A Study on the Security Implications of Information Leakages in Container Clouds

Xing Gao, Benjamin Steenkamer, Zhongshu Gu, Mehmet Kayaalp, Dimitrios Pendarakis, Haining Wang

IEEE Transactions on Dependable and Secure Computing, 2018

Presentation by: Benjamin Steenkamer

CPEG 697: Advanced Cybersecurity

November 25, 2018

University of Delaware

Paper Overview

1. Introduction
2. **Background**
3. **Information Leakage in Containers**
4. **Constructing Convert Channels**
5. ~~Synergistic Power Attacks~~
6. **Defense Approach**
7. **Defense Evaluation**
8. **Discussion**
9. Related Work
10. Conclusion

Overview of Findings

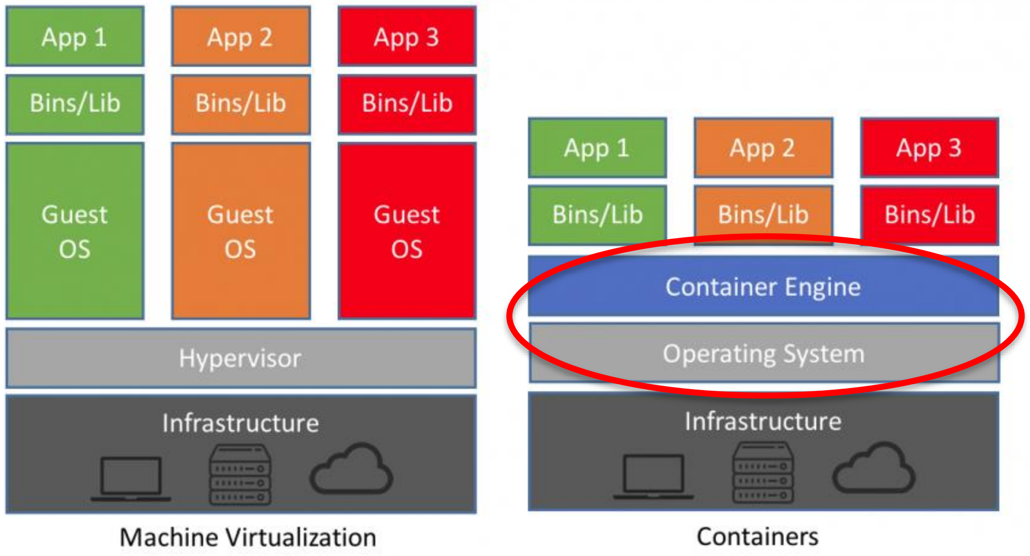
Using common container software, we:

- Systematically identify many potential and realized leakage channels in multitenant cloud container environments
  - Rank their severity, risk level
  - Whether they can be used for co-residence detection
- Verify their full or partial existence on five real-world, commercial cloud container services
- Show there are security implications from these leakages
  - Infer private data, detect and verify co-residence, build covert channels, launch other cloud attacks
  - **Build two functioning covert channels and characterize their performance**
  - Design and conduct synergistic power attacks
- Determine leakage is due to incomplete coverage of container isolation in the Linux kernel
- Propose a two-stage defense against these vulnerabilities
  - Power-based namespace to deal with synergistic power attacks
  - Has effective prevention and acceptable performance overhead

Background

The Container and Its Advantages

- Containers are lightweight, virtualized, and (ideally) isolated runtime environments
- Containers share resources (host kernel, hardware, etc.); This can be a security issue
  - A virtual machine (VM) has its own virtualized OS and kernel; Not shared with other VMs or host machine
  - In certain ways, VMs can be more secure than containers
- Containers are popular because they're lightweight and achieve better performance than virtual machines



Source: https://blog.netapp.com/blogs/containers-vs-vms/

Cloud Container Environments

- Cloud providers offer many different types of services: SaaS, PaaS, IaaS
  - We will focus on multi-tenancy cloud containers (mainly **IaaS**)
- Several companies, individuals, and potential attackers can coinhabit the same resources on a cloud server
- They *should* be able to run any software they want on their container(s) and not be affected by other users
- In other words: The container infrastructure *should* be abstracted and isolated

Container Software

- Docker and LinuXContainer (LXC) were chosen for this research
  - Widely used on cloud services; work with popular container orchestrators (e.g. Kubernetes)
  - Run on the Linux kernel
- Allows for a multi-tenancy platform that shares OS kernel and hardware resources per server
  - Inherent security and privacy concerns when sharing resources with strangers
- An attacker can pose as a legitimate tenant with other customers on the same physical machine
  - This potential vulnerability is not limited to just Docker and LXC

Information Leakage

Namespaces

- Core Linux kernel features that containers rely on
  - Many containers run on the Linux kernel; important to analyze this feature
- Acts as an isolation mechanism
  - Isolate system resources into groups of processes
    - A process can be part of multiple namespace types
  - Kernel shows custom view of resources to each process, based on namespace(s)
    - E.g.: Different mount (MNT) namespaces may see different file system structures
    - Process in the same process ID (PID) namespace can see the same process IDs
  - **Changes in one namespace should not be visible to or affect another namespace**
  - **Seven** types of namespaces currently implemented

Incomplete Isolation

- Incomplete implementation of isolation measures is a real issue for containers
  - **Due to missing context checks or general lack of full namespace**
  - Can expose system-wide resources to every container
  - Power consumption, performance data, global kernel data, process scheduling, …
  - Can allow for co-residence verification, provide information to help launch DoS attacks, …

- Memory-based **pseudo file system** contains several examples of this
  - Controlled interface from user space to kernel
  - Access and edit kernel data via normal file I/O
  - There are several types, but we focus on **procfs** and sysfs
  - We used automated tool to cross-validate potential leakage from these systems in a container environment
    - **Incomplete namespace implementation on host resources were detected**

Leakage Channels

LEAKAGE CHANNELS IN COMMERCIAL CONTAINER CLOUD SERVICES

Leakage Channels	Leakage Information	Potential Vulnerability			Container Cloud Services[1]				
		Co-re	DoS	Info leak	CC_1	CC_2	CC_3	CC_4	CC_5
/proc/locks	Files locked by the kernel	●	○	●	●	●	●	●	◐
/proc/zoneinfo	Physical RAM information	●	○	●	●	●	●	●	◐
/proc/modules	Loaded kernel modules information	○	○	●	●	●	●	●	●
/proc/timer_list	Configured clocks and timers	●	○	●	●	●	●	○	●
/proc/sched_debug	Task scheduler behavior	●	○	●	○	○	●	○	●
/proc/softirqs	Number of invoked softirq handler	●	●	●	●	●	●	●	●
/proc/uptime	Up and idle time	●	○	●	●	●	●	●	◐
/proc/version	Kernel, gcc, distribution version	○	○	●	●	●	●	●	●
/proc/stat	Kernel activities	●	●	●	●	●	●	●	◐
/proc/meminfo	Memory information	●	●	●	●	●	●	●	○
/proc/loadavg	CPU and IO utilization over time	●	○	●	●	●	●	●	◐
/proc/interrupts	Number of interrupts per IRQ	●	○	●	●	●	●	●	●
/proc/cpuinfo	CPU information	●	○	●	●	●	●	●	○
/proc/schedstat	Schedule statistics	●	○	●	●	●	●	●	◐
/proc/sys/fs/*	File system information	●	○	●	●	●	○	●	●
/proc/sys/kernel/random/*	Random number generation info	●	○	●	●	●	●	●	●
/proc/sys/kernel/sched_domain/*	Schedule domain info	●	○	●	●	●	●	●	●
/proc/fs/ext4/*	Ext4 file system info	●	○	●	●	●	●	●	●
/sys/fs/cgroup/net_prio/*	Priorities assigned to traffic	○	○	●	●	●	○	○	○
/sys/devices/*	System device information	●	●	●	●	●	●	○	○
/sys/class/*	System device information	○	●	●	●	●	●	○	○

Channels exist due to perceived low priority and/or potential difficulty in isolating

UNIVERSITY OF DELAWARE

Covert Channels

- Communication between isolated systems via shared resources
- **Stealthily** transfer data from a compromised machine to the network or other attacker controlled machine
  - Attacker controlled machine is a co-resident on the same physical machine in this study
  - Allow attacker to avoid detection of data transmission and stay in system for a long time
  - Data transmission can be relatively slow
- Can also be used for co-residence detection
  - Unique static identifiers and dynamically implanted unique identifiers
  - Necessary for establishing a channel
  - Examples:

LEAKAGE CHANNELS FOR CO-RESIDENCE VERIFICATION

Leakage Channels	$\mathbb{U}$	$\mathbb{V}$	$\mathbb{M}$	Rank
/proc/sys/kernel/random/boot_id	●	○	○	
/sys/fs/cgroup/net_prio/net_prio.ifpriomap	●	○	○	
/proc/sched_debug	●	●	●	
/proc/timer_list	●	●	●	
/proc/locks	●	●	●	
/proc/uptime	●	●	◐	
/proc/stat	●	●	◐	

Covert Channels

- **Any shared resource on a system can be exploited to make one**
  - Any of the previously identified leakage channels and more
  - Co-residence verification is important because of this
- Ideally: high-bandwidth, reliable, undetectable by cloud provider and victim resident
- Varying degrees susceptibility to noise in the cloud environment

UNIVERSITY OF DELAWARE

Covert Channel Implementation Examples

Previous research has shown convert channels can be built on multi-tenancy cloud environments with:

- Shared L2 cache (0.2 bps)
- Last/lowest level caches (3.2 bps)
- Memory bus contention (110 bps)
- Thermal-based; core temperature reading (12.5 bps to 50 bps)

- All these channels are characterized by low data transfer rates
- Leakage channels identified may be able to provide more bandwidth and reliability

Constructing Convert Channels

UNIVERSITY OF DELAWARE

/proc/locks Overview

Leakage Channels	Leakage Information	Potential Vulnerability			Container Cloud Services[1]				
		Co-re	DoS	Info leak	CC_1	CC_2	CC_3	CC_4	CC_5
`/proc/locks`	Files locked by the kernel	●	○	●	●	●	●	●	◐

- Unique, dynamic identifier that can be manipulated
- /proc/locks is a pseudo-file that displays the all current file locks the kernel has in place
  - File locks prevent race conditions with file I/O
  - Used for debugging
  - Type, process ID, inode number
- Data is not fully protected by namespaces
  - Container can see locks belonging to the host and other containers

/proc/locks Format

```
user@kali:~$ cat /proc/locks
1: POSIX  ADVISORY  WRITE 1039 08:01:3148877 0 EOF
2: POSIX  ADVISORY  WRITE 1029 08:01:3148660 0 EOF
3: POSIX  ADVISORY  WRITE 1024 08:01:3148659 0 EOF
4: POSIX  ADVISORY  READ  1840 08:01:4462056 128 128
5: POSIX  ADVISORY  READ  1840 08:01:4461952 1073741826 1073742335
6: POSIX  ADVISORY  READ  1837 08:01:4462056 128 128
7: POSIX  ADVISORY  READ  1837 08:01:4461952 1073741826 1073742335
8: POSIX  ADVISORY  WRITE 1043 08:01:3148878 0 EOF
9: POSIX  ADVISORY  WRITE 519 08:01:1978486 0 EOF
10: POSIX  ADVISORY  WRITE 519 08:01:1977188 0 EOF
11: POSIX  ADVISORY  WRITE 519 08:01:1977187 0 EOF
12: POSIX  ADVISORY  READ  1833 08:01:4462056 128 128
13: POSIX  ADVISORY  READ  1833 08:01:4461952 1073741826 1073742335
14: POSIX  ADVISORY  READ  1843 08:01:4462056 128 128
15: POSIX  ADVISORY  READ  1843 08:01:4461952 1073741826 1073742335
16: POSIX  ADVISORY  WRITE 1034 08:01:3148876 0 EOF
17: FLOCK  ADVISORY  WRITE 512 00:14:13853 0 EOF
```

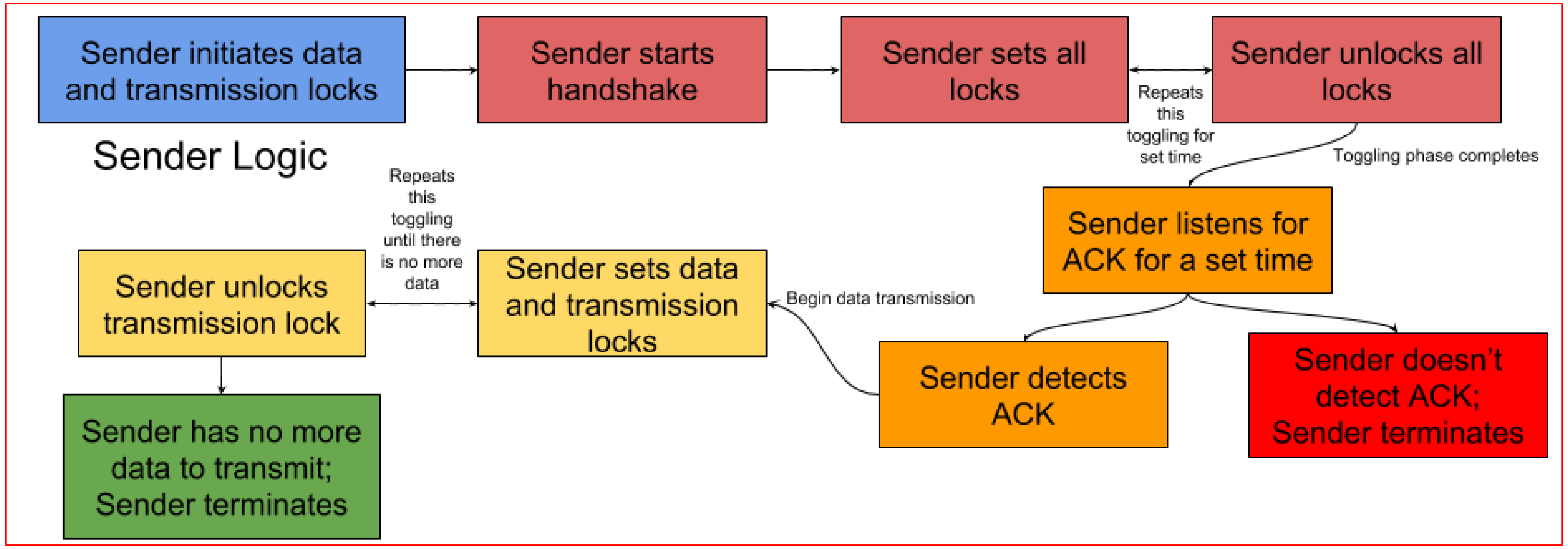
/proc/locks format is:

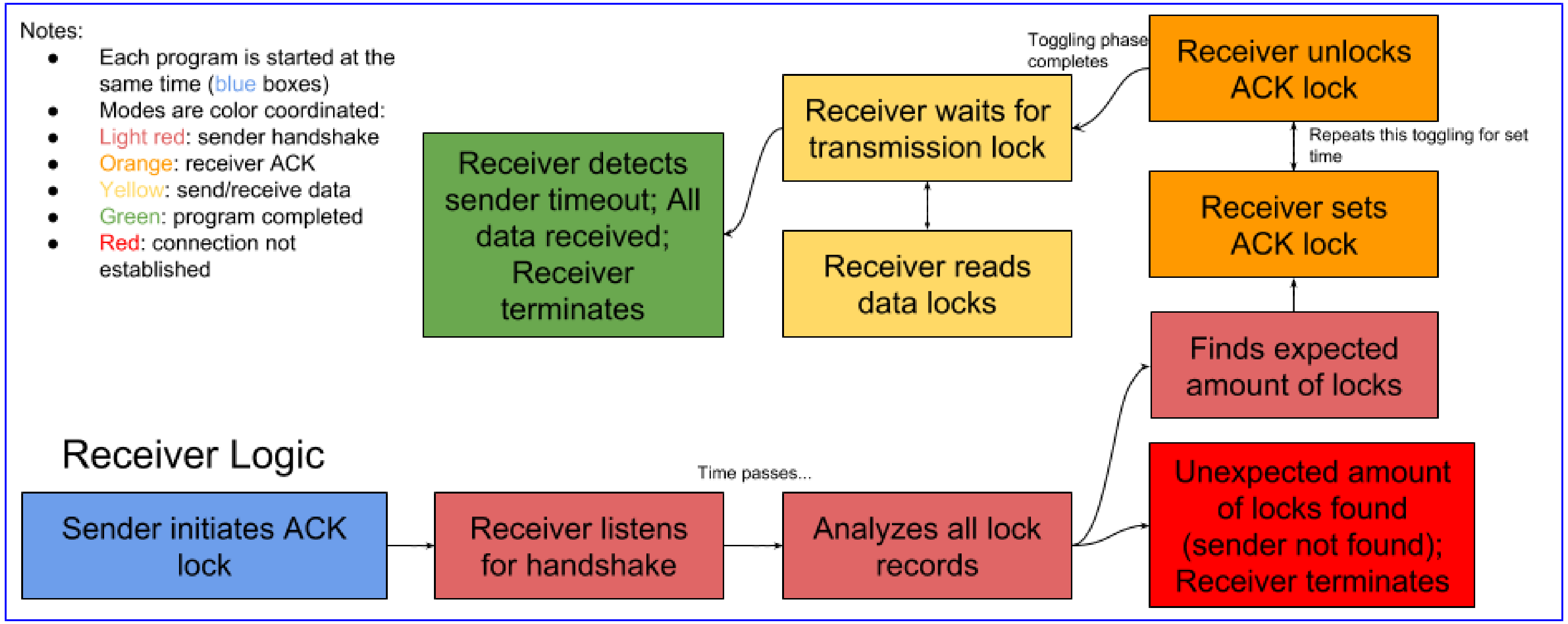
- `Index number: | Class of lock (FLOCK or POSIX) | ADVISORY or MANDATORY |READ or WRITE | PID | Lock ID MAJOR-DEVICE:MINOR-DEVICE:`**`INODE-NUMBER`**` | Start and end of locked region`
- Source: https://www.centos.org/docs/5/html/5.2/Deployment_Guide/s2-proc-locks.html
- `Information updates in real-time`
- As of Linux 4.9, PID is set to 0 if outside current PID namespace

/proc/locks Covert Channel Operation

- Sender: Victim container that is controlled by the attacker
- Receiver: Container controlled by attacker that is a co-resident of same physical machine
- Assume: Attacker already verified sender and receiver are co-residents
- Transmits data in “binary”

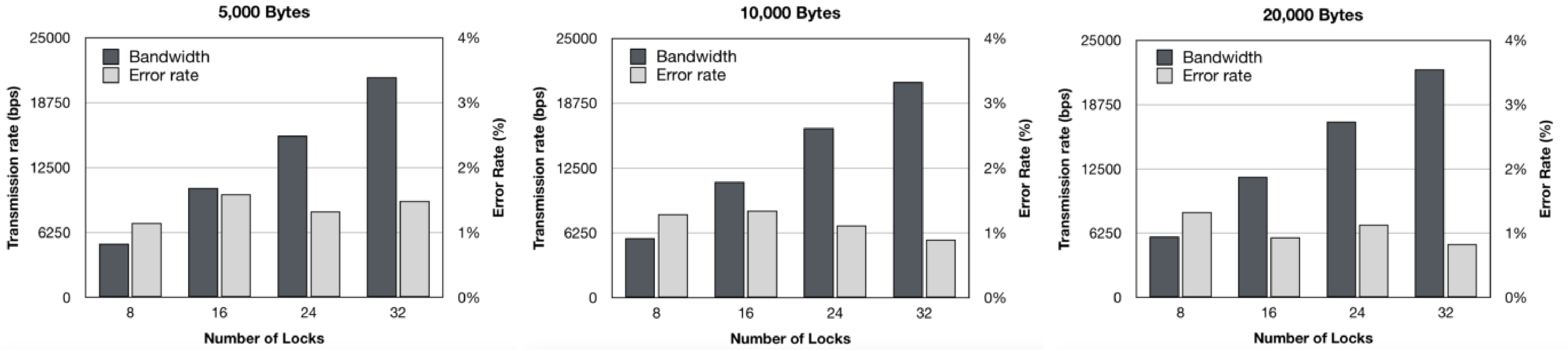
- Placing a lock on a file represent a “1”; Unlocking file represent a “0”
  - Adds or removes unique entry (inode #) to /proc/locks
  - Files are not visible to each container, but unique lock entries will appear in /proc/locks
    - “Inode numbers are guaranteed to be unique only within a filesystem” – `man inode`
    - Does not change for the same given file
  - The sender and receiver do a TCP-like handshake to identify each other’s locks and synchronize
  - Receiver checking if known lock is present in /proc/locks means 1 is sent, 0 if lock is not present
- Multiple locks can used to transmit multiple bits in parallel


Sender initiates data and transmission locks
Sender starts handshake
Sender sets all locks
Repeats this toggling for set time
Sender unlocks all locks
Sender Logic
Toggling phase completes
Sender listens for ACK for a set time
Repeats this toggling until there is no more data
Sender unlocks transmission lock
Sender sets data and transmission locks
Begin data transmission
Sender detects ACK
Sender doesn't detect ACK; Sender terminates
Sender has no more data to transmit; Sender terminates


Notes:
• Each program is started at the same time (blue boxes)
• Modes are color coordinated:
• Light red: sender handshake
• Orange: receiver ACK
• Yellow: send/receive data
• Green: program completed
• Red: connection not established
Toggling phase completes
Receiver unlocks ACK lock
Receiver waits for transmission lock
Repeats this toggling for set time
Receiver detects sender timeout; All data received; Receiver terminates
Receiver sets ACK lock
Receiver reads data locks
Finds expected amount of locks
Receiver Logic
Time passes...
Unexpected amount of locks found (sender not found); Receiver terminates
Sender initiates ACK lock
Receiver listens for handshake
Analyzes all lock records

/proc/locks Covert Channel Performance

- Tested channel in a real multitenant cloud environment
- Cloud environments are noisy: many programs can also be holding/releasing locks
- More locks used in a transmission = more bandwidth
  - More locks = less covert? Limited benefit to increasing locks?
- Low error rate thanks to TCP-like transmission
- Very high bandwidth for a covert channel



Bandwidth and error rate of the lock-based covert channel.

/proc/meminfo Overview

Leakage Channels	Leakage Information	Potential Vulnerability			Container Cloud Services[1]				
		Co-re	DoS	Info leak	CC_1	CC_2	CC_3	CC_4	CC_5
/proc/meminfo	Memory information	●	●	●	●	●	●	●	○

- Performance data that can be manipulated
- /proc/meminfo is a pseudo-file that displays memory usage information
  - Info reflects system wide performance
  - Containers can see the total memory resources of the entire host machine
  - Memory allocated/freed in one container will be visible to others
- A unique workload pattern using memory alloc/free can be seen by co-residents

- **Inherently noisy**: many Linux processes, resident apps are using memory all the time

UNIVERSITY OF DELAWARE

/proc/meminfo Format

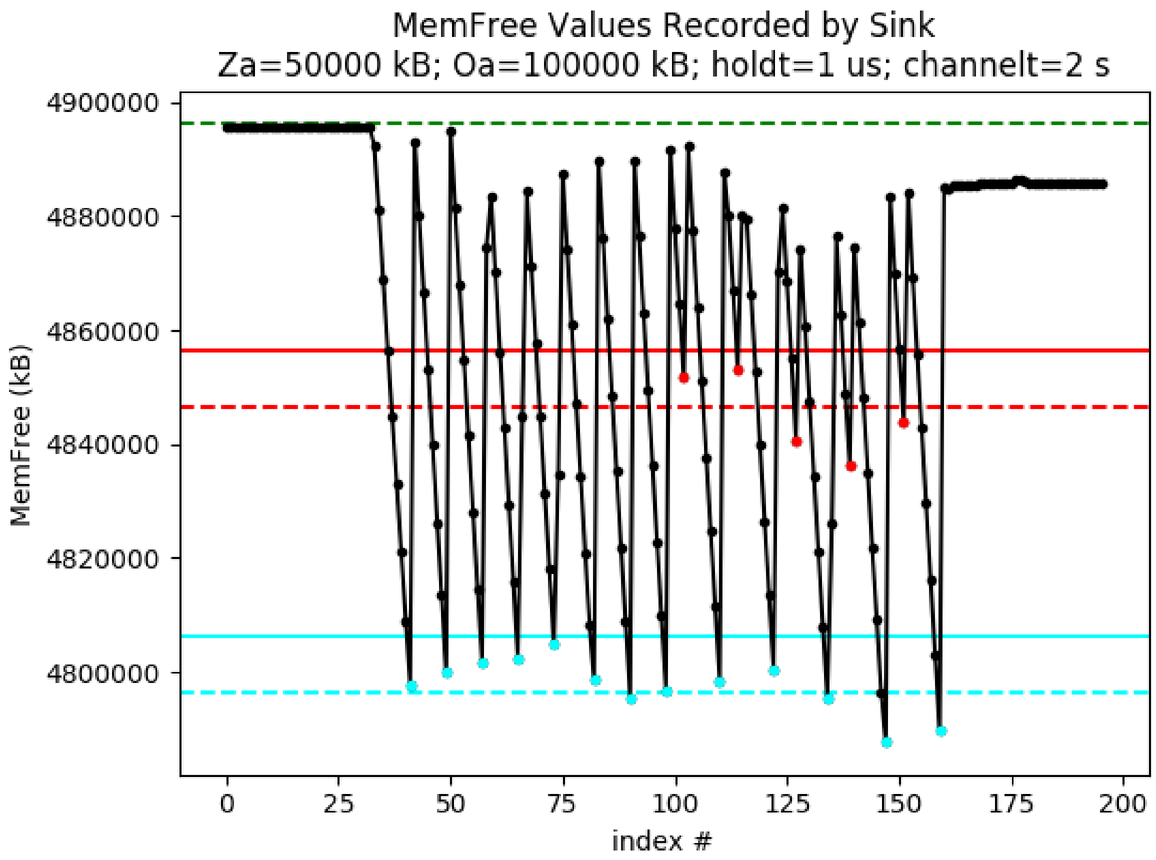
- Contains *lots* of information about memory usage
- Focus on **MemFree**, current amount of physical, unused RAM that is ready to be allocated
  - We could have chosen other values
  - Trade off between easy manipulation and how noisy the value will be
- Updates in real-time, so values are very accurate

```
user@kali:~$ cat /proc/meminfo
MemTotal:        4042988 kB
MemFree:         2757484 kB
MemAvailable:    2987440 kB
Buffers:           56236 kB
Cached:           363456 kB
SwapCached:            0 kB
Active:           926364 kB
Inactive:         215504 kB
Active(anon):     722812 kB
Inactive(anon):     6088 kB
Active(file):     203552 kB
```

/proc/meminfo Covert Channel Operation

- Sender: Victim container that is controlled by the attacker
- Receiver: Container controlled by attacker that is a co-resident of same physical machine
- Assume: Attacker already verified sender and receiver are co-residents
- Transmits data in “binary”

- Containers get baseline MemFree value over a period of time
  - Little fluctuation tells us a stable channel is possible
- Sender then allocates a large amount of memory to represent a transmission bit
  - E.g.: 100 MiB for “1” and 50 MiB “0”
  - The value of MemFree immediately drops
- Receiver detects change in MemFree, determines difference between baseline value, and decodes 1 or 0
- Sender frees allocated memory; MemFree returns to the baseline value for next transmission
- A single bit is transmitted
- Sender transmits fixed pattern before actual data; very simple “handshake”



- **Source=sender, Sink=receiver**
- **Za**: Zero alloc; The amount of memory the source is allocating each time it sends a 0.
- **Oa**: One alloc; The amount of memory the source is allocating each time it sends a 1.
- **holdt**: hold time; The amount of time the source is keeping a chunk of memory allocated before it frees it. The source also waits this amount of time after freeing memory before it allocates a new chunk.
- **channelt**: channel time; The amount of time the sink will record MemFree values after the calibration is done.

/proc/meminfo Covert Channel Performance

- Cloud environments are noisy, especially for memory usage
- Channel will be more successful in low-noise environment with sender allocating/freeing large chunks
- Large values make channel more robust, but decreases bandwidth
  - Inversely proportional
  - Very small allocations more susceptible to noise; handshake may fail
- Could include checksums, hamming code in transmission for data correction; decreases bandwidth
- Have different allocation levels, more bits per transmission
- Running average to detect gradual change in baseline value

BANDWIDTH FOR MEMINFO BASED COVERT CHANNELS

Bit 1 (kb)	Bit 0 (kb)	Bandwidth (bps)	Error Rate
100,000	50,000	8.79565	0.3%
90,000	45,000	9.72187	0.5%
80,000	40,000	11.0941	0.4%
70,000	35,000	12.7804	0.8%
65,000	35,000	13.603	0.5%

UNIVERSITY OF DELAWARE

Results, Defense, and Evaluation

University of Delaware

Overall Results

- Lock-based channel is high bandwidth and robust on real cloud environments
- Meminfo is susceptible to noise, lower bandwidth, but is able to transmit in an ideal environment
- Only real requirement is for containers to be co-residents on the same physical server
- Other channels require either: being on same CPU package, be on nearby cores, or have low real world success
- Many improvements can be made to our implementations (increase bandwidth and reliability)

COMPARISON AMONG DIFFERENT COVERT CHANNELS

Method	Bandwidth (bps)	Error Rate
Lock (8 locks)	5149.9	1.14%
Meminfo	13.603	0.5%
Cache [32]	751	3.1%
Memory bus [42]	107.9±39.9	0.75%
Memory deduplication [43]	80	0%
Thermal [11]	45	1%

Defense Strategies

- In the Linux kernel, incomplete implementations of namespaces is a main cause of information leakage channels in multitenant container
- To defend against this, a **two stage defense mechanism** is proposed:
  - 1) Masking the channels
  - 2) Enhancing container resource isolation model

Stage 1: Masking the channels

- System admins can deny access to identified channels in a container via security policies
- **Pros**:
  - No changes to kernel source code needed
  - Immediately stops leakage
- **Cons**:
  - Legitimate applications may depend on this information
    - Undermines idea of container: virtual platform with native runtime environment

Defense Strategies

Stage 2: Enhancing container's resource isolation model

- Fix any missing namespace context checks and implement new namespaces to virtualize more system resources
- Authors already reported some shortcoming: CVE-2017-5967 [LINK]
  - Patched quickly since namespace already existed
- Other channels identified have no current namespace implementations
- **Pros**:
  - Some fixes can be quick; e.g.: missing context check
  - Complete namespace implementation would be effective at stopping leakage channel
    - "The ideal solution"
- **Cons**:
  - Significant effort to create new, full namespace implementation (started in 2002, Linux 2.4.19)
  - Can affect multiple kernel subsystems
  - Some system resources are not easily adapted into namespaces
    - e.g.: interrupts, scheduling information, temperature

Power-based Namespace Case Study

- Proof-of-concept namespace added to the Linux kernel for power-based information
- Blocks leakage channels that could cause synergetic power attacks
  - Presents only relevant power usage information to a to given container
  - **Results can be generalized for any type of leakage channel**
- New namespace was evaluated for:
  - **Accuracy**: 5% error or less when reporting energy usage information
    - Within improvement, a new namespaces will obtain equal accuracy as non-namespace
  - **Security**: Container A could no longer see the effects of Container B's power consumption
    - Proves new namespaces can be effective at blocking leakage channels between containers
  - **Performance**: 9.66% overhead on single thread; 7.03% overhead on multi-threads (8 threads)
    - Addition of namespace with introduce an acceptable overhead
    - Optimizations to others subsystems could decrease overhead

Again: Overview of Findings

Using common container software, we:

- Systematically identify many potential and realized **leakage channels in multitenant cloud container environments**
  - Rank their severity, risk level
  - Whether they can be used for co-residence detection
- Verify their full or partial existence on five real-world, commercial cloud container services
- **Show there are security implications from these leakages**
  - Infer private data, detect and verify co-residence, build covert channels, launch other cloud attacks
  - **Build two functioning covert channels and characterize performance**
  - Design and conduct synergistic power attacks
- Determine leakage is due to **incomplete coverage of container isolation in the Linux kernel**
- **Propose a two-stage defense against these vulnerabilities**
  - Power-based namespace to deal with synergistic power attacks
  - Effective prevention and acceptable performance overhead

Conclusion

- Multitenant cloud containers using a shared Linux kernel contain leakage channels
  - allow for co-resident verification, covert channels, and other attacks
- “The root cause for information leakage … is the incomplete implementation of the isolation mechanisms in the Linux kernel.”
  - Need for more complete or new namespaces
- New namespaces are difficult, time consuming, and in contention with security, performance, and usability
  - They are effective at stopping these leakages

- A Study on the Security Implications of Information Leakages in Container Clouds
  - Journal: *IEEE Transactions on Dependable and Secure Computing*
  - Journal website link: https://ieeexplore.ieee.org/document/8523802
- `/proc/locks` and `/proc/meminfo` covert channel source code:
  - https://github.com/bsteen/cloud-covert-channels