Lecture 3: Isolation Subversion, Rootkits, and Intrusion

Sandeep K. Shukla

Indian Institute of Technology Kanpur

Acknowledgements

- Dan Boneh (Stanford University)
- John C. Mitchell (Stanford University)
- Nicolai Zeldovich (MIT)
- Jungmin Park (Virginia Tech)
- Patrick Schaumont (Virginia Tech)
- C. Edward Chow
- Arun Hodigere
- Web Resources

Lecture 3: Subverting Isolation, Rootkits, and Intrusion

- Module 3.1: VM based Isolation
  - Module 3.1.1 Subversion
- Module 3.2: Confinement Principle
- Module 3.3: Software Fault Isolation
- Module 3.4: Rootkits
  - Module 3.4.1 – Rootkit Basics
  - Module 3.4.2 – Rootkit Types
  - Module 3.4.3 – Rootkit Detection
- Module 3.5: IDS – Intrusion Detection Systems
  - Module 3.5.1 – Commercial IDS

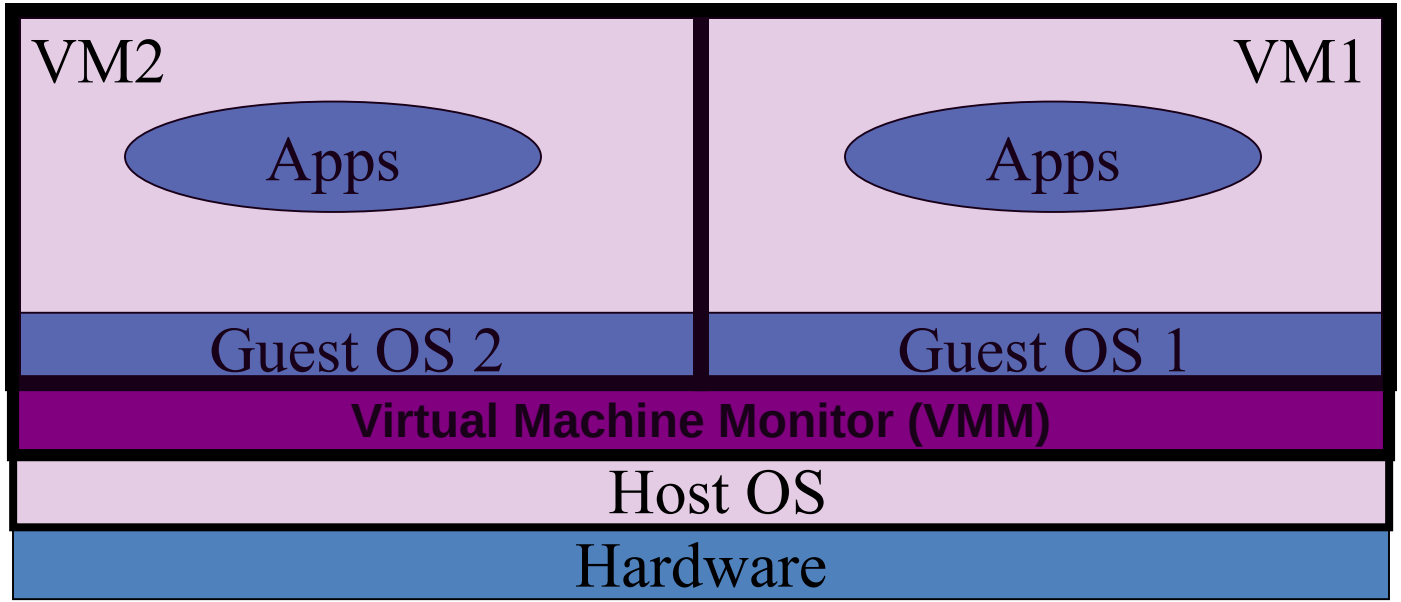
Module 3.1

Isolation via Virtual Machines & Subversion

Isolation

Isolation via
Virtual Machines

Virtual Machines



Example: **NSA NetTop**

single HW platform used for both classified and unclassified data

Why so popular now?

VMs in the 1960's:

- Few computers, lots of users
- VMs allow many users to shares a single computer

VMs 1970's – 2000: non-existent

VMs since 2000:

- Too many computers, too few users
  - Print server, Mail server, Web server, File server, Database , …
- Wasteful to run each service on different hardware
- More generally: VMs heavily used in cloud computing

VMM security assumption

VMM Security assumption:

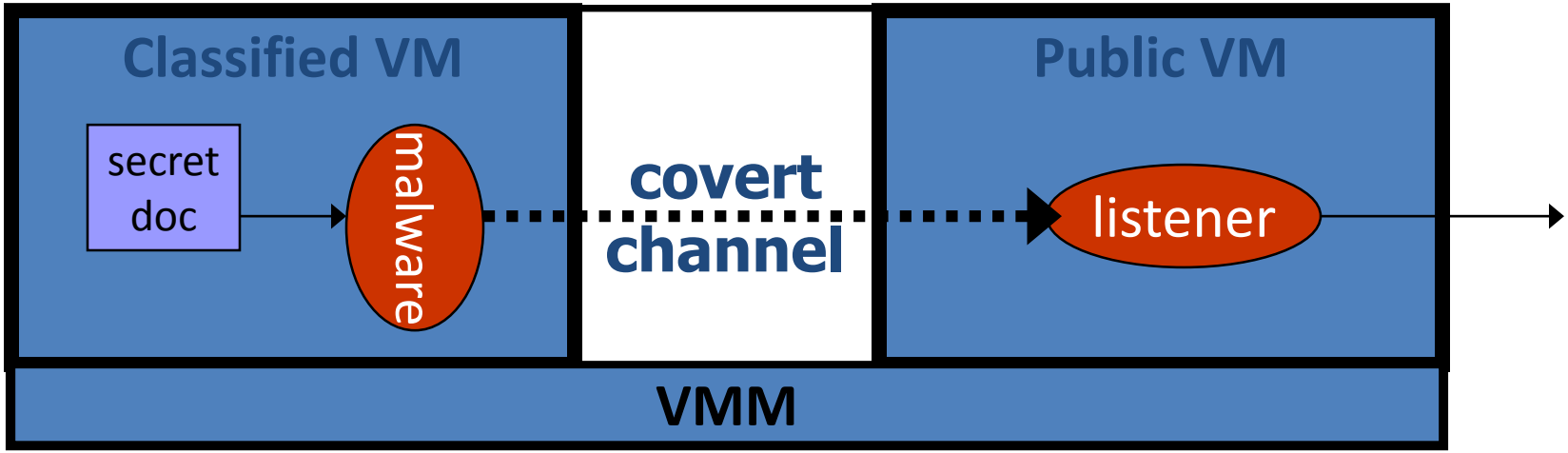
- Malware can infect guest OS and guest apps
- But malware cannot escape from the infected VM
  - Cannot infect host OS
  - Cannot infect other VMs on the same hardware

Requires that VMM protect itself and is not buggy

- VMM is much simpler than full OS
  - … but device drivers run in Host OS

Problem: covert channels

- **Covert channel**: unintended communication channel between isolated components
  - Can be used to leak classified data from secure component to public component



An example covert channel

Both VMs use the same underlying hardware

To send a bit $b \in \{0,1\}$ malware does:

- b= 1: at 1:00am do CPU intensive calculation
- b= 0: at 1:00am do nothing

At 1:00am listener does CPU intensive calc. and measures completion time

$$b = 1 \quad \Leftrightarrow \quad \text{completion-time} > \text{threshold}$$

Many covert channels exist in running system:

- File lock status, cache contents, interrupts, …
- Difficult to eliminate all

Suppose the system in question has two CPUs: the classified VM runs on one and the public VM runs on the other.

Is there a covert channel between the VMs?

There are covert channels, for example, based on the time needed to read from main memory

Module 3.1.1

VM Based Isolation, Introspection, and Subversion

VMM Introspection: [GR’03]

protecting the anti-virus system

Intrusion Detection / Anti-virus

Runs as part of OS kernel and user space process

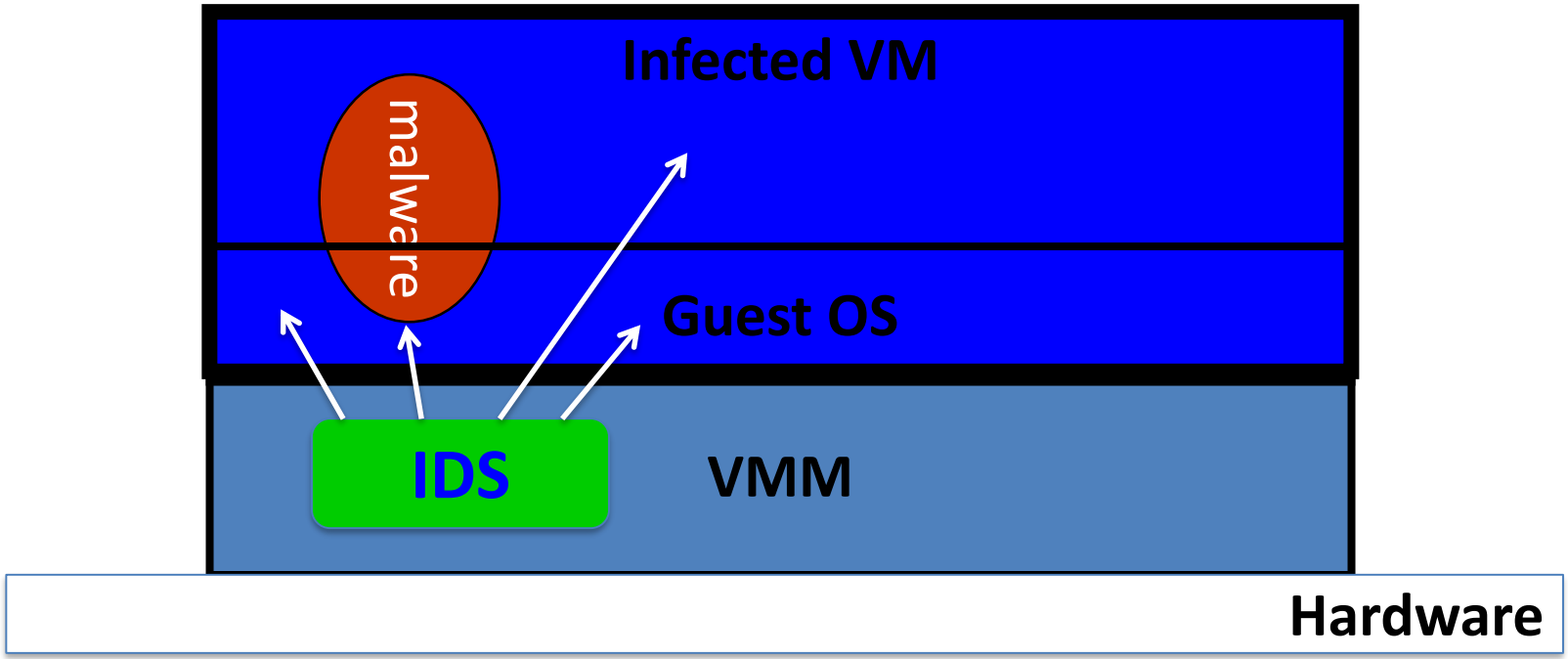
- Kernel root kit can shutdown protection system
- Common practice for modern malware

Standard solution: **run IDS system in the network**

- Problem: insufficient visibility into user's machine

Better: **run IDS as part of VMM (protected from malware)**

- VMM can monitor virtual hardware for anomalies
- VMI: Virtual Machine Introspection
  - Allows VMM to check Guest OS internals


Infected VM
malware
Guest OS
IDS
VMM
Hardware

Sample checks

Stealth root-kit malware:

- Creates processes that are invisible to "ps"
- Opens sockets that are invisible to "netstat"

1. **Lie detector check**
   - Goal: detect stealth malware that hides processes and network activity
   - Method:
     - VMM lists processes running in GuestOS
     - VMM requests GuestOS to list processes (e.g. ps)
     - If mismatch: kill VM

Sample checks

2. **Application code integrity detector**
   - VMM computes hash of user app code running in VM
   - Compare to whitelist of hashes
     - Kills VM if unknown program appears

3. **Ensure GuestOS kernel integrity**
   - example: detect changes to sys_call_table

4. **Virus signature detector**
   - Run virus signature detector on GuestOS memory

Module 3.2

VM Subversion

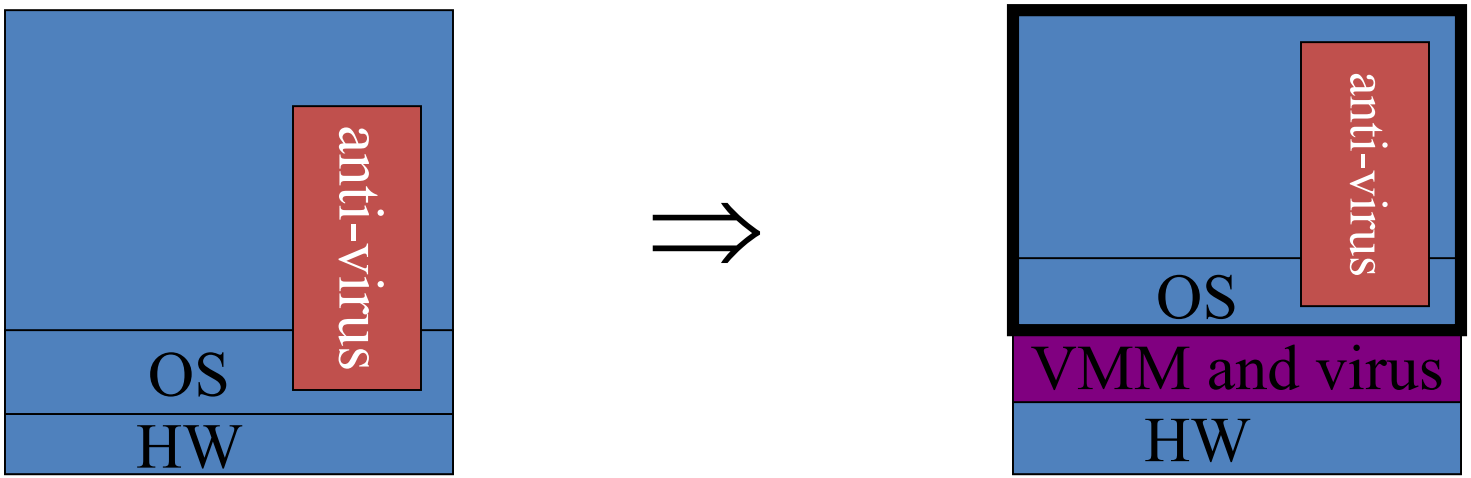
Isolation

Subverting VM Isolation

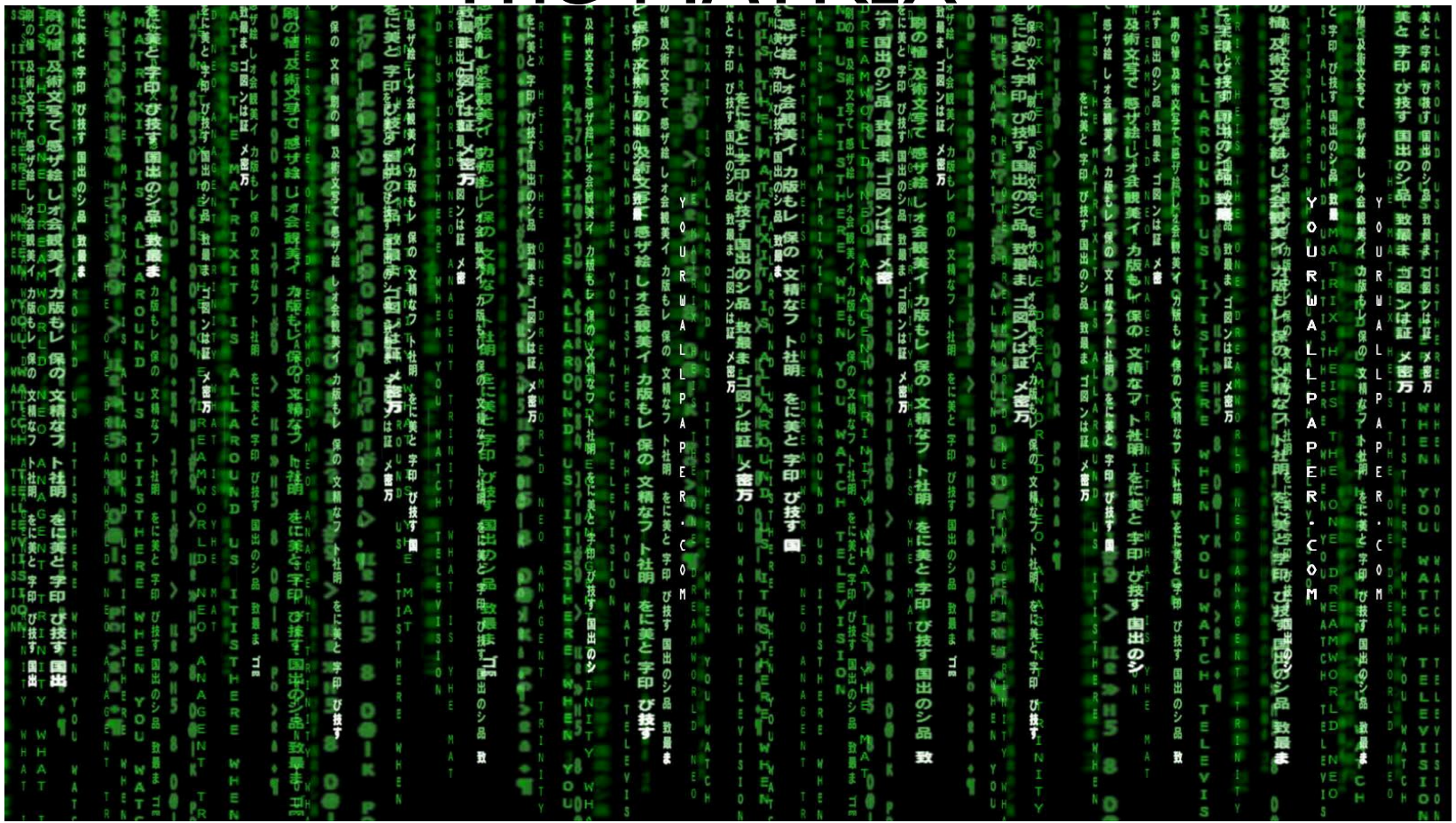
Subvirt [King et al. 2006]

Virus idea:

- Once on victim machine, install a malicious VMM
- Virus hides in VMM
- Invisible to virus detector running inside VM



The MATRIX




The Red Pill
The Blue Pill

VM Based Malware (blue pill virus)

- **VMBR**: a virus that installs a malicious VMM (hypervisor)

- **Microsoft Security Bulletin: (Oct, 2006)**
  - Suggests disabling hardware virtualization features by default for client-side systems

- **But VMBRs are easy to defeat**
  - A guest OS can detect that it is running on top of VMM

VMM Detection

Can an OS detect it is running on top of a VMM?

Applications:

- Virus detector can detect VMBR
- Normal virus (non-VMBR) can detect VMM
  - refuse to run to avoid reverse engineering
- Software that binds to hardware (e.g. MS Windows) can refuse to run on top of VMM
- DRM systems may refuse to run on top of VMM

VMM detection (red pill techniques)

- VM platforms often emulate simple hardware
  - VMWare emulates an ancient i440bx chipset
    … but report 8GB RAM, dual CPUs, etc.
- VMM introduces time latency variances
  - Memory cache behavior differs in presence of VMM
  - Results in relative time variations for any two operations
- VMM shares the TLB with GuestOS
  - GuestOS can detect reduced TLB size
- … and many more methods **[GAWF'07]**

VMM Detection

Bottom line: **The perfect VMM does not exist**

VMMs today (e.g. VMWare) focus on:

Compatibility: ensure off the shelf software works

Performance: minimize virtualization overhead

- VMMs do not provide **transparency**
  - **Anomalies reveal existence of VMM**

Module 3.3

Software Fault Isolation (SFI)

Isolation

Software Fault Isolation

Software Fault Isolation [Whabe et al., 1993]

Goal: confine apps running in same address space

- Codec code should not interfere with media player
- Device drivers should not corrupt kernel

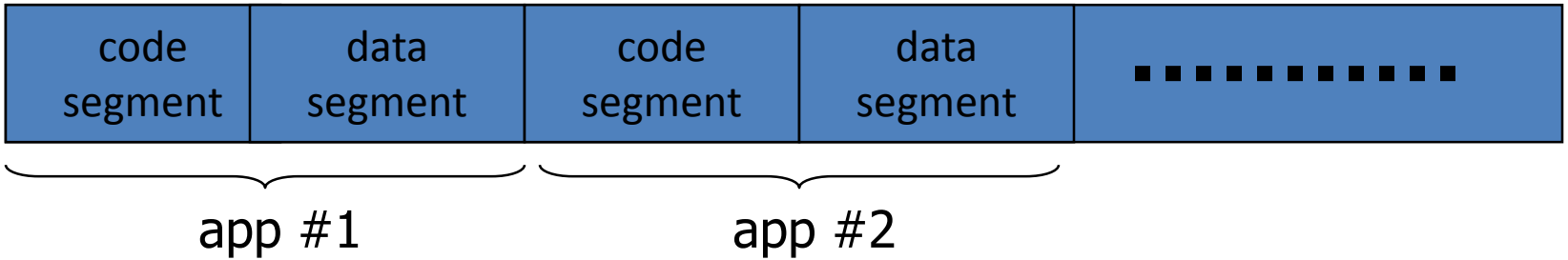
Simple solution: runs apps in separate address spaces

- Problem: slow if apps communicate frequently
  - requires context switch per message

Software Fault Isolation

SFI approach:

- Partition process memory into segments



- Locate unsafe instructions: **jmp, load, store**
  - At compile time, add guards before unsafe instructions
  - When loading code, ensure all guards are present

Segment matching technique

- Designed for MIPS processor. Many registers available.
- **dr1, dr2**: d
  - compiler
  - **dr2** conta
- Indirect load instructi ← [R34] becomes:

Guard ensures code does not load data from another segment

```
dr1 ← R34
Scratch-reg ← (dr1 >> 20)        : get segment ID
compare scratch-reg  and  dr2    : validate seg. ID
trap if not equal
R12 ← [dr1]                      : do load
```

Address sandboxing technique

- **dr2**: holds segment ID
- Indirect load instruction **R12 ← [R34]** becomes:

```
dr1 ← R34  &  segment-mask     : zero out seg bits
dr1 ← dr1  |  dr2              : set valid seg ID
R12 ← [dr1]                    : do load
```

- Fewer instructions than segment matching

  … but does not catch offending instructions
- Similar guards places on all unsafe instructions

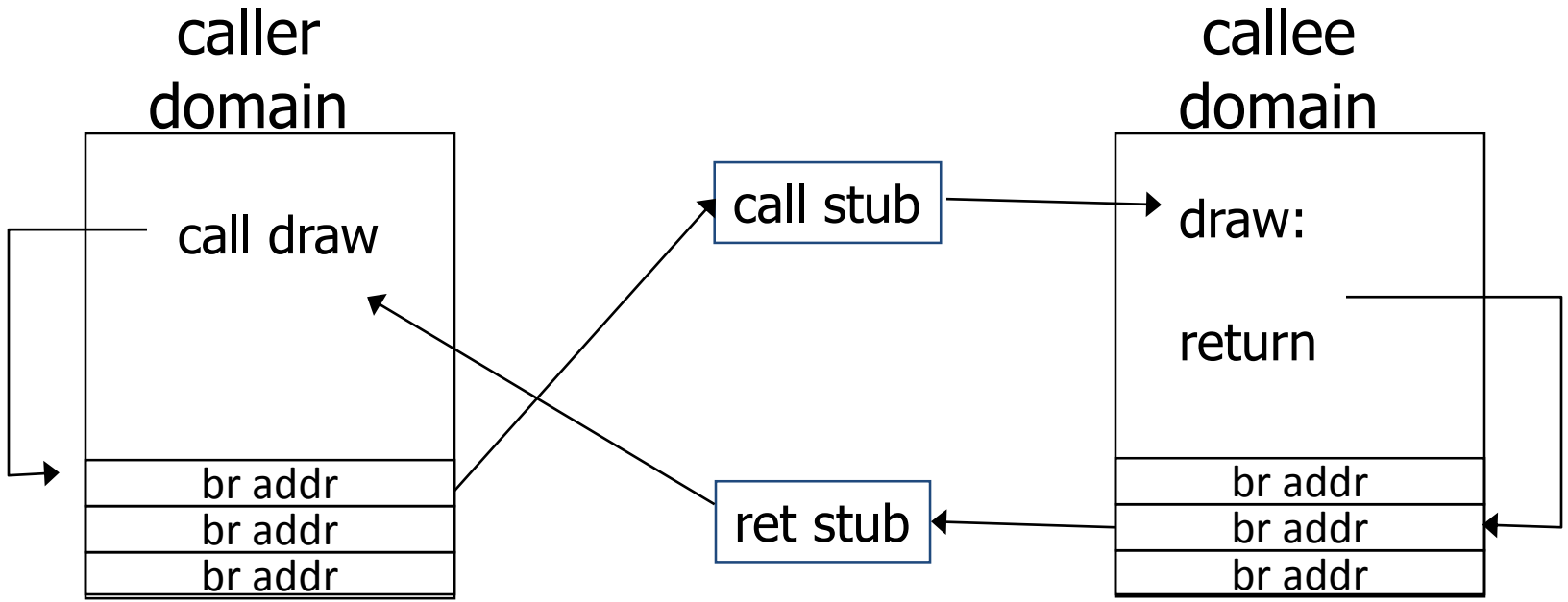
Problem: what if **jmp [addr]** jumps directly into indirect load?

(bypassing guard)

Solution:

jmp guard must ensure **[addr]** does not bypass load guard

Cross domain calls



- Only stubs allowed to make cross-domain jumps
- Jump table contains allowed exit points
  - Addresses are hard coded, read-only segment

SFI Summary

- Shared memory: use virtual memory hardware
  - map same physical page to two segments in addr space

- Performance
  - Usually good: mpeg_play, 4% slowdown

- Limitations of SFI: harder to implement on x86 :
  - variable length instructions: unclear where to put guards
  - few registers: can't dedicate three to SFI
  - many instructions affect memory: more guards needed

Isolation: summary

- Many sandboxing techniques:

  Physical air gap, Virtual air gap (VMMs),
  System call interposition, Software Fault isolation
  Application specific (e.g. Javascript in browser)

- Often complete isolation is inappropriate
  - Apps need to communicate through regulated interfaces

- Hardest aspects of sandboxing:
  - Specifying policy: what can apps do and not do
  - Preventing covert channels

Module 3.4

Rootkits

A Detour: Few Words about Rootkits

Topics

Rootkits

- User-mode Rootkits
- Kernel Rootkits
- Detecting Rootkits
- Recovery from a Rootkit

Module 3.4.1

Rootkit Basics

What is a rootkit?

Collection of attacker tools installed after an intruder has gained access

- Log cleaners
- File/process/user hiding tools
- Network sniffers
- Backdoor programs

Rootkit Goals

1. Remove evidence of original attack and activity that led to rootkit installation.
2. Hide future attacker activity (files, network connections, processes) and prevent it from being logged.
3. Enable future access to system by attacker.
4. Install tools to widen scope of penetration.
5. Secure system so other attackers can’t take control of system from original attacker.

Concealment Techniques

- Remove log and audit file entries.
- Modify system programs to hide attacker files, network connections, and processes.
- Modify logging system to not log attacker activities.
- Modify OS kernel system calls to hide attacker activities.

Installation Concealment

- Use a subdirectory of a busy system directory like /dev, /etc, /lib, or /usr/lib
- Use dot files, which aren't in ls output.
- Use spaces to make filenames look like expected dot files: ". " and ".. "
- Use filenames that system might use
  - /dev/hdd (if no 4th IDE disk exists)
  - /usr/lib/libX.a (libX11 is real Sun X-Windows)
- Delete rootkit install directory once installation is complete.

Attack Tools

- Network sniffer
  - Including password grabber utility
- Password cracker
- Vulnerability scanners
- Autorooter
  - Automatically applies exploits to host ranges
- DDOS tools

History of Rootkits

1989: Phrack 25 Black Tie Affair: wtmp wiping.

1994: Advisory CA-1994-01 about SunOS rootkits.

1996: Linux Rootkits (lrk3 released.)

1997: Phrack 51 halflife article: LKM-based rootkits

1998: Silvio Cesare's kernel patching via kmem.

1999: Greg Hoglund's NT kernel rootkit paper

History of Rootkits

2005: Sony ships CDs with rootkits that hide DRM and spyware that auto-installs when CD played.

2006: SubVirt rootkit moves real OS to a VM.

Module 3.4.2

Types of Rootkits

Rootkit Types

User-mode Rootkits

- **Binary Rootkits** replace user programs.
  - Trojans: ls, netstat, ps
  - Trojan backdoors: login, sshd.
- **Library Rootkits** replace system libraries.
  - Intercept lib calls to hide activities and add backdoors.

Kernel Rootkits

- Modify system calls/structures that all user-mode programs rely on to list users, processes, and sockets.
- Add backdoors to kernel itself.

Binary Rootkits

- Install trojan-horse versions of common system commands, such as ls, netstat, and ps to hide attacker activities..
- Install programs to edit attacker activity from log and accounting files.
- Install trojan-horse variants of common programs like login, passwd, and sshd to allow attacker continued access to system.
- Install network sniffers.

Linux Root Kit (LRK) v4 Features

chsh	Trojaned! User->r00t
crontab	Trojaned! Hidden Crontab Entries
du	Trojaned! Hide files
fix	File fixer!
ifconfig	Trojaned! Hide sniffing
inetd	Trojaned! Remote access
linsniffer	Packet sniffer!
login	Trojaned! Remote access
ls	Trojaned! Hide files
netstat	Trojaned! Hide connections
passwd	Trojaned! User->r00t
ps	Trojaned! Hide processes
rshd	Trojaned! Remote access
sniffchk	Program to check if sniffer is up and running
syslogd	Trojaned! Hide logs
tcpd	Trojaned! Hide connections, avoid denies
top	Trojaned! Hide processes
wted	wtmp/utmp editor!
z2	Zap2 utmp/wtmp/lastlog eraser!

Linux Root Kit (LRK) v4 Trojans

ifconfig – Doesn’t display PROMISC flag when sniffing.

login – Allows login to any account with the rootkit password. If root login is refused on your terminal login as "rewt". Disables history logging when backdoor is used.

ls – Hides files listed in /dev/ptyr. All files shown with 'ls -/' if SHOWFLAG enabled.

passwd – Enter your rootkit password instead of old password to become root.

ps – Hides processes listed in /dev/ptyp.

rshd – Execute remote commands as root: rsh -l rootkitpassword host command

syslogd – Removes log entries matching strings listed in /dev/ptys.

Binary Rootkit Detection

Use non-trojaned programs

- ptree is generally uncompromised
- tar will archive hidden files, the list with -t
- lsof is also generally safe
- Use known good tools from CD-ROM.

File integrity checks

- tripwire, AIDE, Osiris
- rpm –V –a
- Must have known valid version of database offline or attacker may modify file signatures to match Trojans.

Library Rootkits

- t0rn rootkit uses special system library libproc.a to intercept process information requested by user utilities.
- Modify libc
  - Intercept system call data returning from kernel, stripping out evidence of attacker activities.
  - Alternately, ensure that rootkit library providing system calls is called instead of libc by placing it in /etc/ld.so.preload

Kernel Rootkits

Kernel runs in supervisor processor mode

- Complete control over machine.

Rootkits modify kernel system calls

- execve modified to run Trojan horse binary for some programs, while other system calls used by integrity checkers read original binary file.
- setuid modified to give root to a certain user.

Advantage—Stealth

- Runtime integrity checkers cannot see rootkit changes.
- All programs impacted by kernel Trojan horse.
- Open backdoors/sniff network without running processes.

Types of Kernel Rootkits

Loadable Kernel Modules

- Device drivers are LKMs.
- Can be defeated by disabling LKMs.
- ex: Adore, Knark

Alter running kernel in memory.

- Modify /dev/kmem directly.
- ex: SucKit

Alter kernel on disk.

Kernel Rootkit Detection

List kernel modules

- lsmod
- cat /proc/modules

Examine kernel symbols (/proc/kallsyms)

- Module name listed in [] after symbol name.

Kernel Rootkit Detection

Check system call addresses

- Compare running kernel syscall addresses with those listed in System.map generated at kernel compile.

All of these signatures can be hidden/forged.

Knark

- Linux-based LKM rootkit
- Features
  - Hide/unhide files or directories
  - Hide TCP or UDP connections
  - Execution redirection
  - Unauthenticated privilege escalation
  - Utility to change UID/GID of a running process.
  - Unauthenticated, privileged remote execution daemon.
  - Kill –31 to hide a running process.
- modhide: assistant LKM that hides Knark from module listing attempts.

Module 3.4.3

Rootkit Detection

Rootkit Detection

Offline system examination

- Mount and examine disk using another OS kernel+image.
- Knoppix: live CD linux distribution.

Computer Forensics

- Examine disk below filesystem level.
- Helix: live CD linux forensics tool.

Rootkit Detection Utilities

chkrootkit

- Detects >50 rootkits on multiple UNIX types.
- Checks commonly trojaned binaries.
- Examines log files for modifications.
- Checks for LKM rootkits.
- Use –p option to use known safe binaries from CDROM.

carbonite

- LKM that searches for rootkits in kernel.
- Generates and searches frozen image kernel process structures.

Detection Countermeasures

- Hide rootkit in unused sectors or in unused fragments of used sectors.
- Install rootkit into flash memory like PC BIOS, ensuring that rootkit persists even after disk formatting and OS re-installation.

Rootkit Recovery

- Restore compromised programs from backup
  - Lose evidence of intrusion.
  - Did you find all the trojans?
- Backup system, then restore from tape
  - Save image of hard disk for investigation.
  - Restore known safe image to be sure that all trojans have been eliminated.
  - Patch system to repair exploited vulnerability.

Key Points

- Backdoors allow intruder into system without using exploit again.
- Rootkits automatically deeply compromise a system once root access is attained.
- Rootkits are easy to use, difficult to detect.
- Don’t trust anything on a compromised system—access disk from a known safe system, like a Knoppix CD.
- Recovery requires a full re-installation of the OS and restoration of files from a known good backup.

References

1. Oktay Altunergil, “Scanning for Rootkits,” http://www.linuxdevcenter.com/pub/a/linux/2002/02/07/rootkits.html, 2002.
2. Silvio Cesare, “Runtime kernel kmem patching,” http://vx.netlux.org/lib/vsc07.html, 1998.
3. William Cheswick, Steven Bellovin, and Avriel Rubin, *Firewalls and Internet Security, 2nd edition*, 2003.
4. Anton Chuvakin, “An Overview of UNIX Rootkits,” iDEFENSE whitepaper, 2003.
5. Dave Dittrich, “Rootkits FAQ,” http://staff.washington.edu/dittrich/misc/faqs/rootkits.faq, 2002.
6. Greg Hoglund and Gary McGraw, *Exploiting Software: How to Break Code*, Addison-Wesley, 2004.
7. Samuel T. King et. al., “SubVirt: Implementing malware with virtual machines”, http://www.eecs.umich.edu/virtual/papers/king06.pdf, 2006.
8. McClure, Stuart, Scambray, Joel, Kurtz, George, *Hacking Exposed, 3rd edition*, McGraw-Hill, 2001.
9. Peikari, Cyrus and Chuvakin, Anton, *Security Warrior*, O’Reilly & Associates, 2003.
10. pragmatic, (nearly) Complete Loadable Linux Kernel Modules, http://www.thc.org/papers/LKM_HACKING.html, 1999.
11. Marc Russinovich, “Sony, Rootkits and Digital Rights Management Gone Too Far,” http://blogs.technet.com/markrussinovich/archive/2005/10/31/sony-rootkits-and-digital-rights-management-gone-too-far.aspx
12. Jennifer Rutkowska, “Red Pill: or how to detect VMM using (almost) one CPU instruction**,”** http://www.invisiblethings.org/papers/redpill.html, 2004.
13. Ed Skoudis, *Counter Hack Reloaded*, Prentice Hall, 2006.
14. Ed Skoudis and Lenny Zeltser, *Malware: Fighting Malicious Code*, Prentice Hall, 2003.
15. Ranier Wichman, “Linux Kernel Rootkits,” http://la-samhna.de/library/rootkits/index.html, 2002.

Module 3.5

Intrusion and Intrusion Detection

Intrusion Detection Systems (IDS)

Another Digression

Intrusion and Intrusion Detection

- Intrusion : Attempting to break into or misuse your system.
- Intruders may be from outside the network or legitimate users of the network.
- Intrusion can be a physical, system or remote intrusion.

Different ways to intrude

- Buffer overflows
- Unexpected combinations
- Unhandled input
- Race conditions

Intrusion Detection Systems (IDS)

Signature Based:

Intrusion Detection Systems look for attack signatures, which are specific patterns that usually indicate malicious or suspicious intent.

Anomaly Detection Based:

Machine learning techniques used to characterize normal behavior from anomalous behavior

Intrusion Detection Systems (IDS)

- Different ways of classifying an IDS

  IDS based on
  - anomaly detection
  - signature based misuse
  - host based
  - network based

Anomaly based IDS

- This IDS models the normal usage of the network as a noise characterization.
- Anything distinct from the noise is assumed to be an intrusion activity.
  - E.g flooding a host with lots of packet.
- The primary strength is its ability to recognize novel attacks.

Drawbacks of Anomaly detection IDS

- Assumes that intrusions will be accompanied by manifestations that are sufficiently unusual so as to permit detection.
- These generate many false alarms and hence compromise the effectiveness of the IDS.

Signature based IDS

- This IDS possess an attacked description that can be matched to sensed attack manifestations.
- The question of what information is relevant to an IDS depends upon what it is trying to detect.
  - E.g DNS, FTP etc.

Signature based IDS (contd.)

- ID system is programmed to interpret a certain series of packets, or a certain piece of data contained in those packets,as an attack. For example, an IDS that watches web servers might be programmed to look for the string “phf” as an indicator of a CGI program attack.
- Most signature analysis systems are based off of simple pattern matching algorithms. In most cases, the IDS simply looks for a sub string within a stream of data carried by network packets. When it finds this sub string (for example, the ``phf'' in ``GET /cgi-bin/phf?''), it identifies those network packets as vehicles of an attack.

Drawbacks of Signature based IDS

- They are unable to detect novel attacks.
- Suffer from false alarms
- Have to be programmed again for every new pattern to be detected.

Host/Applications based IDS

- The host operating system or the application logs in the audit information.
- These audit information includes events like the use of identification and authentication mechanisms (logins etc.) , file opens and program executions, admin activities etc.
- This audit is then analyzed to detect trails of intrusion.

Drawbacks of the host based IDS

- The kind of information needed to be logged in is a matter of experience.
- Unselective logging of messages may greatly increase the audit and analysis burdens.
- Selective logging runs the risk that attack manifestations could be missed.

Strengths of the host based IDS

- Attack verification
- System specific activity
- Encrypted and switch environments
- Monitoring key components
- Near Real-Time detection and response.
- No additional hardware

Stack based IDS

- They are integrated closely with the TCP/IP stack, allowing packets to be watched as they traverse their way up the OSI layers.
- This allows the IDS to pull the packets from the stack before the OS or the application have a chance to process the packets.

Network based IDS

- This IDS looks for attack signatures in network traffic via a promiscuous interface.
- A filter is usually applied to determine which traffic will be discarded or passed on to an attack recognition module. This helps to filter out known un-malicious traffic.

Strengths of Network based IDS

- Cost of ownership reduced
- Packet analysis
- Real time detection and response
- Malicious intent detection
- Operating system independence

Module 3.5.1

Commercial IDS

Commercial ID Systems

- ISS – Real Secure from Internet Security Systems:
  - Real time IDS.
  - Contains both host and network based IDS.
- Tripwire – File integrity assessment tool.
- Bro and Snort – open source public-domain system.

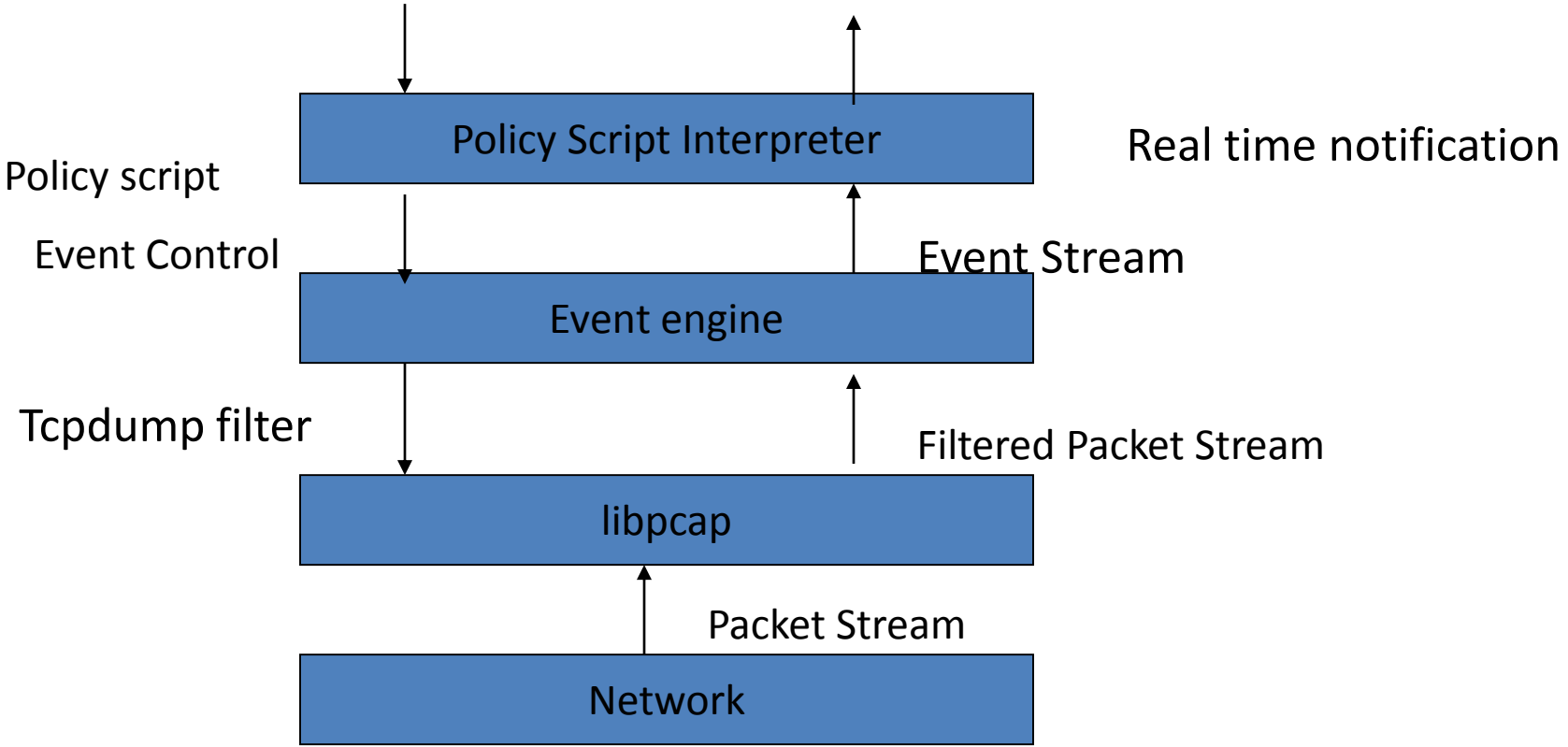
Bro: Real time IDS

- Network based IDS
- Currently developed for six Internet applications: FTP, Finger, Portmapper, Ident, Telnet and Rlogin.

Design goals for Bro

- High-speed, large volume monitoring
- No packet filter drops
- Real time notification
- Mechanism separate from policy
- Extensible

Structure of the Bro System



Bro - libpcap

- It’s the packet capture library used by tcpdump.
- Isolates Bro from details of the network link technology.
- Filters the incoming packet stream from the network to extract the required packets.
- E.g port finger, port ftp, tcp port 113 (Ident), port telnet, port login, port 111 (Portmapper).
- Can also capture packets with the SYN, FIN, or RST Control bits set.

Bro – Event Engine

- The filtered packet stream from the libpcap is handed over to the Event Engine.
- Performs several integrity checks to assure that the packet headers are well formed.
- It looks up the connection state associated with the tuple of the two IP addresses and the two TCP or UDP port numbers.
- It then dispatches the packet to a handler for the corresponding connection.

Bro – TCP Handler

- For each TCP packet, the connection handler verifies that the entire TCP Header is present and validates the TCP checksum.
- If successful, it then tests whether the TCP header includes any of the SYN/FIN/RST control flags and adjusts the connection's state accordingly.
- Different changes in the connection's state generate different events.

Policy Script Interpreter

- The policy script interpreter receives the events generated by the Event Engine.
- It then executes scripts written in the Bro language which generates events like logging real-time notifications, recording data to disk or modifying internal state.
- Adding new functionality to Bro consists of adding a new protocol analyzer to the event engine and then writing new events handlers in the interpreter.

Future of IDS

- To integrate the network and host based IDS for better detection.
- Developing IDS schemes for detecting novel attacks rather than individual instantiations.

Lecture 3: Summary

- Module 3.1: VM based Isolation
  - Module 3.1.1 Subversion
- Module 3.2: Confinement Principle
- Module 3.3: Software Fault Isolation
- Module 3.4: Rootkits
  - Module 3.4.1 – Rootkit Basics
  - Module 3.4.2 – Rootkit Types
  - Module 3.4.3 – Rootkit Detection
- Module 3.5: IDS – Intrusion Detection Systems
  - Module 3.5.1 – Commercial IDS