Shepherding Loadable Kernel Modules through On-demand Emulation

Chaoting Xuan, John Copeland, Raheem Beyah
Georgia Institute of Technology

7/9/2009

Presented by John McHugh

Kernel Rootkits

- a program runs in kernel space to take fundamental control of a computer system, without authorization by the system's owners and legitimate.
- Joanna Rustkowska classified three types of kernel rootkits.

1) Type I: modifying static kernel objects.
2) Type II: modifying dynamic kernel objects.
3) Type III: VMM, SMM and other hardware-based rootkits.

How are malicious code inserted to the kernel ?

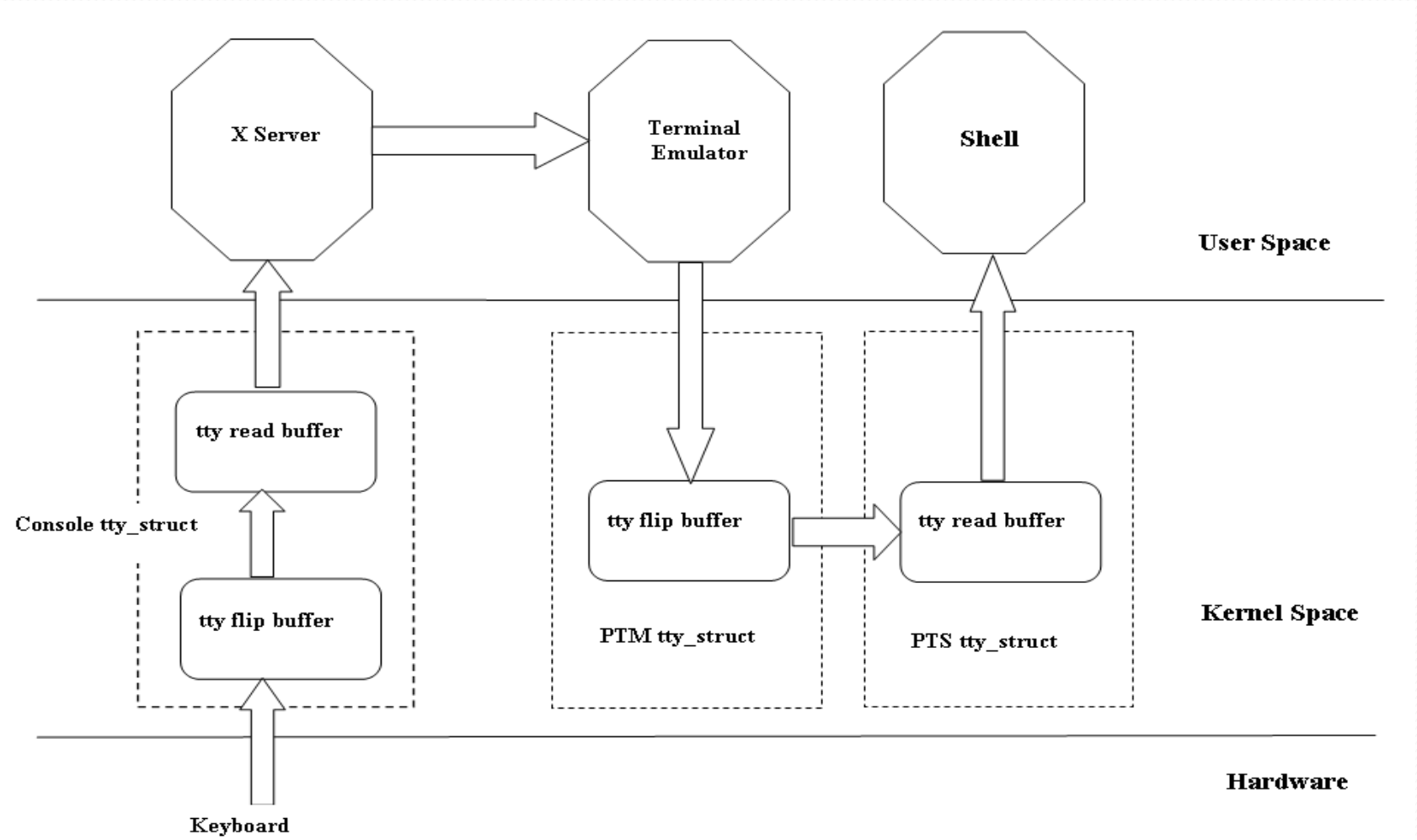
- Loadable kernel module (LKM) or driver (most of Linux and Windows rootkits are implemented in this way).
- /dev/mem, /dev/kmem and \Device\PhysicalMemory (But modifying kernel memory through them has been disabled in recent Linux and Windows versions).
- Exploiting the vulnerabilities of benign kernel code (like buffer overflows).

Rootkit Detection

- Signature-based rootkit detection and kernel integrity verification are two generic rootkit detection approaches, and many research works have been focused the second one.
- Kernel integrity verification has following limitations:

1) It is infeasible to verify all the states of dynamic kernel objects.
2) It cannot cope with non-integrity-violation rootkits, e.g., some type III rootkits and confidentiality-violation rootkits.

A Confidentiality-violation Rootkit.



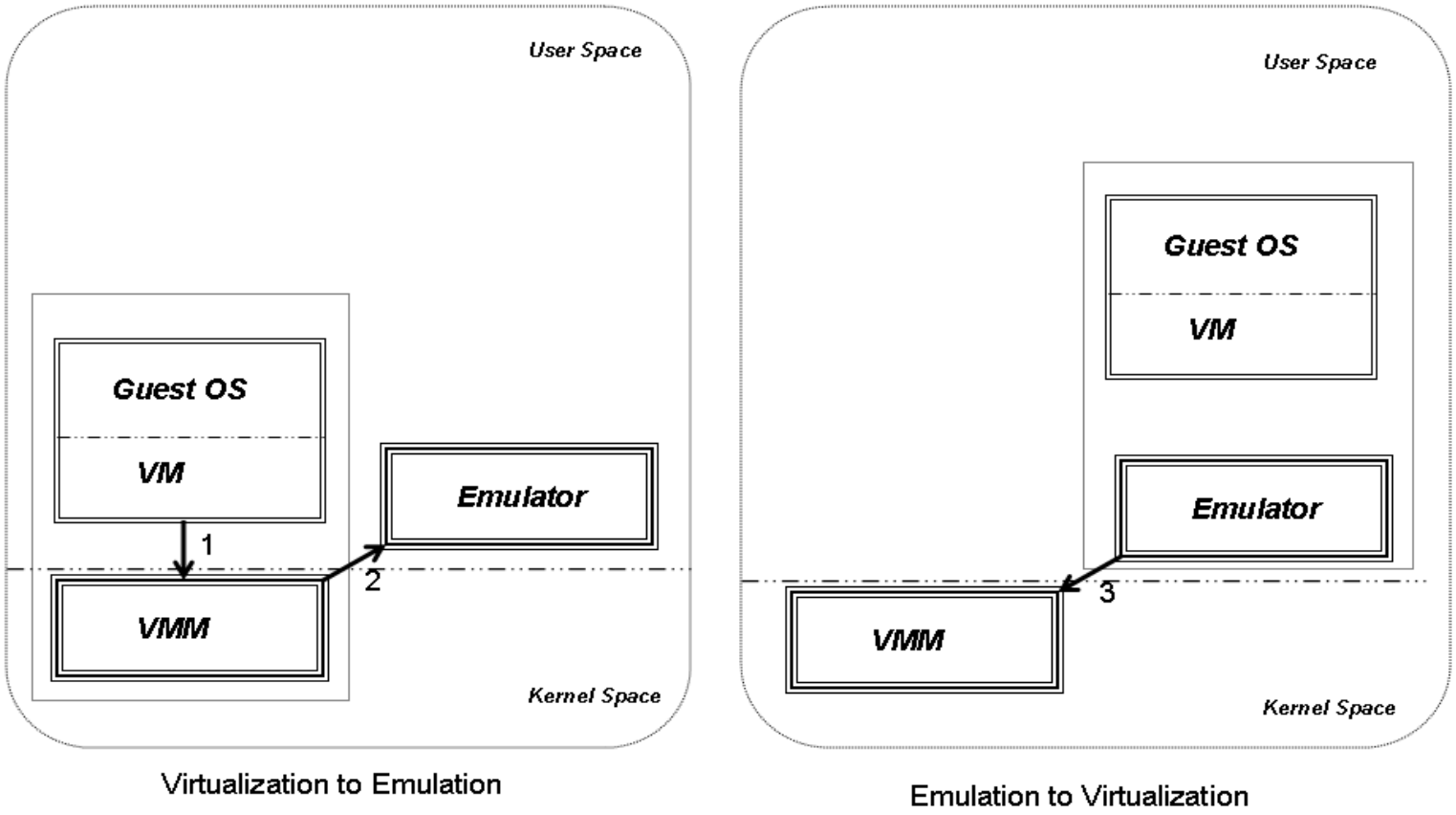
Rootkit Prevention

- Only authenticated/authorized LKMs/drivers are allowed to be loaded and executed in the kernel (Microsoft's driver code signing, NICKLE and SecVisor).
- However, no authentication/authorization authority can assure the goodness of a driver/LKM. VeriSign just sells the certificate to customers and doesn't check the driver code to be signed at all.
- This approach also cannot handle a large number of legacy drivers/LKMs, Windows XP and 32-bit Vista doesn't enforce the code driver signing in default.

DARK

- DARK aims to assure the goodness of a LKM/driver that computer users don't trust, while it is useful to them.
- It combines the program monitoring with the rootkit identification techniques. The suspicious kernel code is monitored and its interactions with the rest of kernel are checked against a group of well-selected security policies.
- DARK is built based on Qemu/Kqemu and its program monitoring is achieved through on-demand emulation.

On-demand Emulation



Virtualization to Emulation

Emulation to Virtualization

Figure 1: DARK Operation Mode Switch

Design

- Use one feature of OS's virtual memory management subsystem.
- Set present bits in page table entries of the LKM's code to 0.
- Page fault exception is generated when the VM attempts to run the LKM code.
- The VMM intercepts and interprets the exception and switches to emulation mode.
- The emulator sets those present bits to 1 and runs the LKM code along with DARK's security policy checking.

Implementation

- Reuse the existing demand emulation framework of Qemu/Kqemu.
- Instrument Linux kernel's *sys_init_module* and *sys_delete_module* functions in kernel/module.c , and the VM issues the 0x90 and 0x91 software interrupt to VMM.
- Add DARK's business logics to V-2-E and E-2-V control code of Qemu/Kqemu.
- Modify Linux module loader (insmod.c) to store a module's text range in its module descriptor.

Security Policy

- They are manually created based on expert knowledge.
- The basic rule format is as below:

Subject	Operation	Object	Action
Module X	Read, Write, Call	Hardware Objects, Kernel Objects	Alarm, Reject

ID	NAME	OPERATION	GLOBAL VARIABLE OR FUNCTION	DATA TYPE	ACTION	DYNAMIC	OPTIONAL
1	Console TTY Buffer	Read	console_table	tty_struct	Alarm	No	No
2	Exception Table	Write	__start___ex_table	Exception_table_entry	Alarm	No	No
3	GDT table	Write	gdt_table	Array	Reject	No	No
4	IDT table	Write	idt_table	Array	Reject	No	No
5	Kernel Text	Write	_text	-	Reject	No	No
6	MM List	Write	init_task	mm_struct	Alarm	Yes	No
7	Module List	Write	module_list	Module	Alarm	Yes	No
8	Module Text	Write	module_list	-	Reject	Yes	No
9	Netfilter Hooks	Call	nf_register_hook	-	Alarm	No	Yes
10	Page Table	Write	init_task	-	Reject	Yes	No
11	Proc Dir Entry List	Write	proc_root	proc_dir_entry	Alarm	Yes	No
12	Proc Inode Ops List	Write	proc_root	Proc_inode_operation	Alarm	Yes	No
13	Proc File Ops List	Write	proc_root	Proc_file_operation	Alarm	Yes	No
14	PTM TTY Buffer	Read	ptm_table	tty_struct	Alarm	Yes	No
15	PTS TTY Buffer	Read	pts_table	tty_struct	Alarm	Yes	No
16	Socket Buffer List	Read	skbuff_head_cache	sk_buff	Alarm	Yes	Yes
17	Syscall Table	Write	sys_call_table	Array	Reject	No	No
18	Task List	Write	init_task	task_struct	Alarm	Yes	No
19	Segment	Write	init_tss	Array	Reject	No	No

DARK System Rules

ID	NAME	OPERATION	HARDWARE OBJECT	ACTION	INSTRUCTIONS
1	BIOS	Write	BIOS ROM	Reject	MOV
2	System Cache	Write	L1, L2 Cache	Alarm	INVD, WBINVD
3	Control Register	Write	CR0, CR3, CR4	Reject	MOV CRn
4	Debug Register	Write	DB1 – DB7	Alarm	MOV DBn
5	IO Port	Read/Write	IO Ports	Alarm	IN, OUT
6	IDT Register	Write	idt register	Reject	LIDT
7	GDT Register	Write	GDT Register	Reject	LGDT
8	MSR	Write	MSR	Alarm	WRMSR
9	System RAM	Write	System RAM	Alarm	MOVE
10	TLB	Write	TLB	Alarm	INVLPG
11	TR Register	Write	TR Register	Reject	LTR

Policy Enforcement

- Kernel rules are organized as two hash tables.
- The LKM code is monitored at the code translation of Qemu, and the cached code is not inspected again, which significantly improves the performance.
- External kernel memory accesses and function invocations can optionally be logged to a local file.

Security Evaluation

- Total 18 Linux rootkits are available for the evaluation: 17 are collected from the Internet; one is written by myself.
- We can't find any type III rootkit that works in the testing system.
- 20 benign modules are selected from Linux source for false positive evaluation.
- All rootkits have been detected and blocked by DARK; only 1 benign module causes an alarm.

ROOTKIT	FUNCTION					TYPE	HIT KERNEL RULE		ACTION
	HID	PE	REE	REC	NEU		Load	Operation	
Adore	X	X				I	17	18	Reject
Adore-ng	X	X			X	II	7, 12, 13	18	Alarm
Adore-ng (hidden)	X	X			X	II	7*, 12, 13	18	Alarm
Darklogger				X		II		15	Alarm
Exception		X			X	I	2	18	Reject
Fileh-lkm	X					I	17		Reject
Hookstub		X				I	4	18	Reject
Hp	X	X				II	18		Alarm
KIS	X		X			I	17		Reject
Knark	X	X	X			I	17	18	Reject
Linspy2				X		I	16		Reject
Nfsniffer				X		II	9	16	Alarm
Nushu					X	II		16	Alarm
Pizzaicmp			X			II	9	16	Alarm
Prrf	X	X				II	11, 12, 13	18	Alarm
Sebek				X		I	7, 17		Reject
Srootkit	X					I	5		Reject
Vlogger				X		I	17	14	Reject
Vlogger (local)				X		II		1	Alarm

Performance Evaluation

- Target LKM is *iptable_filter*.
- Three systems: VMM-only, DARK and DARK-CS.
- Three benchmarks: bonnie, iperf, lmbench.
- When *iptable_filter* is excuted, DARK has about 10% performance penalty.

Bonnie Test Result

	Sequential Output						Sequential Input				Random	
	Per Char		Block		Rewrite		Per Char		Block		Seeks	
	K/sec	%CPU	K/sec	%CPU	K/sec	%CPU	K/sec	%CPU	K/sec	%CPU	/sec	%CPU
VMM	8528±233	64±3	12755±1425	45±5	19082±1490	53.0±3	15805±2301	75±4	129292	71±2	3515±1908	84±4
DARK-CS	8038±345	61±5	11715 1379	41±6	17402±1834	48.2±2	16860±2004	80±5	130266	74±4	4969±1759	85±2
DARK	8168±405	67 6	13949±1106	43±5	18742±2046	49.8±2	14480±2720	73±7	125493	72±4	5117±1254	83±4

Iperf Test Result

	VM as Server (M/sec)		VM as Client (M/sec)	
	TCP	**UDP**	**TCP**	**UDP**
VMM-only	21.8±1.2	1.05±0.1	26.8±2.3	1.13±0
DARK-CS	19.73±0.5	1.01±0	23.99±1.4	1.08±0.1
DARK	19.60±0.6	1.00±0.1	24.05±1.0	1.08±0.1

Future Work

- Continually refine the policy rules to accommodate new rootkit attacks.
- Tweak the tradeoff between performance and security: selectively enabling or disabling on-demand emulation based on certain conditions, e.g., current system load, virtual CPU usage and bandwidth usage.
- Port DARK to Kernel-based Virtual Machine (KVM) that supports hardware virtualization extensions such as Intel's VT-x and AMD's AMD-V.

Questions ?