369000 +] Done with loading samples/bin/windows/system dll/x32/Windows/SysWOW64/kernel32.dll debugger> Initializing load_address 0x0 TDA - ida60927.idb (<GDB remote process>) C:\Users\john\AppData\Local\Temp\ida60927.idb debugger> Listening on :9999 File Edit Jump Search View Debugger Options Windows Help gdb> Breakpoint added at: 0x1062f Remote GDB debugger 장 🕏 🖬 🗗 😭 💷 🐓 💷 😫 🖓 🐨 🐼 🐼 gdb> Breakpoint: 0x1062f gdb> Resume at: 0x1062f gdb> Breakpoint: 0x10630 Library function Regular function Instruction Data Unexplored External symbol gdb> Resume at: 0x10630 Debug View Structures Enums gdb> Breakpoint: 0x10632 B ×
 General registers IDA View-EIP 0 8 × gdb> Resume at: 0x10632 MEMORY:00010635 mov dword ptr [ebp-10h], (db> Breakpoint: 0x10635 EAX 00000002 * ID 0 dword ptr [ebp-18h], 0 MEMORY:0001063C mov VIP Ø gdb> Resume at: 0x10635 00000000 ECX. MEMORY:00010643 push offset asc_10620 ; " \n" VIFØ gdb> Breakpoint: 0x1063c EDX 00000000 EMORY:00010648 call near ptr unk_10AF0 AC Ø gdb> Resume at: 0x1063c EBX 00000000 VH Ø RF Ø MORY:0001064D add esp. 4 gdb> Breakpoint: 0x10643 MEMORY:00010650 push offset aDeviceAmsint32 ESP FFFFCFEØ NT 0 MEMORY:00010655 push offset unk 10DE8 FFFFCFFC EBP gdb> Resume at: 0x10643 off 10850 MEMORY:0001065A call ESI 00000000 gdb> Breakpoint: 0x10648 MEMORY:00010660 push offset aDosdevicesAmsi ; "\\DosDevices\\amsint32" DF Ø EDI 00000000 gdb> Breakpoint added at: 0x1064d MEMORY:00010665 push offset unk_10E30 IF Ø 4 off 10850 TE O MEMORY:0001066A call gdb> Resume at: 0x10648 MEMORY:00010670 push offset unk 10E28 Modules $DbgPrint(format = ' \n') = 0x2$ MEMORY:00010675 push MEMORY:00010677 push Path MEMORY:00010679 push gdb> Breakpoint: 0x1064d GDB remote process> MEMORY:0001067B push offset unk 10DE8 gdb> Breakpoint removed at: 0x1064d 4 MEMORY:00010680 push MEMORY:00010682 mov eax, [ebp+8] MEMORY:00010685 push eax Threads MEMORY:00010686 call off 10B4C MEMORY:0001068C mov [ebp-0Ch], eax Decimal Hex State MEMORY:0001068F cmp dword ptr [ebp-0Ch], 0 101 65 Ready UNKNOWN 0001064D: MEMORY:0001064D (Synchronized with EIP) -0 8 × Hey View-1 Stack view 0010D60 00 00 00 00 5C 00 44 00 65 00 76 00 69 00 63 00\.D.e.v.i.c. -0010D70 65 00 5C 00 61 00 6D 00 73 00 69 00 6E 00 74 00 e.\.a.m.s.i.n.t. 33 00 32 00 00 00 00 00 5C 00 44 00 6F 00 73 00 3.2. \.D.o.s. 00010090 44 00 65 00 76 00 69 00 63 00 65 00 73 00 5C 00 00010040 61 00 60 00 73 00 69 00 6F 00 74 00 33 00 32 00 D.e.v.i.c.e.s.\. a.m.s.i.n.t.3.2. UNKNOWN 00010DA8: MEMORY:aDosdevicesAmsi+20 +| FFFFFFF; FI (Synchronizer + Output window Flushing buffers, please wait...ok con 11

MacOS

MacOS Overview

- > XNU, hybrid kernel
 - Mach-O x86_64 executable file
 - Kernel Programming Interface (KPI)
 - All implementation code of KPIs are inside
- Kernel extension (KEXT):
 - Born to be a bundle
 - Load in on-demand by kernel
 - Extra attributes come from plist file
- Goal: emulate kernel rootkit in KEXT



Load KEXT

- Load all SEGMENT64s of kernel & KEXT to emulator
- Kernel extension (KEXT)
 - Resolve local relocations
 - Resolve dynamic symbols:
 - Create jmp code section
 - Get symbol address from loaded kernel
 - Retrieve initial methods/functions addresses
 - IOKit: ::start()
 - Generic driver: __*realmain*



Emulating KEXT Initialization

- Initialize kernel environment
 - Setup MAC policy list
 - Allocate new object in emulator
 - Fill the address on loaded *mac_policy_list* symbol on kernel space
 - Create some default processes
 - Allocate new objects in emulator
 - Link them by LIST/SLIST structure
 - Fill the addresses on loaded *allproc* symbol on kernel space
 - Make some vnodes, current credential, ...
- Run emulation for pre-process methods / functions
 - IOKit: *attach()*, *probe()*, ...
 - Generic driver: *kmod_info,* ...
- Emulate from entry point of driver

Instrument KPI

- Map every KPIs to user-defined hook methods
 - Simplify features
 - Use native functions
- Interact with real environment
 - *getattrlistbulks()*: retrieve attributes from a path, then pack the result by calling *vfs_attr_pack()* KPI
- Syscall emulation
 - Get **sysent** from kernel
 - Assign arguments to registers & run corresponding syscall entry address



Nested Emulation

- Call to a native KPI from a hooked KPI
 - Need to save current context to emulate new code path
 - Solution
 - Construct junk code
 - Push address of junk code to stack
 - Return
- The junk code has 3 missions
 - Prepare arguments (registers / stack)
 - Add RSP to clear arguments if necessary
 - Jump to address of that KPI



Emulate Code Path

- User "talks" to driver through callbacks
- > Solution
 - Event Management System (EMS)
 - Hook KPIs to collect callback functions
 - Emulate callbacks by user trigger specific events
- From Asynchronous system to
 Synchronous system
 - Load driver
 - EMS collect callbacks from driver
 - User choose to trigger an event with customzed inputs



Event Management System - SYSCTL

- Hook sysctl_register_oid() to register SYSCTL events
 - Retrieve name of handler from struct
 - Retrieve callback & commit to EMS
- Hook sysctl_unregister_oid() to unregister events from EMS
- Hook sysctl_root() to trigger event from EMS
- User pass input as a *sysctlbyname_args* object to *sysctlbyname()*
 - sysctlbyname() converts input to some structs
 - **sysctl_root()** take structs & trigger event



Event Management System - NKE

- Hook *ctl_register()* to register NKE events
 - Retrieve name of ctl instance from struct
 - Check if callbacks are valid, then commit to EMS
 - Store address of struct to *ctl_ref*
- Hook *ctl_deregister()* to unregister events from EMS
 - Get name from struct stored in *ctl_ref*
- User creates socket object and *mbuf* data to trigger NKE event



Event Management System - Network Filter

- Hook sflt_register() or ipf_addv4() to register NETWORK events
 - Retrieve name from struct
 - Retrieve callback & commit to EMS
- Hook sflt_unregister() to deregister the events from EMS
- User creates a fake source and destination of transition
 - Need to get cookie before create connection
- Use scapy to create packet with custom layers (Ether / IP / TCP) & pack to mbuf_t, then trigger the event.



Event Management System - MAC policy

- Hook *mac_policy_register()* to register
 MAC events
 - Retrieve name from struct
 - Check if callbacks are valid, then commit to EMS
 - Update *mac_policy_list* symbol in kernel space
 - Trigger event *mpo_policy_init* & *mpo_policy_initbsd*
- Hook mac_policy_unregister() to unregister events from EMS
- User triggers some events



Event Management System - KAuth

- Hook kauth_listen_scope() to KAuth events
 - Retrieve identifier from struct
 - Retrieve callback and commit to EMS
- Hook kauth_unlisten_scope() to unregister events from EMS
- User triggers some events
 KAUTH_FILEOP_OPEN,
 KAUTH_FILEOP_CLOSE, ...



Rubilyn Rootkit

- Released on full disclosure in 2012
 - Grant root access to pid
 - Hide files / folders by hooking *getdirentries64()*
 - Hide a process
 - \circ $\hfill Hide an user from 'who' / 'w'$
 - Hide a network port from netstat
 - Sysctl interface for userland control
 - Execute a binary through ICMP ping

Rubilyn Rootkit: new Generation

> Problems

- Failed to load on MacOS High Sierra and later.
- Hide files / folders by hooking *getdirentries64()*
 - "ls" applications now uses getattrlistbulk()
 - Interrupt Descriptor Table (IDT) is located on another page
 - Syscall Table memory is read-only
- Execute a binary through ICMP ping \rightarrow Data structure now is different

> Solution

- Use *saved RIP* and *step back* to find kernel base
- Scan syscall handlers to find *sysent*
- Modify *cr0* register to overwrite entries in *sysent*
- Re-construct ICMP packet based on new structures
- Demo: <u>https://youtu.be/k5vfZUTX3sM</u>

Terminal Shell Edit View Window Help

...

me nt

me

~/Drivers/demo -- bash

mes-Mac:demo me\$ ps aux | grep ./agent

mes-Mac:demo me\$ ps aux | grep ./agent

⊊ Tue 10:48 AM Q 😑

~/Drivers/demo -- bash

0:00.01 ./agent

872 s002 S+ 10:47AM 0:00.01 grep ./age

10:46AM

»» [

me 1325 0.0 0.0 4267736 668 8002 R+ 10:47AM 0:00.00 grep ./age nt mes-Mac:demo me\$ ls agent test_mmap test_pid.c test test_mmap.c mes-Mac:demo me\$ ls test test_mmap.c test.c test_pid mes-Mac:demo me\$ /Applications/Calculator.app/Contents/MacOS/Calculator

1321 0.0 0.0 4267736

1318 0.0 0.0 4267716

demo — -bash — 89×27 --/Drivers/demo — -bash

796 s001 S+

Emulate Rubilyn

- > Load the rootkit and emulate its initial functions from _*realmain* symbol.
- Register sysctl handlers on Event Management System
- > Build some network, process objects, ...
- User pass inputs to trigger code path of rootkit's features
- Demo: <u>https://youtu.be/99QiLnmjBYE</u>

_ipf_addv4(filter = 0xffffff7000013ec0, filter_ref = 0xfffff	$7000015ca0) = 0 \times 0$			
***************************************	kłololokkkololokklolok			
[demigod] Get root for process:				
sysctlbyname(p = 0x500000540, uap = 0x500001788) (PASSTHRU)				
_kalloc_canblock(psize = 0x7ffd099ffed8, canblock = "True", s	ite = 0xffffff8000c9b688) = 0x5000017b8			
_copyio(copy_type = 0x0, user_addr = 0x500001568, kernel_add	= 0x5000017b8, nbytes = 0x11, lencopied = 0	$0x0$, use_kernel_map = $0x0$) = $0x0$		
_bzero(dst0 = 0x7ffd099fff28, length = 0x50) (PASSTHRU)				
_sysctl_root(from_kernel = "False", string_is_canonical = "The sysce of the system of	ue", namestring = 0x5000017b8, namestringle	$n = 0 \times 12$, name = $0 \times 7 \text{ffd} 099 \text{fff} 90$, na	amelen = 0xc, req = 0x7ffd099ffed8) = 0x0	
_sysctl_handle_int(oidp = 0xffffff7000013c90, arg1 = 0xfffff	7000015c90, arg2 = 0x0, req = 0x7ffd099ffed	8) (PASSTHRU)		
_sysctl_io_number(req = 0x7ffd099ffed8, bigValue = 0x0, value	Size = 0x4, pValue = 0xffffff7000015c90, ch	anged = 0×0) (PASSTHRU)		
_sysctl_old_user(req = 0x7ffd099ffed8, p = 0x7ffd099ffe48, l	= 0x4) (PASSTHRU)			
_sysctl_new_user(req = 0x7ffd099ffed8, p = 0x7ffd099ffe48, l	= 0x8) (PASSTHRU)			
_copyio(copy_type = 0x0, user_addr = 0x500001580, kernel_addr	= 0x7ffd099ffe48, nbytes = 0x8, lencopied =	= $0x0$, use_kernel_map = $0x0$) = $0x0$		
_proc_find(pid = 0x539) = 0x500001048				
_lck_mtx_lock(lock = 0x5000010a0) = 0x1				
_kauth_cred_setuidgid(cred = 0x5000014d8, uid = 0x0, gid = 0x	$0) = 0 \times 500001818$			
_lck_mtx_unlock(lock = 0x5000010a0) = 0x1				
_kfree_addr(addr = 0x5000017b8)				
tuanit96@MacBook-Pro-2:~/Projects/demigod <master> x</master>				
<pre>\$ python3 bh_usa_demo/macos/SuperRootkit.pyfile samples/bi</pre>	n/macos/SuperRootkit.kextphide 1337			[12:07
<pre>[+] Parsing kernel: qiling/examples/rootfs/x8664_macos/System</pre>	/Library/Kernels/kernel.development			
[+] Memory for external static symbol is created at 0xffffff	00001b000 with size 0x1000			
[+] Kernel loaded at 0xffffff8000200000				
[+] Found entry point: 0xffffff7000007780				
[+] Found exit point: 0xffffff7000007990				
_printf(format = "[SuperRootKit] Size of proc: %[u"] = 0x0				
_printf(format = "[Superkootkit] Kernel base = 0x%(lx") = 0	XU			
_sysctl_register_oid(oidp = 0xffffff/000013c40)				
_sysctl_register_old(oldp = 0xffffff/000013c90)				
_sysctl_register_old(oldp = 0xffffff/000013ce0)				
_sysctl_register_old(oldp = 0xfffffff000013d30)				
_syscil_register_old(oldp = 0x111111/000013000)				
$ syscic_register_oid(oidp = 0xffffff7000012-20) $				
$syscil_register_old(oldp = 0x1111117000013e20)$				
_systic_register_old(oldp = 0xffffff7000013er0)	7000015cc0) = 0x0			
	(000015Ca0) = 0X0			
[demigod] Hide process:				
(ucet] but $process$.				
_systetuyhame(p = 0x300000340, uap = 0x300001700) (FASSINO)	$i = 0 \times fffffffffffffffffffffffffffffffff$			
$_$ convio(conv type = 0x0 user addr = 0x500001568 kernel addr	= 0x5000017b8 pbytes $= 0x12$ lepconied $=$	AvA use kernel man - AvA) - AvA		
bzero(dst0 = 0x7ffd000fff28]enoth = 0x50) (PASSTHRII)	= 0x3000017b0, hbytes = 0x12, tencopied = 0	0x0; use_kernet_map = 0x07 = 0x0		
	ue" namestring = 0x5000017b8 namestringle	n = 0x13 name = 0x7ffd099fff90 na	amelen = $0xc$ reg = $0x7ffd099ffed8$) = $0x0$	
sysctl handle int(oidp = Axffffff7000013ce0, arg1 = Axffffff	7000015c94 arg = $0x900001750$, numescrift	8) (PASSTHRII)		
sysctl in number(reg = $0x7ffd099ffed8$, bigValue = $0x0$, value	Size = $0x4$, $nValue = 0xffffff7000015c94$, ch	anged = 0×0 (PASSTHRU)		
<pre>sysctl old user(reg = 0x7ffd099ffed8, p = 0x7ffd099ffe48, 1</pre>	$= 0 \times 4$ (PASSTHRU)	angea shof (Theornito)		
sysctl new user(reg = $0x7ffd099ffed8$, p = $0x7ffd099ffe48$, l	$= 0 \times 8$) (PASSTHRU)			
copyio(copy type = 0x0, user addr = 0x500001580, kernel add	= 0x7ffd099ffe48, nbytes = 0x8, lencopied	= $0x0$, use kernel map = $0x0$) = $0x0$		
proc list lock()				
proc list unlock(
_kfree_addr(addr = 0x5000017b8)				
tuanit96@MacBook-Pro-2:~/Projects/demigod <master> x</master>				
\$				[12:07
	6	∺ <mark>e</mark> 0.0 kB↓	0.0 kB↑ 2 master + •	

Analyze Rubilyn: Debug

- Enable gdb server on emulator
 - Use --gdb ":<port>" option
- Connect gdb remotely to gdb server
 - target remote <address>:<port>
- Debug Rubilyn with gdb
 - Set breakpoints: b *<address>
 - Step next, step inside: *si, ni, ...*
 - Continue: **c**
- Demo: <u>https://youtu.be/vSIjF9IG6Ck</u>

× python3 (Python)			gdb (ssh)					
[+] 0xffffff7000007d84 0f 86 44 00 00 00	jbe 0xffffff7000007dce							
[+] 0xffffff7000007d8a 48 8b 45 d8	mov rax, qword ptr [rbp – 0x28]		0xffffff7000007780	push	%rbp			
[+] 0xffffff7000007d8e 48 05 08 00 00 00	add rax, 8		0xffffff7000007781	mov	%rsp,%rbp			
[+] 0xffffff7000007d94 48 89 45 c8	mov qword ptr [rbp – 0x38], rax		0xffffff7000007784	sub	\$0x40,%rsp			
[+] 0xffffff7000007d98 48 8b 7d c8	mov rdi, qword ptr [rbp - 0x38]		0xffffff7000007788	mov	%rdi,-0x10(%rbp)			
[+] 0xffffff7000007d9c 48 8d 35 ea 51 00 00	lea rsi, [rip + 0x51ea]		0xffffff700000778c	mov	%rsi,-0x18(%rbp)			
[+] 0xffffff7000007da3 ba 07 00 00 00	mov edx, 7		0xffffff7000007790	mov	%rbp,%rax			
[+] 0xffffff7000007da e8 cf 33 01 00	call 0xffffff700001b17c		0xffffff7000007793	mov	%rax,-0x20(%rbp)			
[+] 0xffffff700001b17c 48 83 ec 08	sub rsp, 8		0xffffff7000007797	movabs	\$0xffffff8000200000,%rax			
[+] 0xffffff700001b180 c7 04 24 f0 cf 48 00	mov dword ptr [rsp], 0x48cff0		0xffffff70000077a1	mov	%rax,-0x28(%rbp)			
[+] 0xffffff700001b187 c7 44 24 04 80 ff ff ff	mov dword ptr [rsp + 4], 0xffffff80		0xffffff70000077a5	lea	0x5798(%rip),%rdi	# 0xffffff700000cf44		
[+] 0xffffff700001b18f c3	ret		0xffffff70000077ac	mov	\$0x3e0,%esi			
[+] 0xffffff800048cff0 55	push rbp		0xffffff70000077b1	mov	\$0x0,%al			
$_strncmp(s1 = "demigod", s2 = "demigod", n = 0x7) ($	PASSTHRU)	B+	> 0xffffff70000077b3	callq	0xffffff700001b140			
[+] 0xffffff800048cff1 48 89 e5	mov rbp, rsp		0xffffff70000077b8	mov	-0x28(%rbp),%rsi			
[+] 0xffffff800048cff4 31 c0	xor eax, eax		0xffffff70000077bc	lea	0x57a4(%rip),%rdi	# 0xffffff700000cf67		
[+] 0xffffff800048cff6 48 85 d2	test rdx, rdx		0xffffff70000077c3	mov	%eax,-0x34(%rbp)			
[+] 0xffffff800048cff9 75 13	ine 0xffffff800048d00e		0xffffff70000077c6	mov	\$0x0.%al			
[+] 0xffffff800048d00e 0f be 0f	movsx ecx. byte ptr [rdi]		0xffffff70000077c8	calla	0xffffff700001b140			
[+] 0xffffff800048d011 44 0f be 06	movsx r8d, byte ptr [rsi]		0xffffff70000077cd	cmpa	\$0x00x28(%rbp)			
[+] 0xffffff800048d015 44 38 c1	cmp cl. r8b		0xffffff70000077d2	ie	0xffffff700000793e			
[+] 0xffffff800048d018 75 06	ine 0xffffff800048d020		0xffffff70000077d8	movabs	\$0xffffff800084b3a0.%rax			
[+] 0xffffff800048d01a 84 c9	test cl. cl		0xffffff70000077e2	mov	%rax.0xdc57(%rip)	# 0xffffff7000015440		
[+] 0xffffff800048d01c 75 e2	ine 0xffffff800048d000		0xffffff70000077e9	movabs	\$0xffffff800087cbf0.%rax			
[+] 0xffffff800048d000 48 ff ca	dec rdx		0xffffff70000077f3	mov	%rax.0xdc4e(%rip)	# 0xffffff7000015448		
[+] 0xffffff800048d003 48 ff c7	inc rdi		0xffffff70000077fa	movabs	\$0xffffff800087cc10.%rax			
[+] 0xffffff800048d006 48 ff c6	inc rsi		0xffffff7000007804	mov	%rax.0xdc45(%rip)	# 0xffffff7000015450		
[+] 0xffffff800048d009 48 85 d2	test rdx. rdx		0xffffff700000780b	movabs	\$0xffffff8000e5ff70.%rax			
[+] 0xffffff800048d00c 74 10	je 0xffffff800048d01e		0xffffff7000007815	mov	%rax.0xdc3c(%rip)	# 0xffffff7000015458		
[+] 0xffffff800048d00e 0f be 0f	movsx ecx. byte ptr [rdi]		0xffffff700000781c	movabs	\$0xffffff800033f200.%rax			
[+] 0xffffff800048d011 44 0f be 06	movsx r8d, byte ptr [rsi]		0xffffff7000007826	mov	%rax.0xdc33(%rip)	# 0xffffff7000015460		
[+] 0xffffff800048d015 44 38 c1	cmp cl. r8b		0xffffff700000782d	movabs	\$0xffffff8000ecf920.%rax			
[+] 0xffffff800048d018 75 06	ine 0xffffff800048d020		0xffffff7000007837	mov	%rax.0xdc2a(%rip)	# 0xffffff7000015468		
[+] 0xffffff800048d01a 84 c9	test cl. cl							
[+] 0xffffff800048d01c 75 e2	ine 0xffffff800048d000	re	mote Thread 42000.1996 In	1:			L??	PC: 0xffffff70000077b3
[+] 0xffffff800048d000 48 ff ca	dec rdx	(a	db) si					
[+] 0xffffff800048d003 48 ff c7	inc rdi	0×	ffffff7000007781 in ?? ()					
[+] 0xffffff800048d006 48 ff c6	inc rsi	(a	db) si					
[+] 0xffffff800048d009 48 85 d2	test rdx. rdx	θx	fffffff7000007784 in ?? ()					
[+] 0xffffff800048d00c 74 10	je 0xffffff800048d01e	(a	db) si					
[+] 0xffffff800048d00e 0f be 0f	movsx ecx, byte ptr [rdi]	θx	() () () () () () () () () () () () () (
[+] 0xffffff800048d011 44 0f be 06	movsx r8d, byte ptr [rsi]	(g	db) si					
[+] 0xffffff800048d015 44 38 c1	cmp cl, r8b	Θx	fffffff700000778c in ?? ()					
[+] 0xffffff800048d018 75 06	jne 0xffffff800048d020	(g	db) b *0xffffff70000077b3					
[+] 0xffffff800048d01a 84 c9	test cl, cl	Br	eakpoint 1 at 0xfffffff00	000077b3				
[+] 0xffffff800048d01c 75 e2	jne 0xffffff800048d000	_ (q	db) c					
[+] 0xffffff800048d000 48 ff ca	dec rdx	Co	ontinuing.					
[+] 0xffffff800048d003 48 ff c7	inc rdi							
[+] 0xffffff800048d006 48 ff c6	inc rsi	Br	eakpoint 1, 0xffffff70000	077b3 j	n ?? ()			
[+] 0xffffff800048d009 48 85 d2	test rdx, rdx	(0	db)					
[+] 0xffffff800048d00c 74 10	je 0xffffff800048d01e							
[+] 0xffffff800048d00e 0f be 0f	movsx ecx, byte ptr [rdi]							

□ 7% ____

— = 0.0 kB↓

0.0 kB↑ 🕴 master + •

Linux

Linux Overview

> Linux kernel

- System calls play key roles to provide services
- Also interface with userland via IOCTL
- Various callbacks indirectly triggered by userland
 - File operations, etc
- Kernel module in LKM: .KO
 - ELF format, but all in sections
 - No program headers
- Goal: emulate kernel rootkit in .KO driver



Load LKM File (.KO)

- ELF loader for .KO file
 - Parse all sections of ELF file and map all sections into emulator
 - Resolve external API to provide our own implementation.
 - Relocate external functions and symbols (data)
- Setup CPU context
 - GS segment (current_task)
- Initialize Linux syscall table
 - Write address of our own syscall implementation into syscall table
 - Hook syscalls to execute our syscall implementation
- Locate initialization function
 - **init_module** symbol in SYMTAB section

Linux Internal Structures

- ➤ task_struct
 - Information for userland tasks.
- ➤ file_operations
 - Callback to file operations (read/write/open/close/etc)
- ≻ module
 - Information about LKM, with links to all modules
- ➤ user_namespace
 - Access to user credentials
- ➤ file_struct
- linux_direntXX
 - Directory information

Emulating LKM Initialization

- Setup optional arguments for init_module entry
 - Ignored when LKM is loaded without arguments
 - Similar to input arguments for **sys_init_module**
 - Buffer point to LKM arguments & total length
- Emulate from init_module
 - Stop emulation when reaching RET

Hook Linux Kernel APIs

- Emulate some APIs necessary for LKM to work
 - __fentry__
 - __x86_indirect_thunk_rax
 - ____stack_chk_fail
 - _copy_to_user, _copy_from_user
 - kmalloc, kfree
 - Device management
 - misc_register, misc_deregister
 - register_chrdev
 - device_create

• Syscalls

- sys_read, sys_write, sys_open
- prepare_creds, commit_creds
- ∎ etc

Emulate Code Path

- Extract syscalls or callbacks from LKM loading phase
- Pass input arguments provided by users via registers
 - SYSV ABI
- Userspace communicates with LKM via syscalls or IOCTL
 - READ, WRITE, OPEN syscalls
 - IOCTL syscalls
- Other special interfaces
 - proc_file_fops for /proc access
 - Emulate .read, .write
 - **nf_hook_ops** for netfilter access (monitor network packets).
 - Emulate .hook
 - **notifier_block** to access to keystrokes (keylogger)
 - Emulate .notifier_call

Demo: Linux mOhamed Rootkit

- Load in as LKM (rootkit.ko)
- > A proof-of-concept kernel rootkit, with typical rootkit behaviors
 - Hide kernel modules (and itself)
 - Hide files
 - Hide network ports
 - Grant root access to current process (backdoor)
 - Overwrite syscalls: READ, WRITE, OPEN, CLOSE
 - Interface with userland customized file in /proc
- Code available
 - <u>https://github.com/m0hamed/lkm-rootkit</u>
- ≻ Demo
 - Load rootkit & initialize it
 - Retrieve address of rootkit's WRITE syscall
 - Emulate rootkit WRITE syscall directly, with selected input
 - Trace execution with debugger of IDA

Demo: Debug mOhamed Rootkit

×1 Fi	le Edit Selection View	Go Run Terminal Help		DDA - rootkit.ko C:	⊠ <
and a	A colibumu A dab	den-land eu Y		File Edit Jung Search View Debugger Options Windows Help	
C1	e santy.py e ded	onnoad.py	 conclus 	_ ▶ □ □ Renote 606 debugger 上 16 君 [函 弊 ¥] 章 単 爭 章 ∃ Э ♀ □ [函 函] ⊠ 46 ↔	
	@ dm-load.py 2				9
	1 #!/usr/bin/env	v ovthon3		Ubrary function 📕 Regular function 📕 Instruction 🐘 Data 📕 Unexplored External symbol	
				Debug Wew 🖸 🖪 Structures 🖸 🖽 Enums 💟	
80				🔯 IDA View-RIP 🗆 🗗 🖉 orggruu.gdb.1086.core 🗖 🗗 🛪	×
	4			Loss:00000000000DFD loc_1DFD: ; CODE XREF: .bss:00000000000DE21j 🛋 RAX 0000000000000 🛋 10 0	
A	5 import struct, 6 sys.nath.anner	, sys		bs::00000000000000000000000000000000000	-
				-bss:00000000000000000000000000000000000	
	8 from giling in	sport *		bs:reeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee	2
40	9 from qiling.os	s.windows.wdk_const import		bss:00000000000100E mov rax, cs:off_27F0	ĸ
				bs: foodedddddddaf215 mou Pck, 220m bs: foodeddddddaf2151 mou Pck, 220m	
B	12 from ailing.os	s.linux.syscall nums impor		bss:00000000000121 mov rdi, 2109h	ad 1
				bss:0000000000001E28 mov cs:off_27C8, rdx	-
				bss:0000000000000132 mov rdx, [rax10h]	×
	15 if not len(sys	s.argv) in (2, 3):	utaut ideal di sasa [dafai]†1" X. sus. anai [0]	bss:00000000001E3E mov cs:off_2700, rdx	
	17 svs.exit(-	-1)	ochocrombiorranioeranici w shrranRa[o]	bs::0000000000001E45 xor edx, edx	15
	PROFILEMS 16 OUTPUT	DEBUG CONSOLE TERMINAL		bs::00000000000164E call near ptr unk_100000	<
				- b5::00000000000000000000000000000000000	
	[+] 0x1d4a	48 c7 c6 d8 1c 08 00	mov rsi, 0x1cd0	bs::0000000000001E58 mov rdi, 21FBh	
	[+] 0x1d51 [+] 0x1d58	48 89 35 79 8a 88 88	mov qword ptr [rip + 8xa79], rsi	III) ISSI GOOGGOOGGOOGGIES CAIL REAF PT UNK LOUGOO	
	[+] 0x1d5f	48 89 70 88	mov qword ptr [rax + 8], rsi	bs::000000000000166 pop rbp	
	[+] 0x1d63	e8 b8 e2 ff 08	call 0x1000020	ISS. COCOCOCOCOCOCOCICO TELI	
	[+] 0x1000020	00 00	add byte ptr [rax], al	UMRNOVM (0000000000EED9: bssloc_IE09 (Synchronized with RIP)	
	[+] 0x1d68	Sd	pop rbp		
	[+] 8x1db9 [+] 8x1e8e	c3 48.8h.85.db.89.88.88	ret mov rax, gword ntr [rin + 0x9db]		
	[+] 0x1e15	48 c7 c1 28 22 08 00	mov rcx, 8x2228	D0000000000000000 0 E 2 33 13 00 00 48 85 55 F4 12 00 00 48 85 FE FF	
	[+] Exteic	be b6 01 00 00	mov esi, 0x1b6	000000000000000000000 75 0C EB 2E 48 83 C6 08 48 83 FE FF 74 24 48 81 u	
	[+] 0x1e21 [+] 0x1e28	48 C7 C7 09 Z1 00 00 48 8b 10	mov rdx, gword ptr [rax]	00000000 00000000000000000000000000000	4
	[+] 8x1e2b	48 89 15 96 09 00 00	mov qword ptr [rip + 0x996], rdx		
	[+] 0x1e32	48 8b 50 10	mov rdx, qword ptr [rax + 0x10]	Telepater hardwer is stortcut key is Ltri-Alt-k	
	[+] 0x1e30	48 89 15 85 89 88 88	mov qword ptr [rip + 0x98b], rdx	Use the same hotkey Ctrl-Alt-K to open 'Fill Range' window on a selected range of code To rewerk (undo the last notching, choose more Edit Kwnatch lundo last patching	Ĩ 1
	[+] 0x1e45	31 d2	xor edx, edx	Keypatch Search is available from menu Edit Keypatch Search	
	[+] 0x1e47 [+] 0x1e4e	48 c7 00 98 10 00 00 e8 7d e2 ff 09	mov qword ptr [rax], 8x1898	Find more information about Keypatch at http://keystone-engine.org/keypatch	
	proc_create() = 0x1000				
	[+] 8x18888d8	66 66	add byte ptr [rax], al	Python 2.7.13 (default, Jun 26 2017, 14:28:43) (MSC v.1500 64 bit (AMD64)) IDAPython 64-bit v1.7.0 final (serial 0) (c) The IDAPython Team (idapython@coople.com)	
	[+] 0x1e53 [+] 0x1e56	48 85 68	test rax, rax ie 0x1e68		
	[+] 8x1e58	48 c7 c7 fb 21 00 00	mov rdi, 0x21fb	Propagating type information Function argument information has been propagated	
	[+] 0x1e5f	e8 bc e1 ff 00	call 0x1000020	The initial automalyzis has been finished.	
Q	[+] 8x1000020	ee ee	add byte ptr [rax], al	Comman "Processition" failed	
~	[+] 0x1e64	31 60	xor eax, eax	1008: unknown festure 'org.gnu.gdb.1386.11nuz' (was not present in the template) Debugeer statched to process (0140/201492794)	1.
	[+] 0x1e66	Sd	pop rbp	Flushing buffers, please waitok	
				CCE	
× SSH:	172.16.1.2 🦻 master* 🖸	Python 3.6.9 64-bit ['.venv': virtualer	m/) ⊗ 0 ≜ 13 () 23	AU: ide Down Disk: 1168	<i>"</i> 2

More Applications of Demigod

- > Debug kernel rootkits in ring 3, within safety sandbox
 - IDA Pro, GDB client, Binary Ninja etc
- Auto-unpacker for kernel code
 - Trace kernel code to detect the time code is unpacked, then dump & auto rebuild binary
- IOCTL analyzer
 - Auto-discover IOCTL code & callbacks using random inputs
- > Auto-summarize rootkit behaviors to produce high-level reports
 - Auto-discover different rootkit code paths, and summarize its behavior according to rules
 - Demigod enable save execution on checkpoints, taking snapshots, etc
- Some other exciting private ideas ;-)

Status & Future work

- Supported Windows, MacOS & Linux kernel rootkits
- Still need to test & improve to support more rootkits
 - More kernel API
 - More syscalls
 - More kernel subsystems
 - Emulating enough features of real OS is hard & long-term project
- Support more platforms?
 - iOS, Android, etc
- > To be merged into Qiling emulator soon
 - Watch out **https://groundx.io/demigod** for more announcement

Conclusions

- > Demigod is a framework to emulate kernel rootkits
 - Enable analyzing ring 0 code from ring 3, in safety sandbox
 - Monitor, trace, debug, etc
 - Built on top of the excellent **Qiling.io** emulator
 - Cross-platforms: Windows, MacOS & Linux
 - Cross-architectures: X86, Arm, Arm64, Mips
- Enable advanced binary analysis of kernel code
 - Python-based framework for tool building
 - Instrumentation at various levels
- Release to be announced on groundx.io/demigod
 - To be merged into **Qiling.io** very soon

Q&A

Demigod: The Art of Emulating Kernel Rootkits

NGUYEN Anh Quynh <aquynh @ gmail.com>

NGUYEN Hong Quang <quangnh89 @ gmail.com>

DO Minh Tuan <tuanit96 @ gmail.com>