CLARION: Sound and Clear Provenance Tracking for Microservice Deployments

XUTONG CHEN[1], HASSAAN IRSHAD[2],
YAN CHEN[1], ASHISH GEHANI[2],
VINOD YEGNESWARAN[2]

NORTHWESTERN UNIVERSITY[1],
SRI INTERNATIONAL[2]

Microservices, Containers and Provenance Tracking

Microservices

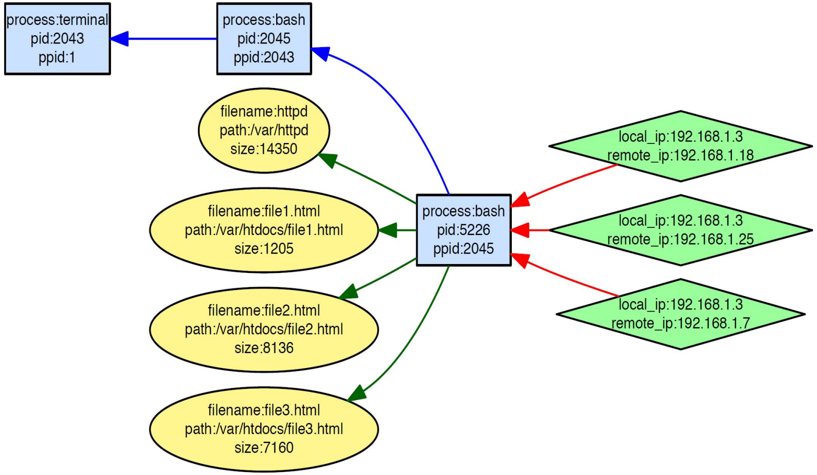
- Enable better resource utilization
- Increasing practical adoption

Containers

- Portable and lightweight software isolation technique
- New security challenges

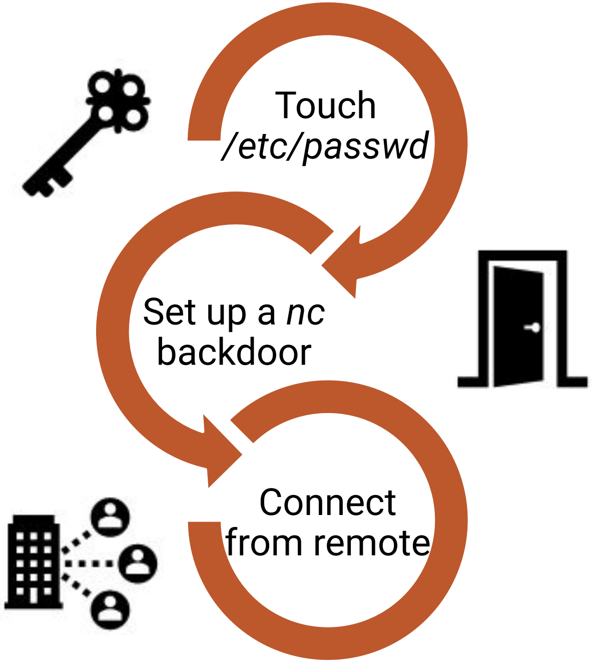
Provenance Tracking

- State-of-the-art host forensic monitoring solution
- Transferred from traditional OS scenario to container scenario



A sample provenance graph generated by SPADE, Middleware'12

Motivating Example: Insider Attack



Touch */etc/passwd*

Set up a *nc* backdoor

Connect from remote

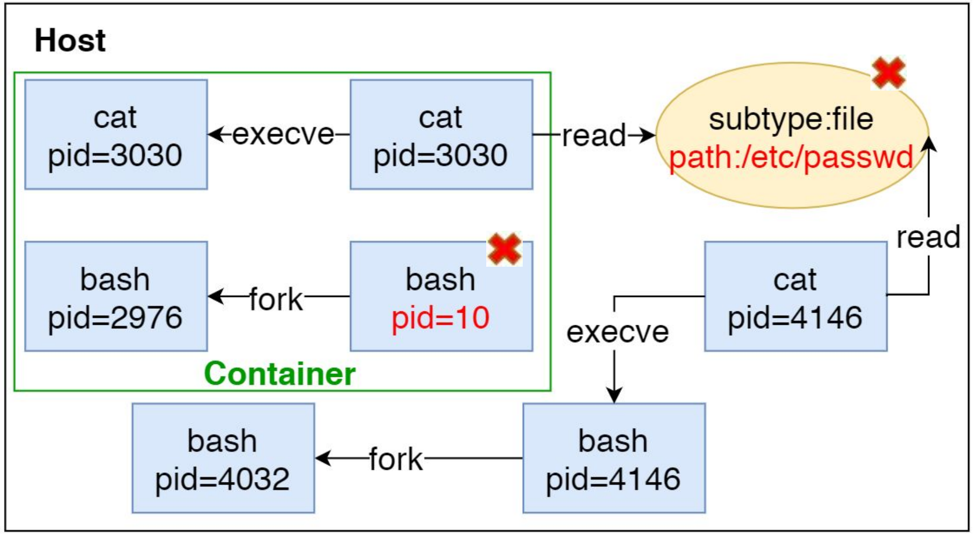
- Three major steps of a trivial insider attack is shown on the left.
- Perform all three steps in both the host and the container.
- We drill down just on the first step in the next slide for simplicity.
- Motivate our work by illustrating limitations in the provenance graphs from three contemporary solutions.

Existing Provenance Tracking Solutions

Namespace-unaware
Container-aware
Winnower: NDSS 2018

Fail on soundness

- Fragmentation in *bash-cat* provenance
- Ambiguities on */etc/passwd*



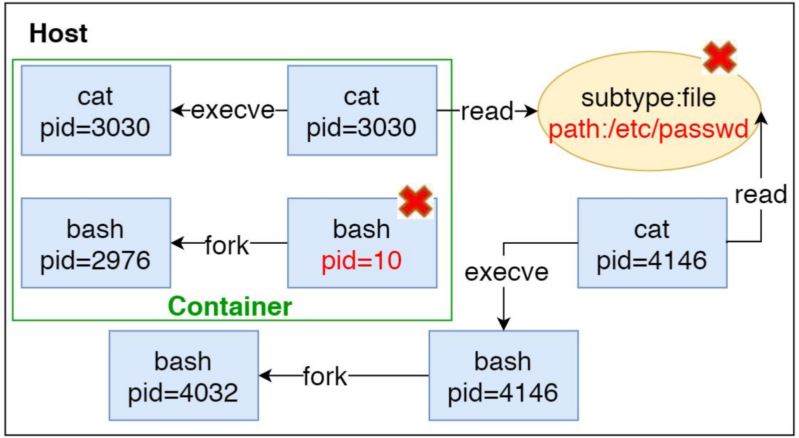
Existing Provenance Tracking Solutions

Namespace-unaware Container-aware

Winnower: NDSS 2018

Fail on soundness

- Fragmentation in *bash-cat* provenance
- Ambiguities on */etc/passwd*

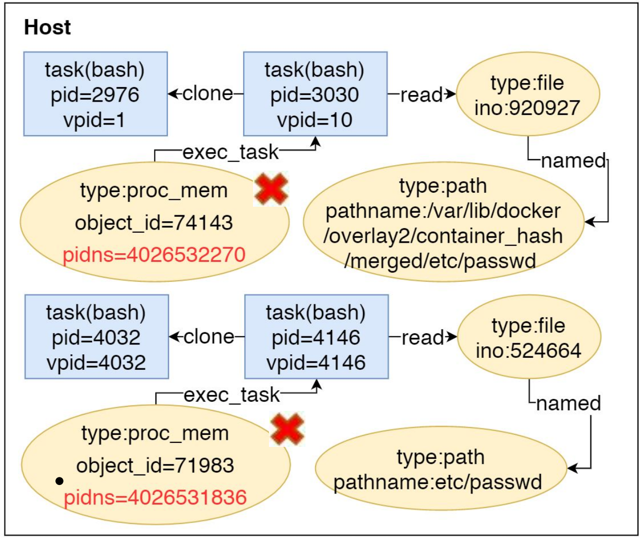


Namespace-aware Container-unaware

Camflow: SoCC 2017

Fail on clarity

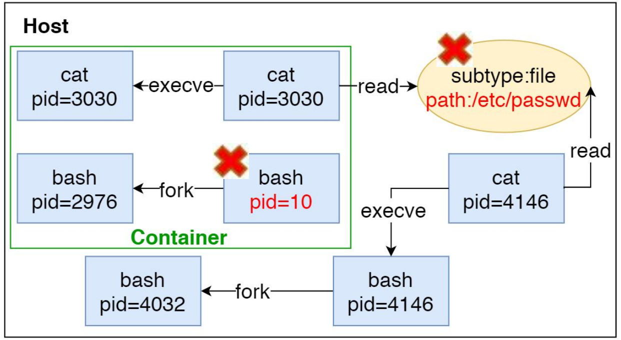
- Unclear insight about the container



Existing Provenance Tracking Solutions

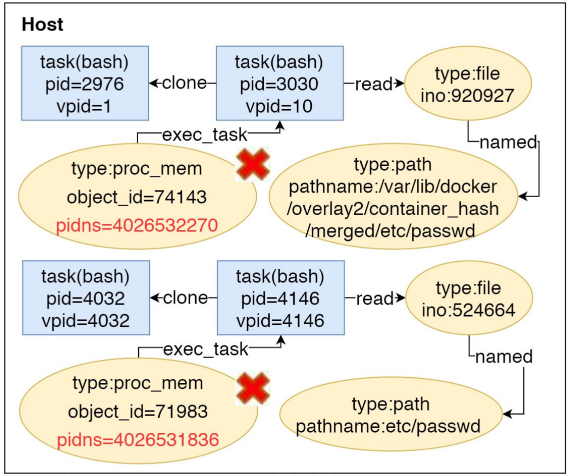
Namespace-unaware
Container-aware
Winnower: NDSS 2018
Fail on soundness

- Fragmentation in *bash-cat* provenance
- Ambiguities on */etc/passwd*

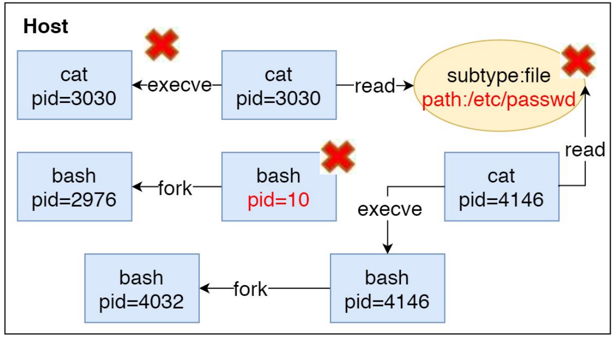


Namespace-aware
Container-unaware
Camflow: SoCC 2017
Fail on clarity

- Unclear insight about the container

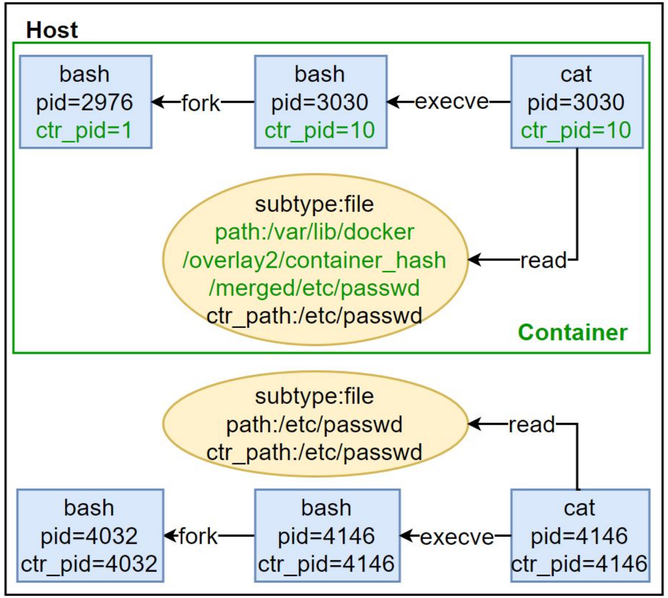


Namespace-unaware
Container-unaware
SPADE: Middleware 2012
Fail on both soundness and clarity



CLARION Solution and Key Contributions

Namespace-aware
Container-aware



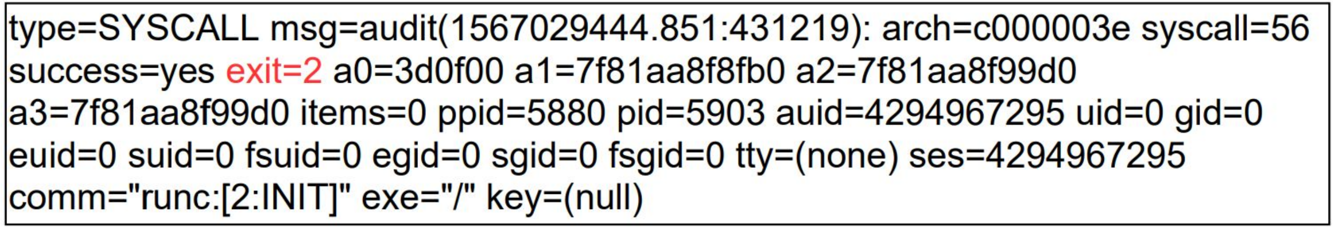
- The first in-depth analysis of the implications of namespaces on provenance tracking.
- CLARION: A **namespace-aware** and **container-aware** provenance tracking solution.
  - Extension of the SPADE Provenance Engine to address clarity and soundness issues.
  - Linux kernel module, netfilter hooks ,and modifications to SPADE application-level handlers.
- Comprehensive evaluation of the effectiveness, efficiency, and generality.

Soundness Challenge:
Inconsistency in low-level events

- *Inconsistency in low-level events* occur when low-level events report data values from varying (host / container) contexts that lead to vertex splitting in provenance graphs.

- Consider the “clone” Linux audit event as an example, its “pid” field (5903) is in host context but its “exit” field (2), which is also a PID, is affected by pid namespace and in container context.

- If provenance tracking system directly uses those two data fields to generate provenance, *graph fragmentation* will occur.



```
type=SYSCALL msg=audit(1567029444.851:431219): arch=c000003e syscall=56
success=yes exit=2 a0=3d0f00 a1=7f81aa8f8fb0 a2=7f81aa8f99d0
a3=7f81aa8f99d0 items=0 ppid=5880 pid=5903 auid=4294967295 uid=0 gid=0
euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=(none) ses=4294967295
comm="runc:[2:INIT]" exe="/" key=(null)
```

Soundness Challenge:
In-depth Analysis of Namespace Implication

Namespace	**Virtualized System Resource**	**What low-level events are affected?**	**Impact on soundness**
PID	Process IDs	Events with PIDs	Yes, fragmentation
Mount	Mount points	Events with file pathnames	Yes, ambiguities
Network	Network device/stack, etc.	Events with Local IPs/ports	Yes, both
IPC	SYSV IPC objects & POSIX msg queue	Events with ID/names of SYSV IPC object/POSIX msg queues	Yes, ambiguities
User	User/group IDs	Events with GIDs and UIDs	No

Time, UTS, Cgroup	Boot/ monotonic clocks;
	Host/NIS domain name;
	Cgroup root directory
		Does not affect provenance dataflow	N/A

PID/Mount/Network/IPC ns can impact soundness. Additional details provided in paper (Tables 1 and 2).

Soundness Challenge is not specific to Linux Audit

Question: *Does soundness challenge persist on other Linux auditing subsystems/OS platforms?*

Namespace	**Sysdig**	**LTTng**
PID	Yes	*No*
Mount	Yes	Yes
Network	Yes	Yes
IPC	Yes	Yes

Namespace	**BSD Jail**	**Solaris Zone**
PID	*No*	*No*
Mount	Yes	Yes
Network	Yes	Yes
IPC	Yes	Yes

- Soundness challenge persists in other Linux auditing subsystems and OS platforms.
- LTTng provides virtualized PIDs and host PIDs together so fragmentation can be solved by correlation.
- No fragmentation is caused by PID in BSD Jail/Solaris Zone because all PIDs are host PIDs.
- Additional details provided in paper (Table 3 and 4).

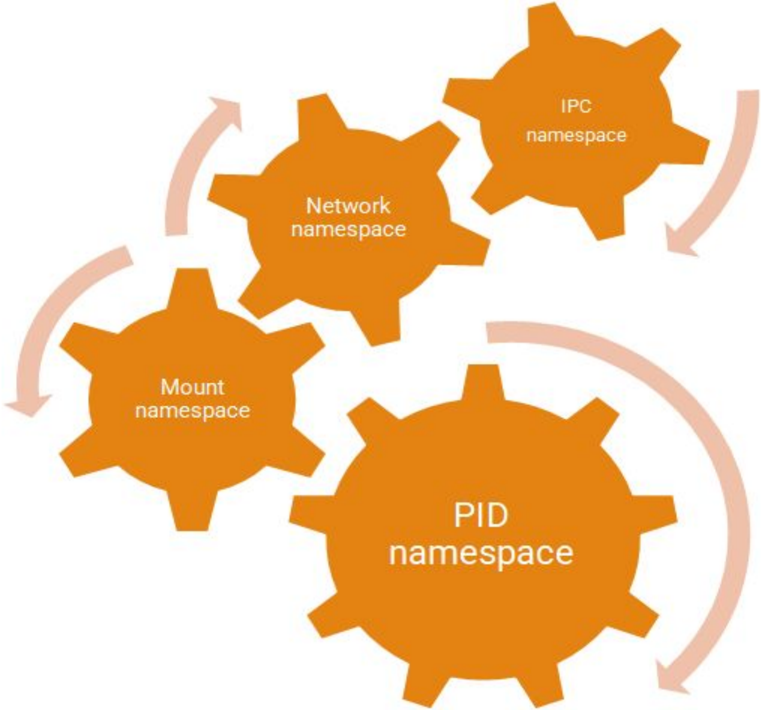
Soundness Challenge: Mapping Virtualized Namespaces

- *Key design*: establish a mapping between the host view and the container view on virtualized resources.

- The mapping correlates the virtualized data with the host data.

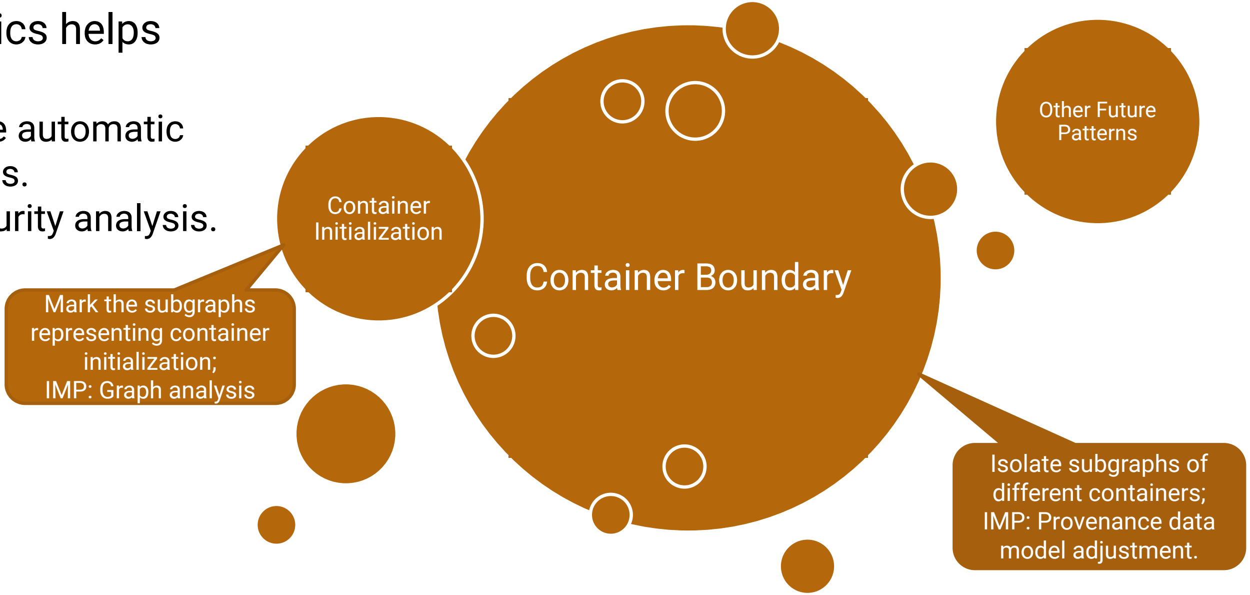
- It helps provenance tracking system to use the **consistent** provenance data.

- The principal namespaces that impact provenance are PID, Mount, Network, and IPC.



Clarity: Essential Container Semantic Patterns

- Understanding essential container semantics helps by:
  1. accelerating the automatic analysis process.
  2. simplifying security analysis.



Mark the subgraphs representing container initialization;
IMP: Graph analysis

Isolate subgraphs of different containers;
IMP: Provenance data model adjustment.

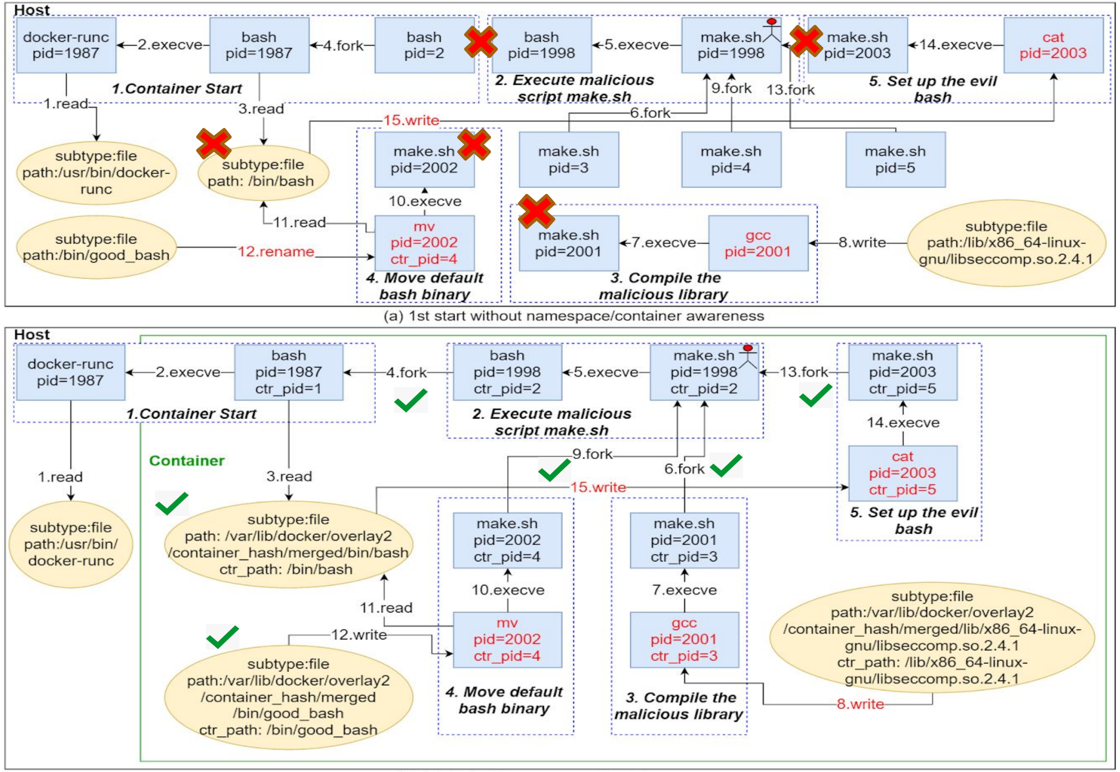
Effectiveness: Real-world Exploit

- Question: *How effective is CLARION in dealing with real-world container microservice attacks?*
- Answer: Real-world exploit

CVE #	Description	Severity
2019-5736	runC related	High. Achieve privilege escalation.
2019-14271	docker-tar related	High. Achieve privilege escalation.
2018-15664	docker-cp related	Normal. Achieve container host file system modification.

- We will focus on CVE 2019-5736 (runC) to show the effectiveness:
  - The exploit involves 2 starts of the compromised container.
  - The first start does the configuration.
  - The second start does the actual damage.

Effectiveness: CVE 2019-5736

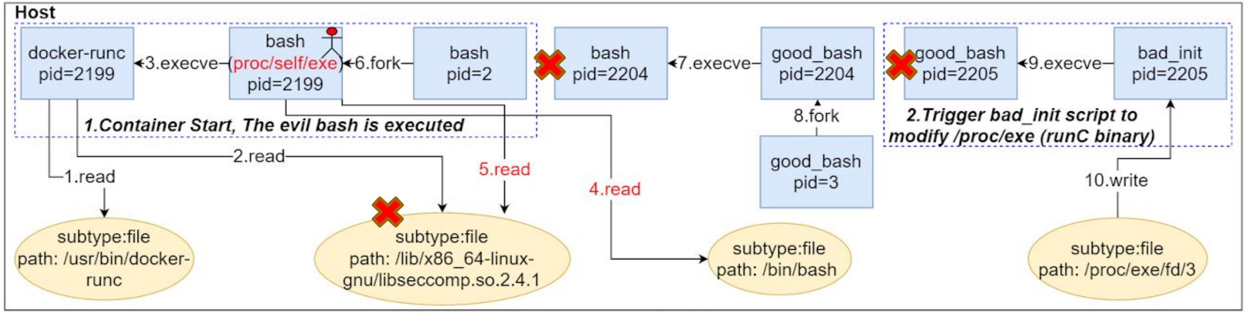


Host
docker-runc pid=1987
2.execve
bash pid=1987
4.fork
bash pid=2
bash pid=1998
5.execve
make.sh pid=1998
make.sh pid=2003
14.execve
cat pid=2003
1.Container Start
2. Execute malicious script make.sh
9.fork
13.fork
5. Set up the evil bash
1.read
3.read
6.fork
15.write
subtype:file path:/usr/bin/docker-runc
subtype:file path: /bin/bash
make.sh pid=2002
make.sh pid=3
make.sh pid=4
make.sh pid=5
10.execve
11.read
mv pid=2002 ctr_pid=4
make.sh pid=2001
7.execve
gcc pid=2001
8.write
subtype:file path:/lib/x86_64-linux-gnu/libseccomp.so.2.4.1
subtype:file path:/bin/good_bash
12.rename
4. Move default bash binary
3. Compile the malicious library
Host
Container
docker-runc pid=1987
2.execve
bash pid=1987 ctr_pid=1
4.fork
bash pid=1998 ctr_pid=2
5.execve
make.sh pid=1998 ctr_pid=2
13.fork
make.sh pid=2003 ctr_pid=5
1.Container Start
2. Execute malicious script make.sh
14.execve
cat pid=2003 ctr_pid=5
9.fork
6.fork
1.read
3.read
15.write
5. Set up the evil bash
subtype:file path:/usr/bin/docker-runc
subtype:file path: /var/lib/docker/overlay2 /container_hash/merged/bin/bash ctr_path: /bin/bash
make.sh pid=2002 ctr_pid=4
make.sh pid=2001 ctr_pid=3
10.execve
7.execve
11.read
12.write
mv pid=2002 ctr_pid=4
gcc pid=2001 ctr_pid=3
subtype:file path:/var/lib/docker/overlay2 /container_hash/merged/lib/x86_64-linux-gnu/libseccomp.so.2.4.1 ctr_path: /lib/x86_64-linux-gnu/libseccomp.so.2.4.1
subtype:file path:/var/lib/docker/overlay2 /container_hash/merged /bin/good_bash ctr_path: /bin/good_bash
4. Move default bash binary
3. Compile the malicious library
8.write

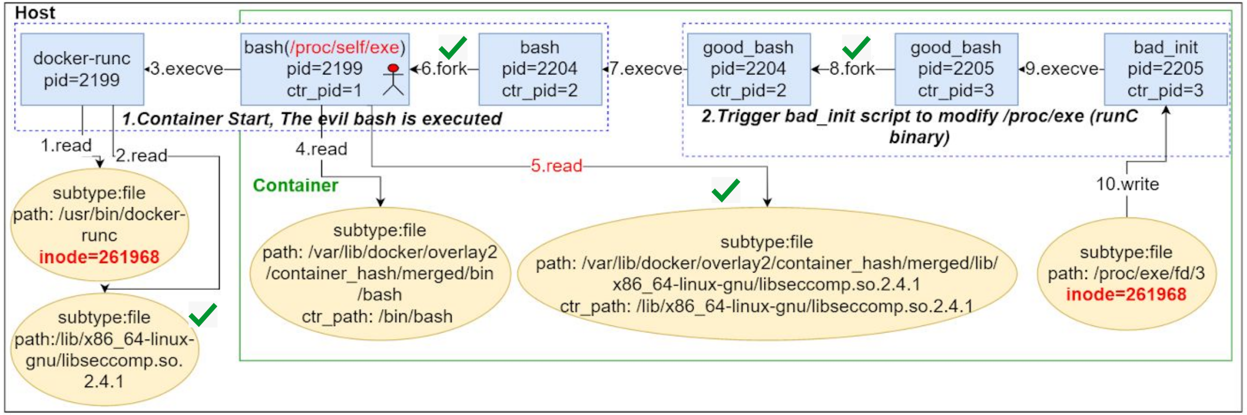
(a) 1st start without namespace/container awareness

(b) 1st start with namespace/container awareness

Effectiveness: CVE 2019-5736


Host
docker-runc pid=2199
3.execve
bash (proc/self/exe) pid=2199
6.fork
bash pid=2
bash pid=2204
7.execve
good_bash pid=2204
8.fork
good_bash pid=3
good_bash pid=2205
9.execve
bad_init pid=2205
1.Container Start, The evil bash is executed
2.Trigger bad_init script to modify /proc/exe (runC binary)
1.read
2.read
5.read
4.read
10.write
subtype:file path: /usr/bin/docker-runc
subtype:file path: /lib/x86_64-linux-gnu/libseccomp.so.2.4.1
subtype:file path: /bin/bash
subtype:file path: /proc/exe/fd/3

(a) 2nd start without namespace/container awareness

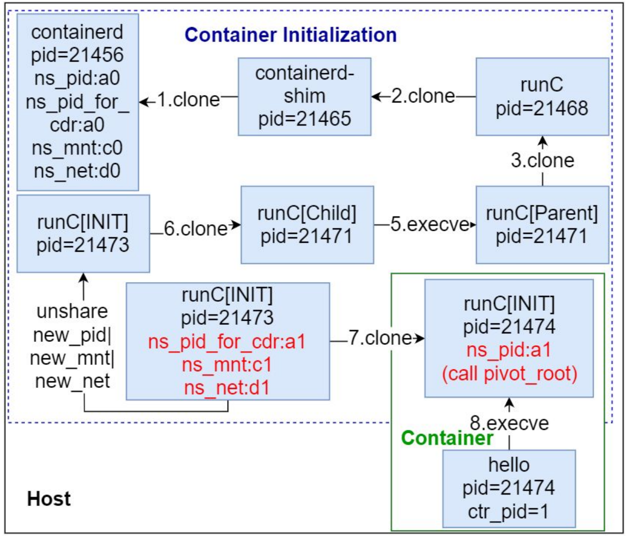

Host
Container
docker-runc pid=2199
3.execve
bash(/proc/self/exe) pid=2199 ctr_pid=1
6.fork
bash pid=2204 ctr_pid=2
7.execve
good_bash pid=2204 ctr_pid=2
8.fork
good_bash pid=2205 ctr_pid=3
9.execve
bad_init pid=2205 ctr_pid=3
1.Container Start, The evil bash is executed
2.Trigger bad_init script to modify /proc/exe (runC binary)
1.read
2.read
4.read
5.read
10.write
subtype:file path: /usr/bin/docker-runc inode=261968
subtype:file path:/lib/x86_64-linux-gnu/libseccomp.so.2.4.1
subtype:file path: /var/lib/docker/overlay2/container_hash/merged/bin/bash ctr_path: /bin/bash
subtype:file path: /var/lib/docker/overlay2/container_hash/merged/lib/x86_64-linux-gnu/libseccomp.so.2.4.1 ctr_path: /lib/x86_64-linux-gnu/libseccomp.so.2.4.1
subtype:file path: /proc/exe/fd/3 inode=261968

(b) 2nd start with namespace/container awareness

Generality: Container Initialization

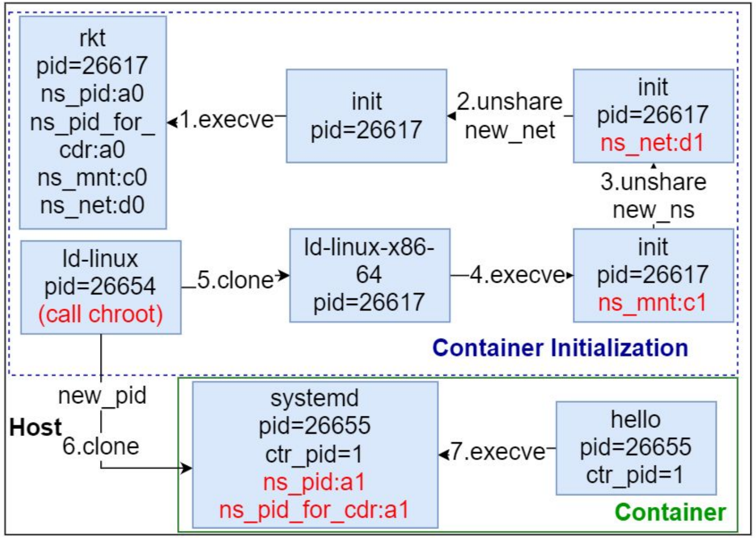
Question: *Is CLARION solution generally applicable across different container engines?*
Answer: Container initialization graph generation and quantitative results for different containers

Docker Hello-world Init



Container Initialization
containerd pid=21456 ns_pid:a0 ns_pid_for_cdr:a0 ns_mnt:c0 ns_net:d0
1.clone
containerd-shim pid=21465
2.clone
runC pid=21468
3.clone
runC[INIT] pid=21473
6.clone
runC[Child] pid=21471
5.execve
runC[Parent] pid=21471
unshare new_pid| new_mnt| new_net
runC[INIT] pid=21473 ns_pid_for_cdr:a1 ns_mnt:c1 ns_net:d1
7.clone
runC[INIT] pid=21474 ns_pid:a1 (call pivot_root)
8.execve
Container
hello pid=21474 ctr_pid=1
Host

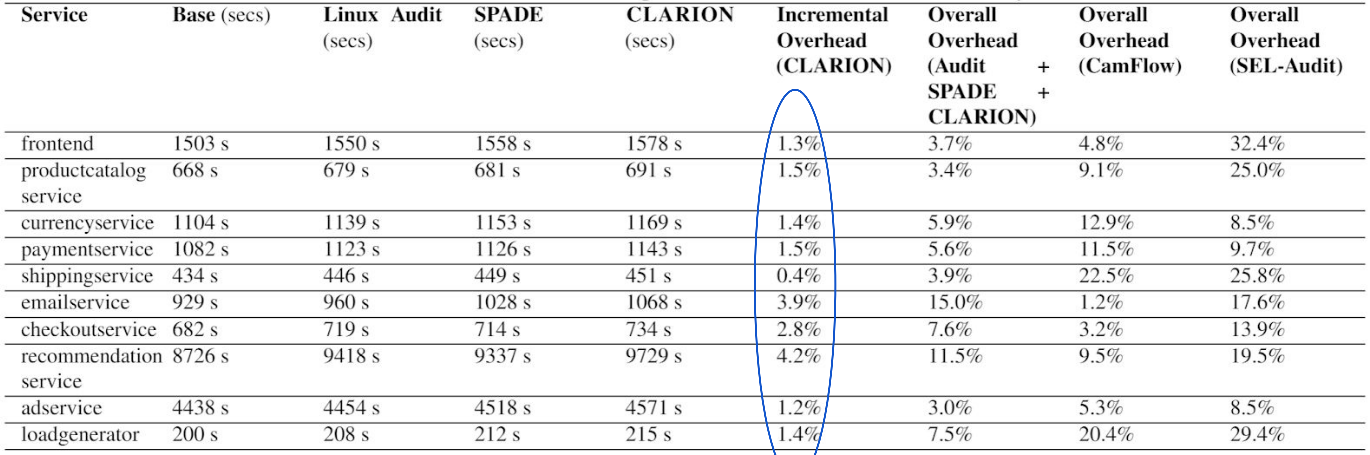
Rkt Hello-world Init



rkt pid=26617 ns_pid:a0 ns_pid_for_cdr:a0 ns_mnt:c0 ns_net:d0
1.execve
init pid=26617
2.unshare new_net
init pid=26617 ns_net:d1
3.unshare new_ns
init pid=26617 ns_mnt:c1
4.execve
ld-linux-x86-64 pid=26617
5.clone
ld-linux pid=26654 (call chroot)
Container Initialization
new_pid
Host
6.clone
systemd pid=26655 ctr_pid=1 ns_pid:a1 ns_pid_for_cdr:a1
7.execve
hello pid=26655 ctr_pid=1
Container

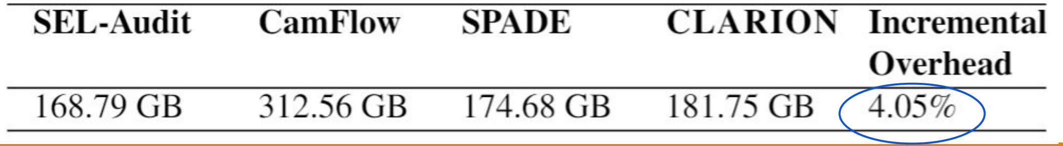
Efficiency: Runtime/Storage Overhead

Runtime Overhead Comparison of Container Provenance Systems



Service	Base (secs)	Linux Audit (secs)	SPADE (secs)	CLARION (secs)	Incremental Overhead (CLARION)	Overall Overhead (Audit + SPADE + CLARION)	Overall Overhead (CamFlow)	Overall Overhead (SEL-Audit)
frontend	1503 s	1550 s	1558 s	1578 s	1.3%	3.7%	4.8%	32.4%
productcatalog service	668 s	679 s	681 s	691 s	1.5%	3.4%	9.1%	25.0%
currencyservice	1104 s	1139 s	1153 s	1169 s	1.4%	5.9%	12.9%	8.5%
paymentservice	1082 s	1123 s	1126 s	1143 s	1.5%	5.6%	11.5%	9.7%
shippingservice	434 s	446 s	449 s	451 s	0.4%	3.9%	22.5%	25.8%
emailservice	929 s	960 s	1028 s	1068 s	3.9%	15.0%	1.2%	17.6%
checkoutservice	682 s	719 s	714 s	734 s	2.8%	7.6%	3.2%	13.9%
recommendation service	8726 s	9418 s	9337 s	9729 s	4.2%	11.5%	9.5%	19.5%
adservice	4438 s	4454 s	4518 s	4571 s	1.2%	3.0%	5.3%	8.5%
loadgenerator	200 s	208 s	212 s	215 s	1.4%	7.5%	20.4%	29.4%

Storage Overhead Comparison



SEL-Audit	CamFlow	SPADE	CLARION	Incremental Overhead
168.79 GB	312.56 GB	174.68 GB	181.75 GB	4.05%

Runtime/storage incremental overheads < 5%

Conclusion

- Existing provenance tracking solutions are inadequate for microservice scenarios
  - Namespace unawareness causes fragmentation and ambiguities (soundness).
  - Container unawareness leads to missing essential container semantics (clarity).
- CLARION
  - The first namespace-aware and container-aware provenance tracking solution
  - Comprehensive evaluation shows effectiveness, generality and efficiency of our solution
    - **Effectiveness:** We generate sound and clear provenance graphs of 3 real-world CVEs, which outperform the graphs generated by the traditional solution.
    - **Generality:** We show that our solution is independent of container engines by providing the container initialization graphs and quantitative results for 3 container engines.
    - **Efficiency:** We use a microservice benchmark to test the runtime/storage overhead of CLARION and find that the overhead is <5% over SPADE.