Automated Defense from Rootkit Attacks

Arati Baliga and Liviu Iftode

Computer Science Department

Rutgers University

110 Frelinghuysen Road, Piscataway, NJ

Xiaoxin (Mike) Chen

VMware Inc.

3145 Porter Drive

Palo Alto, CA

About Us

- Laboratory of Distributed Computing (DISCO Lab)
  - Headed by Prof. Liviu Iftode
  - 10 graduate students
  - http://discolab.rutgers.edu
- Projects
  - Remote Healing Using Backdoors
    - Remote repair/recovery of operating system and application state.
  - Defensive Architectures
    - Automated defense against attacks and intrusions

Outline

- Motivation
- Background
- Our Approach
- Prototype
- Work in Progress
- Future Work
- Related Work
- Conclusion

Motivation

- With the increasing attack trends, human response to intrusions is too slow or not possible
  - Software and OS vulnerabilities
  - Fast spreading worms (SQL Slammer)
- We need systems that can detect intrusions automatically as early as possible and recover

Outline

- Motivation
- Background
- Our Approach
- Prototype
- Work in Progress
- Future Work
- Related Work
- Conclusion

Viruses and Worms

- Viruses replicate by modifying a normal program/file with a copy of itself.
  - Execution in the host program/file results in the execution of the virus
  - Usually needs human action to execute infected progam
- Worms are stand-alone programs that spread through the network by exploiting vulnerabilities in services.

Rootkits

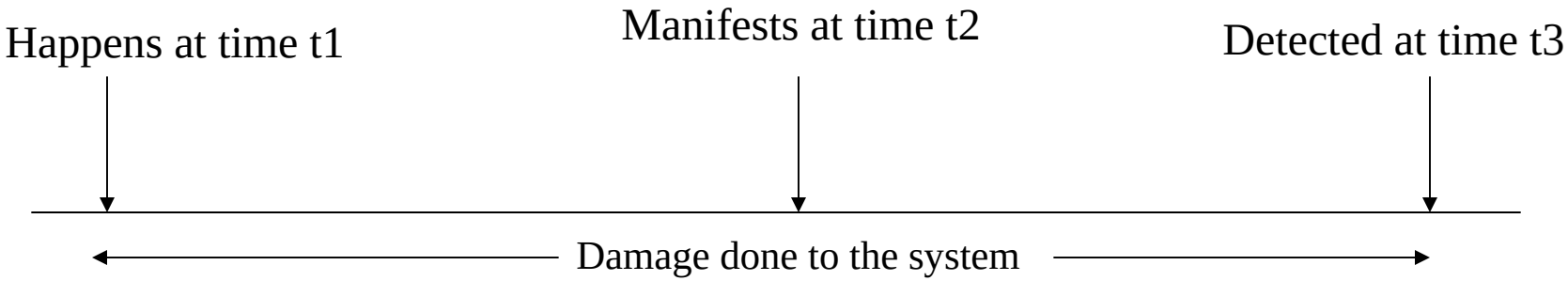
- Collection of tools used by the attacker to hold root privileges on the compromised system.
- Rootkit hiding mechanism:
  - User-level rootkits
    - Replace system binaries like ps and netstat
  - Shared library rootkits
    - Replace shared libraries
  - Kernel-level rootkits
    - Replace entries in system call table
    - Replace entries in interrupt descriptor table
    - Replace kernel/module text.
- Compromise system integrity

Stealth Malware

- Increasing number of virus/worm writers use rootkits to evade detection from anti-virus software.
  - Stealth AOL worm

- In this talk, we refer to this group of malware that hide using rootkit techniques as “stealth malware”.

Intrusion Timeline



- Damage done to the system from t1-t3 needs to be discovered and undone – this takes time!
- Ideally intrusion should be detected at t1(Prevention)
  - Easier for known attacks
  - Hard for new/unknown attacks
- Intrusion Detection Systems
  - Move t3 closer to t2
  - Recover (undo damage)
  - Can t3 be moved between t1 and t2?

Intrusion Detection Systems (IDS)

- Detection methods
  - Signature based detection
  - Misuse detection
  - Anomaly detection
- Detection Location
  - Host Based IDS
  - Network Based IDS

IDS Implementation

- Stand-alone systems: Copilot[Arbaugh et al], Backdoors[Bohra et al]
  - Independent secure device
    - Polling based approaches
    - Does not consume CPU cycles
- Virtualized Environment [VMI, Mendel et al]
  - Can intercept events in the virtual machine
  - Can interpose/change virtual machine state to perform preventive/healing actions

Ideal IDS Properties

- Prevent system from being compromised
- Detect intrusion asap if prevention is not possible.
- Once intrusion is detected, recover the system to restore faith in the system.
  - Transparently
  - Reveal to the user the damage that has occurred.
- Generate an attack profile

Outline

- Motivation
- Background
- Our Approach
- Prototype
- Work in Progress
- Future Work
- Related Work
- Conclusion

Our Approach

- **Detect** malicious process when it performs illegal access to ***protected zones***.
- **Track** dependencies between files and processes to undo damage automatically
- **Undo** damage

Protected Zones

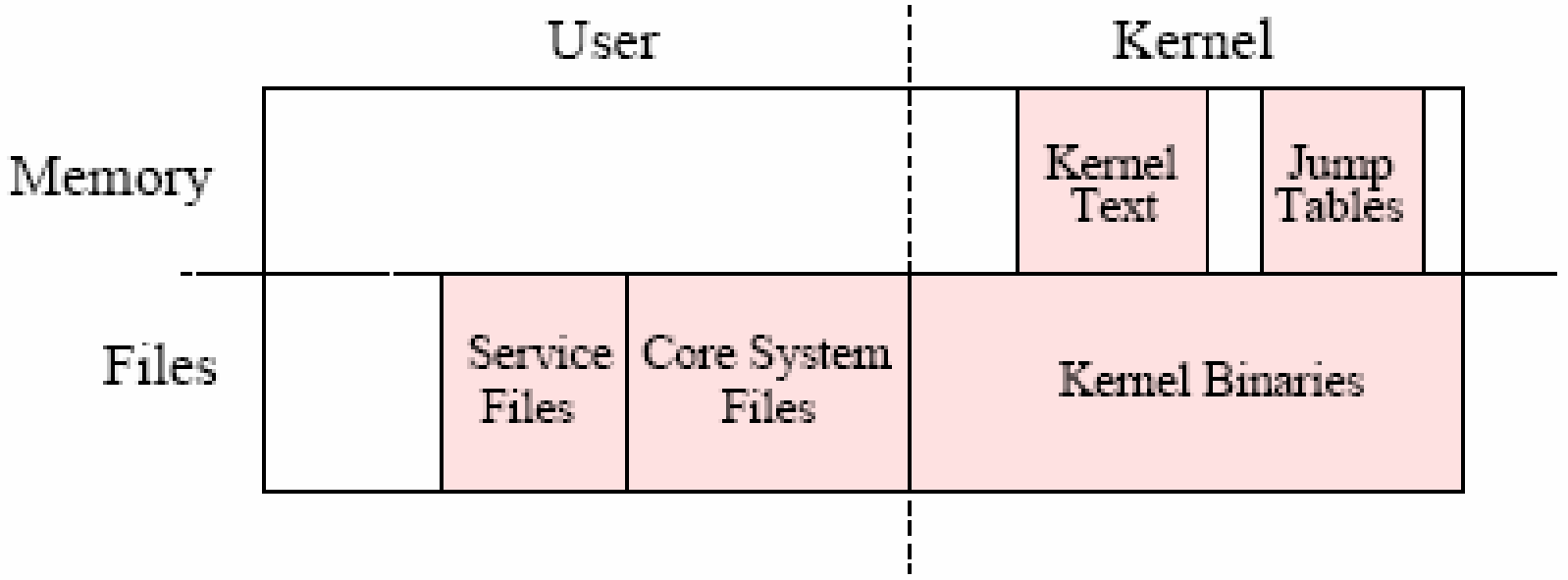


Fig. 1. Figure shows the protected parts of the memory and filesystem that are shaded in pink. This represents the core of the system, which is always protected. The unshaded portions consist of all other files and running programs, which can be compromised at any time.

Protected Zones (cont'd)

- Core System Files
  - Normally do not change during the lifetime of the system
  - Examples: ps, ls, netstat, etc.
- Service Files
  - Can be added by administrator depending on which service needs to be protected.
  - Example: apache binary

Our Approach

- **Detect** malicious process when it performs illegal access to ***protected zones***.
- **Track** dependencies between files and processes to undo damage automatically
- **Undo** damage

Track Dependencies

- **Infer dependencies**
  - Dependencies between files and processes.
  - Parent-child relationships between processes.
- **Store dependencies**
  - Create a tree of dependencies (Dependency tree)
  - Dependency tree is stored in a database.
  - Dependency tree size has to be small enough to provide fast undo.

Infer Dependencies

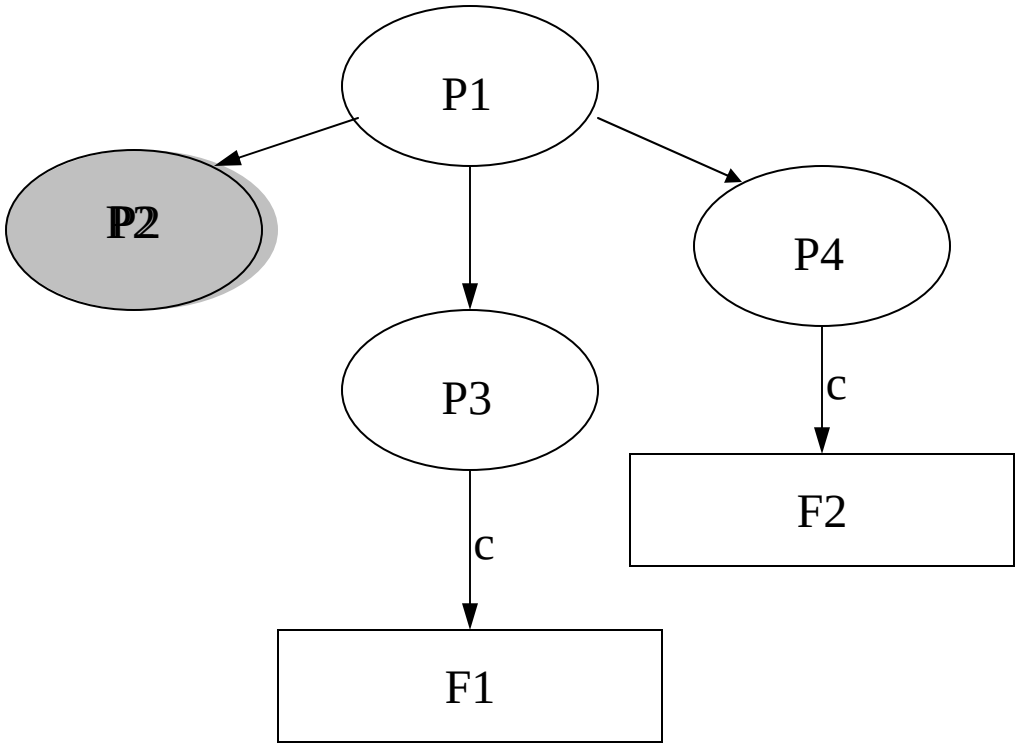
- Get system call data (number, arguments)
- Page directory base identifies process uniquely
  - **cr3** register on the x86 platform
- File identified by full pathname

Infer Dependencies (cont'd)

- On fork
  - Child has same <**esp,eip**> but different **cr3**
  - When processes executed from the same program, fork at a given point, all children have same <**esp,eip**>
  - Ambiguities are resolved using physical page number of stack page
  - Relies on Linux fork copy-on-write semantics

- Apply dependency rules

Dependency Tree



P1 creates P2

P2 exits

P1 creates P3

P1 creates P4

P3 creates F1

P4 creates F2

F1 is deleted

P4 exits

P1 exits

Dependency Storage

- Size of the dependency tree created is proportional to the number of new files and processes created on the file system.
- Storage requirements are modest
  - Dependency rules prune entries that are no longer required
- The dependency tree lives until system is rebooted

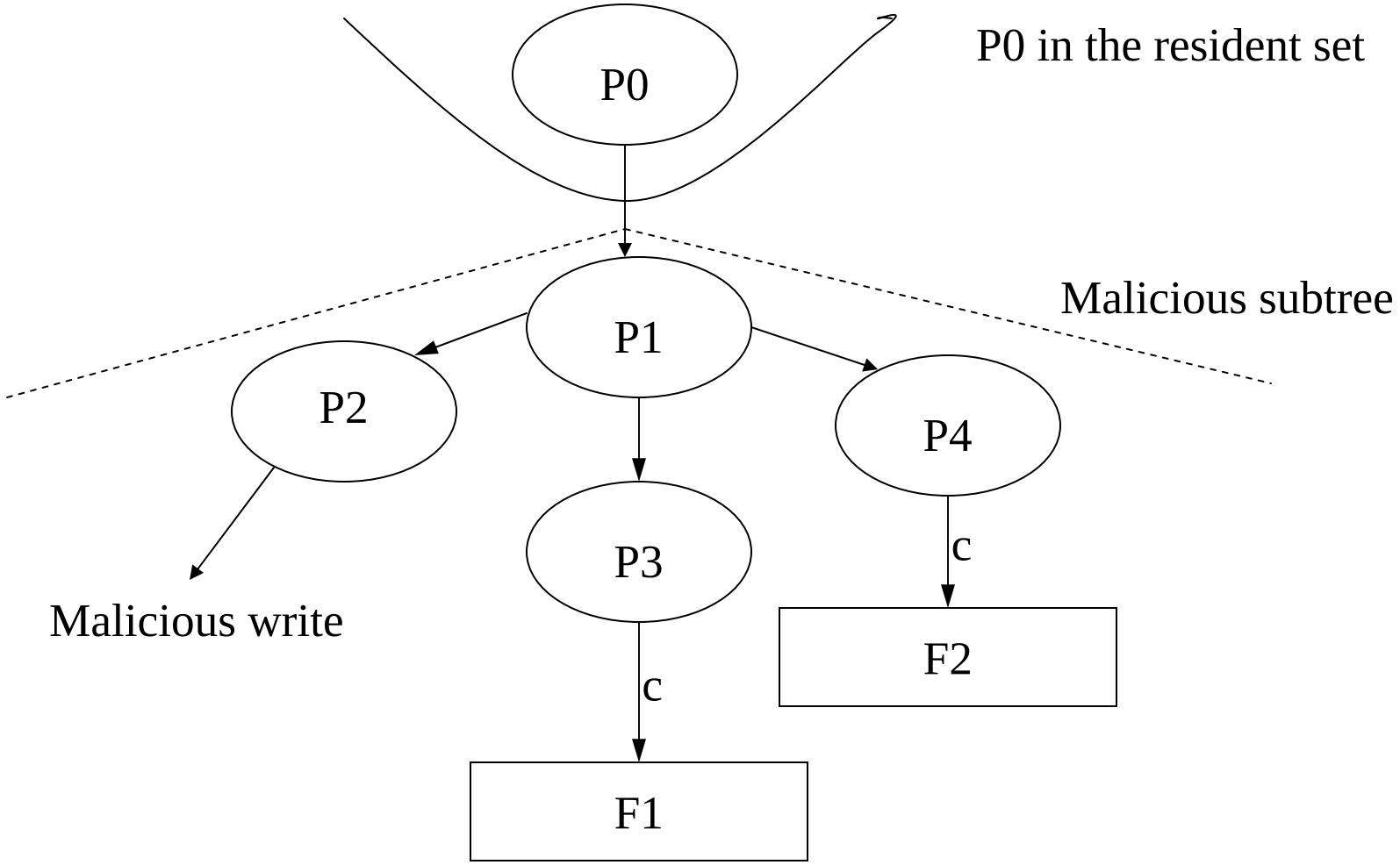
Our Approach

- **Detect** malicious process when it performs illegal access to ***protected zones***.
- **Track** dependencies between files and processes to undo damage automatically
- **Undo** damage
  - Identify and kill malicious processes using the dependency tree
  - Files not deleted

Containment Steps

- Assumes a **process resident set** that always exists in the system.
- Refer to dependency tree and locate malicious subtree
- Kill all possible malicious processes
- Prevents ill-effects
  - Installation/Existence of backdoors.
  - Keyloggers

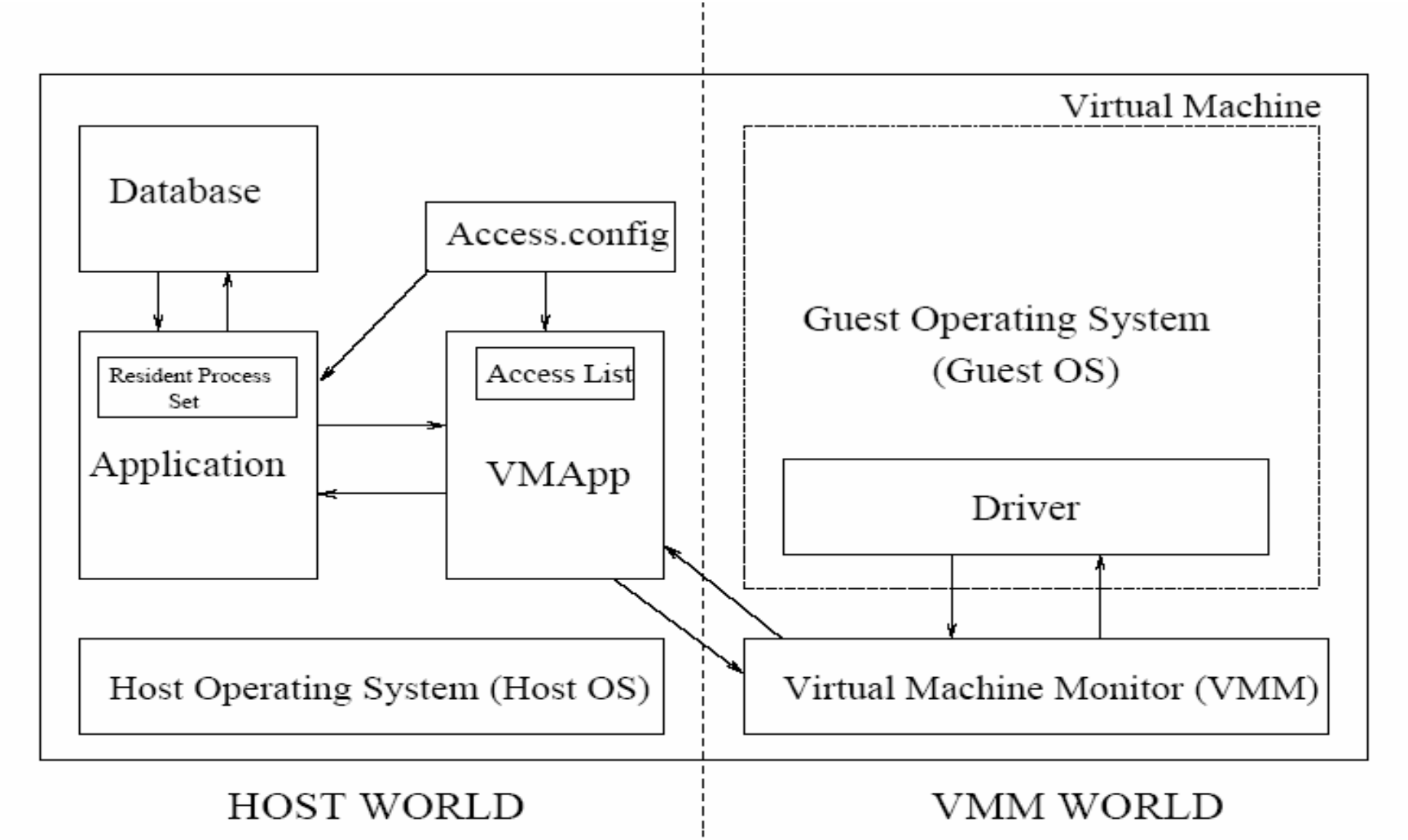
Containment Algorithm



Outline

- Motivation
- Background
- Our Approach
- Prototype
- Work in Progress
- Future Work
- Related Work
- Conclusion

Paladin Prototype



System Calls

File Related	open close creat rename unlink
Process Related	fork clone execve exit exit_group

- mmap, sockets, named pipes will be added later

Implementation Details

- Guest OS must be non-pageable
- Paladin driver
  - Has knowledge of Guest OS kernel
  - Performs symbol lookup of kernel text and jump tables
  - Also responsible for carrying out undo actions

Performance Evaluation

- VMware workstation software
- Database: MySQL
- Guest OS and Host OS: Linux 2.4 kernel.

	System calls (slow path) Without Paladin	**System calls (slow path) With Paladin**	Optimized VMM (Fast path syscalls)
Kernel Copy	**7 mins and 29 secs**	**8 mins 30 secs**	3 mins and 46 secs
Compile Kernel	**53 mins and 3 secs**	**56 mins and 7 secs**	31 mins and 54 secs

System Calls Tracked

	Total system calls	Calls processed
Kernel Copy	305690	77644 (25.4%)
Kernel Compile	7254	1276 (17.6%)

Total system calls v/s system calls processed by Paladin

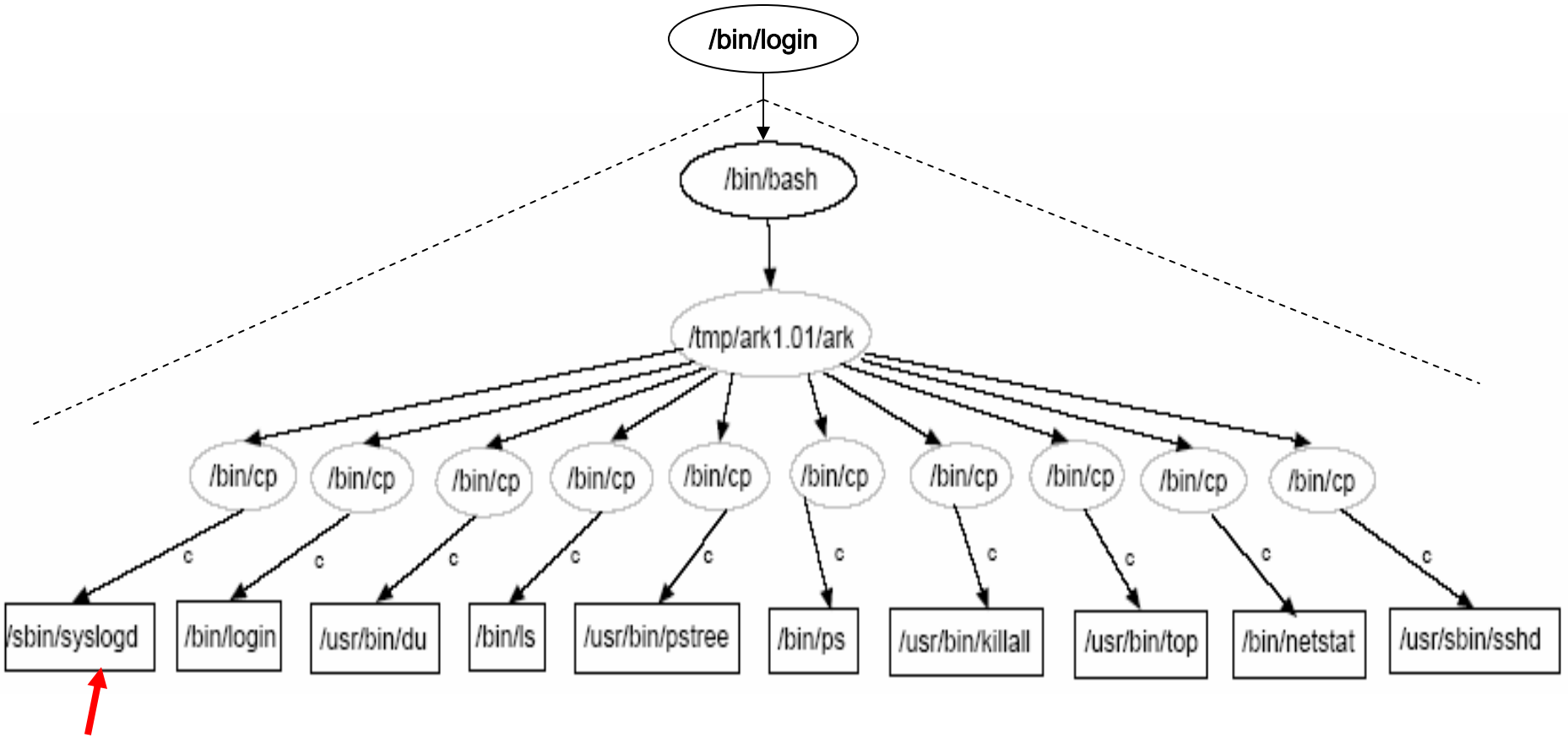
	Without Paladin	With Paladin
fork	1.5 μs	3.5 μs
exit	1.5 μs	1.6 μs
open	0.8 μs	1.5 μs
close	0.5 μs	0.7 μs

Performance of individual system calls

Test Cases

- User level rootkit
  - Ambient Rootkit (ARK)
- Kernel level rootkit
  - SuckIt
- Linux Worm
  - Lion

User-Level Rootkit: Ambient (ARK)



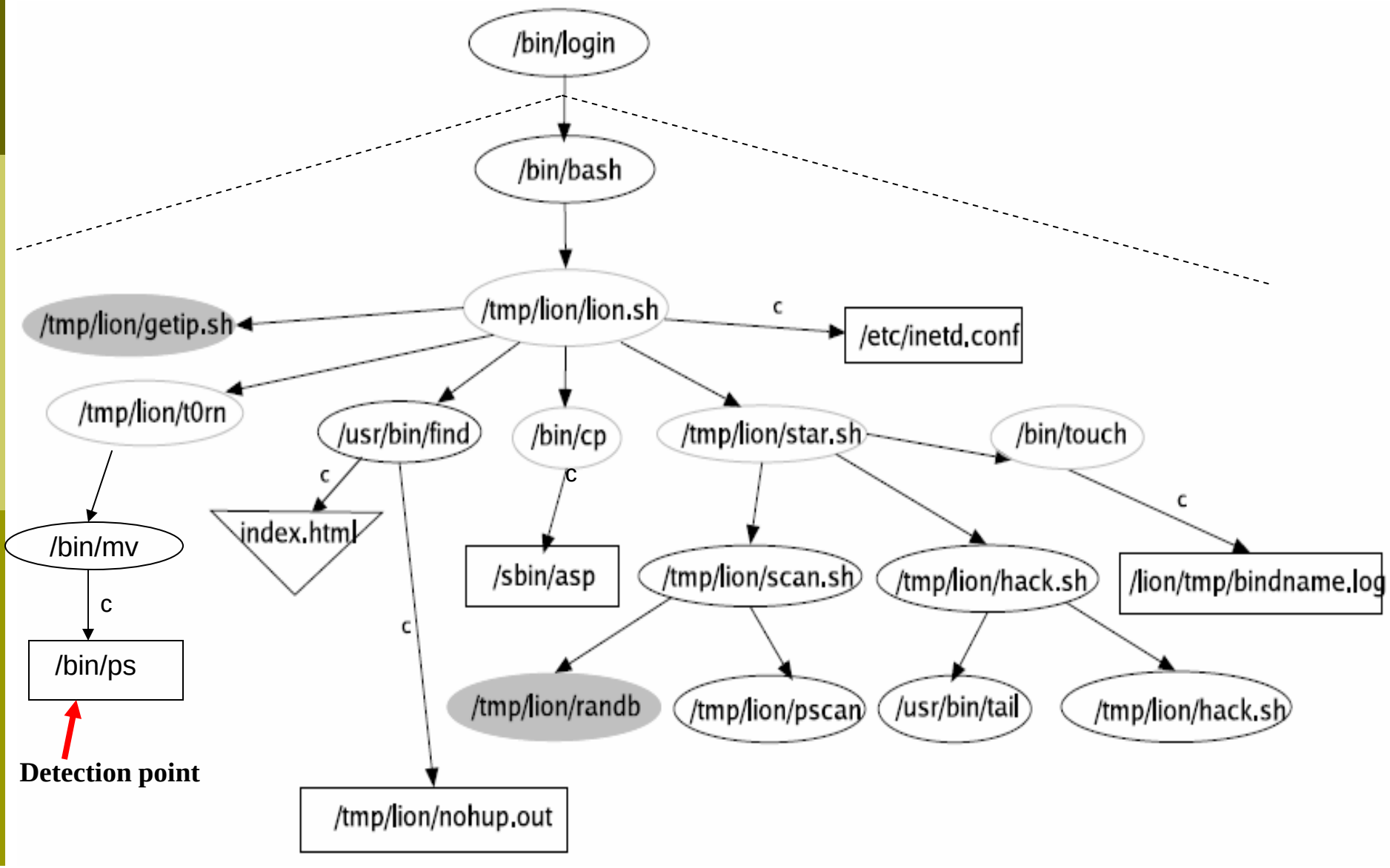
Point of detection

The security framework prevents corruption of all the system tools

Kernel-Level Rootkit: SuckIt

- Modifies instructions in sys_call to redirect to it's own system call table.
- Does not touch original system call table
- Works even when kernel module support is disabled
- Provides covert backdoor that is activated on receiving a special packet.
- The security framework prevents corruption of kernel text and installation of the backdoor.

Worm: Lion


/bin/login
/bin/bash
/tmp/lion/lion.sh
/tmp/lion/getip.sh
c
/etc/inetd.conf
/tmp/lion/t0rn
/usr/bin/find
/bin/cp
/tmp/lion/star.sh
/bin/touch
c
c
c
index.html
/bin/mv
/sbin/asp
/tmp/lion/scan.sh
/tmp/lion/hack.sh
/lion/tmp/bindname.log
c
/bin/ps
c
/tmp/lion/randb
/tmp/lion/pscan
/usr/bin/tail
/tmp/lion/hack.sh
Detection point
/tmp/lion/nohup.out

Limitations

- Current implementation does not prevent
  - Corruption of resident processes
  - Overwriting disk blocks.
- Rootkit attacks that corrupt data without changing any kernel code or tables are not detected
  - Ex: FU rootkit (Next generation rootkits ?)

Outline

- Motivation
- Background
- Our Approach
- Prototype
- Work in Progress
- Future Work
- Related Work
- Conclusion

Work in Progress: Fingerprinting

- Early attack identification through fingerprint matching
  - Find them before they hide
- Automated identification of attacker's files

Fingerprinting Steps

- Dynamic cloning
  - Spawn a clone upon attack detection
- Sandboxing
  - Reconfigure network properties
- Fine-grained monitoring
  - Watch the processes in the malicious subtree
  - Finer control possible.

Example: ARK Fingerprint

Processes:	**Files created**
/tmp/ark1.01/ark	**/dev/capi20.20**
/bin/rm	**/sbin/syslogd**
/sbin/syslogd	**/usr/lib/.ark?**
/bin/cp	**/bin/login**
/usr/sbin/sshd	**/usr/sbin/sshd**
/bin/chmod	**/bin/ls**
/bin/cat	**/usr/bin/du**
/bin/hostname	**/bin/ps**
/sbin/ifconfig	**/usr/bin/pstree**
/bin/grep	**/usr/bin/killall**
/bin/awk	**/usr/bin/top**
/bin/sed	**/bin/netstat**
/sbin/modprobe	/var/run/syslogd.pid
/usr/lib/sendmail	/var/spool/clientmqueue/dfj99KxukX001449
/usr/lib/libhesiod.so.0	/var/spool/clientmqueue/tfj99KxukX001449
	/var/spool/clientmqueue/dfj99L0HiX001457
	/var/spool/clientmqueue/tfj99L0HiX001457
	/var/spool/clientmqueue/dfj99L0cv1001466
	/var/spool/clientmqueue/tfj99L0cv1001466
	/var/spool/clientmqueue/dfj99L0w2M001475
	/var/spool/clientmqueue/tfj99L0w2M001475

Future Work

- Intrusion detection through system call anomalies
  - Generate behavioural profiles using statistical/learning techniques
- Intrusion detection through network packet monitoring
- Collaborative detection across VMs
  - Sharing attack fingerprints
  - Cross node detection

Outline

- Motivation
- Background
- Our Approach
- Prototype
- Future Work
- Related Work
- Conclusion

Related Work

- Model for Intrusion Detection
  - VMI
- Automated Detection
  - Copilot, Strider Ghostbuster, Tripwire
- Automated Post-Intrusion Analysis and Repair
  - Repairable File Service
  - Backtracker
- Automated Online Defense
  - Introvert, Vigilante

Model for Intrusion Detection

- VMI
  - Virtual Machine based Introspection (VMI) for Intrusion Detection [NDSS ‘03]

Detection

- Tools available for rootkit detection
  - Kstat, Chkrootkit, St. Michael, Samhain, F-Secure BlackLight, RootkitRevealer, Tripwire, AIDE
- Copilot
  - Automated detection from an independent PCI device [Security ‘03]
- Strider Ghostbuster
  - A cross-view diff-based approach. [DSN ‘05]

Post-Intrusion Analysis and Repair

Aid to the administrator

- Fix the filesystem and keep good changes
- Find how the intrusion happened

- RFS, Taser
  - Design, Implementation and Evaluation of Repairable File Service [DSN ‘03]
  - The Taser Intrusion Recovery System [SOSP ‘05]
- BackTracker
  - Backtracking Intrusions [SOSP ‘03]

Online Defense

Provides online defense against intrusions

- Introvert
  - Detecting Past and Present Intrusions Through Vulnerability-Specific Predicates [SOSP ‘05]
- Vigilante
  - Vigilante: End-to End Containment of Internet Worms [SOSP ’05]

Conclusion

- The steadily increasing rate of attacks and intrusions requires a robust automated solution
- We have designed an approach that protects the system from rootkit attacks and contains the effects of stealth malware that use rootkits to hide.
- We developed a prototype that demonstarted our approach that could sucessfully counter a user-level rootkit, a kernel-level rootkit and a worm that used rootkit techniques to hide

Thank You !

http://discolab.rutgers.edu

This work is supported by NSF CCR #0133366