SecQuant: Quantifying Container System Call Exposure

Sunwoo Jang[1], **Somin Song**[1], Byungchul Tak[1], Sahil Suneja[2], Michael V. Le[2], Chuan Yue[3], Dan Williams[4]

[1]Kyungpook National University (KNU), Daegu, Republic of Korea

[2]IBM TJ Watson Research Center, Yorktown Heights, NY, USA

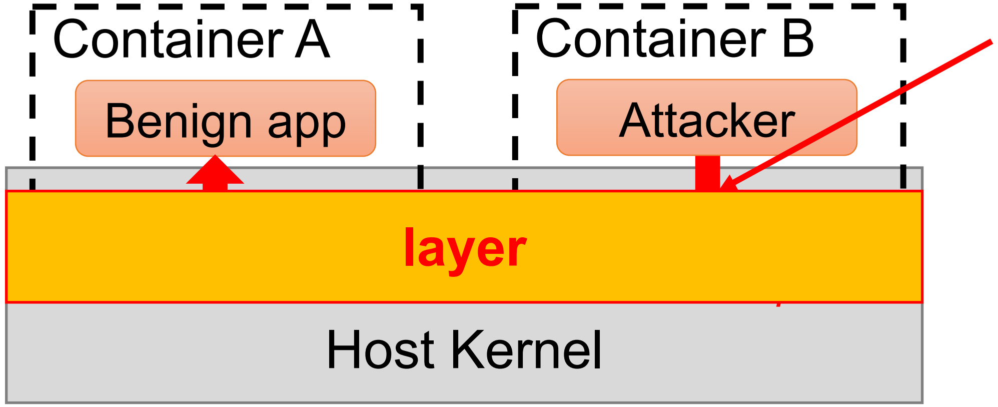
[3]Colorado School of Mines, CO, USA

[4]Virginia Tech, Blacksburg, VA, USA

September 27th, 2022

Security for Container Runtimes

- We focus on the ***container escape (break-out)***
  - Containers = namespaced processes



System calls

- Dirty COW (CVE-2016-5195)
  → `write, madvise`
- Dirty Pipe (CVE-2020-0847)
  → `pipe, splice`
- Dirty Cred (CVE-2022-2588/CVE-2021-4154)
  → `writev`

- Handle system calls for the host kernel
  → Smaller attack surface

- Secure Container Runtimes

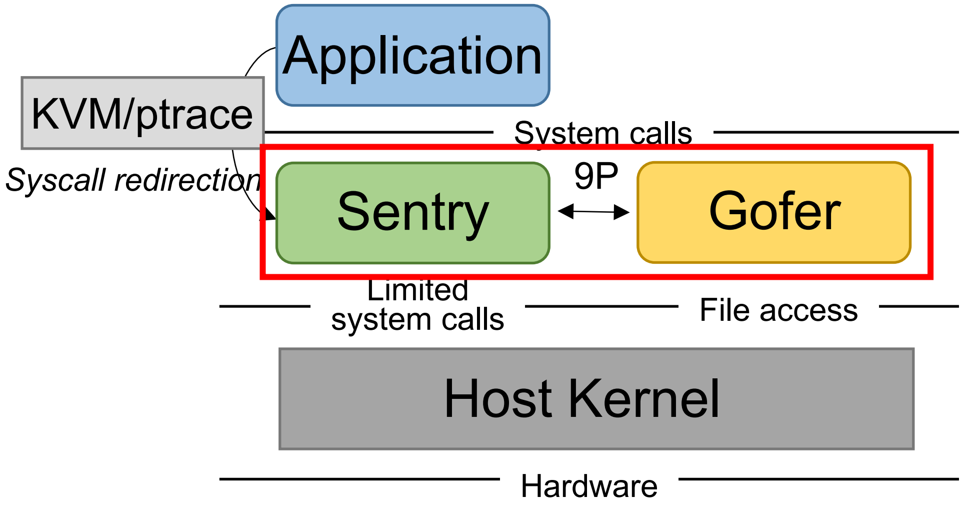
gVisor (Google)

Kata containers

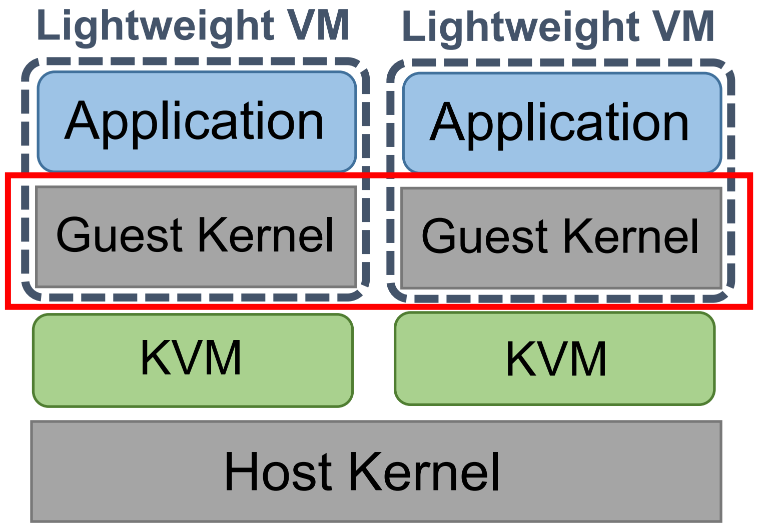
Nabla (IBM)

RunD
[2022 ATC]

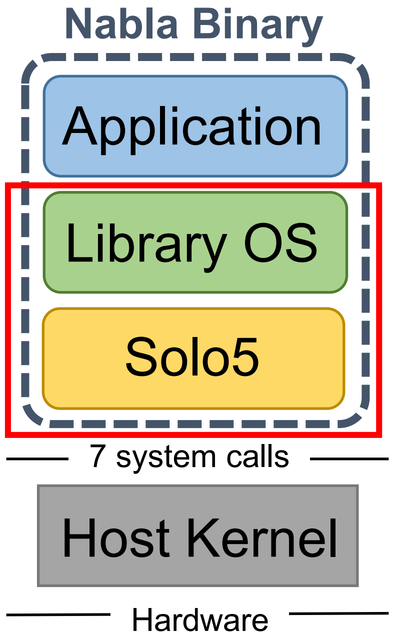
Surrogate (Proxy) Layer: gVisor vs. Kata vs. Nabla



gVisor (Google)



Kata containers

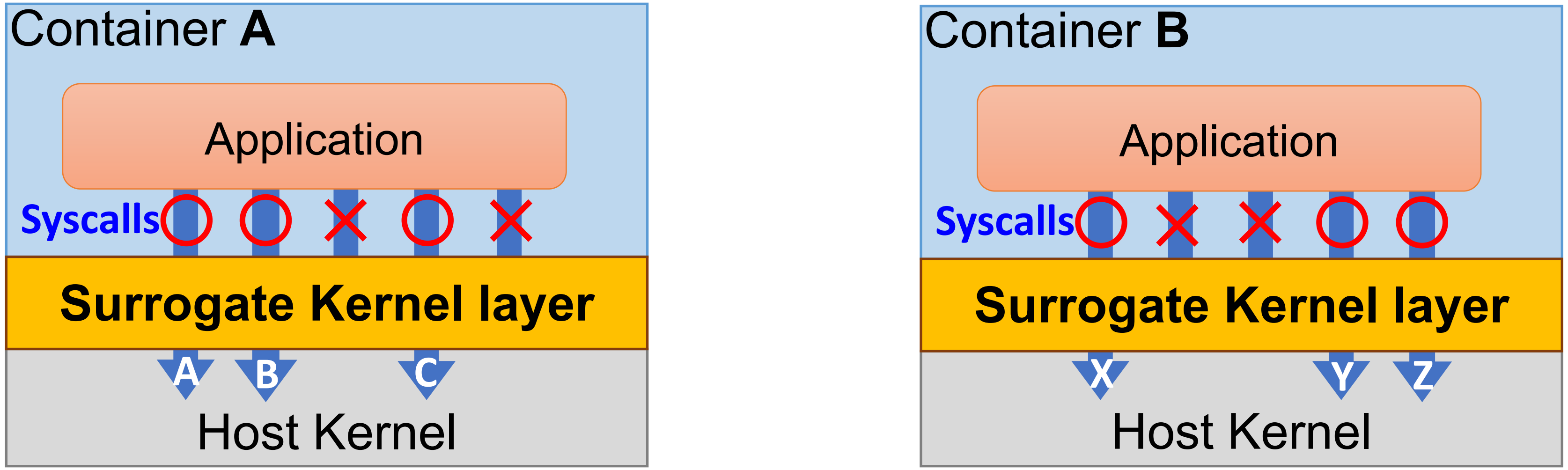


Nabla (IBM)

- **Utility of quantifying container runtime security**
  - Only qualitative statements are given
  - Quantification allows: Comparison, Trend, Engineering, What-if analysis

Our Intuition and Approach

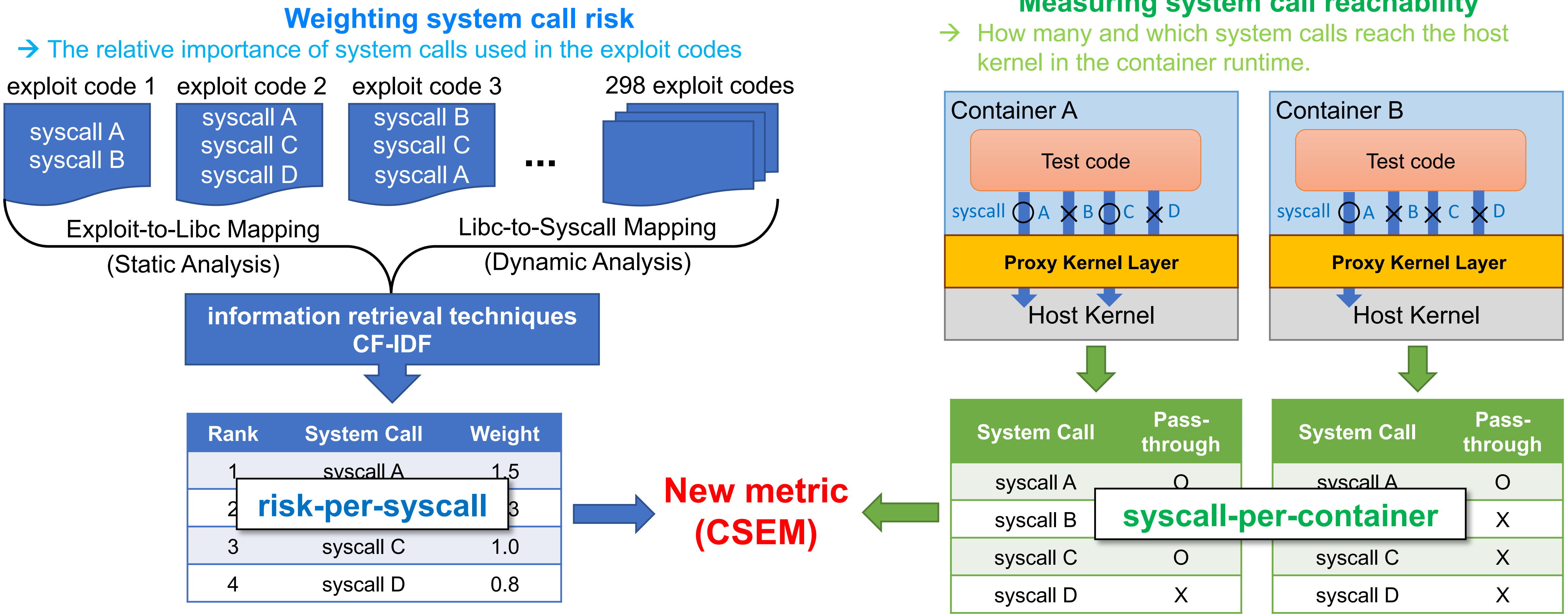
- Attack surface measure: ***system calls***
  - *How many system calls reach the host kernel?*



- Simple counting of system calls (and/or types) is insufficient
- Need to determine the **importance of individual system calls**
  - Exploit codes → which system calls are use in the attacks?

Approach Overview

- An unified metric for comparing container security
  - Two perspectives: **Risk** and **Reachability** of system call



Weighting system call risk

→ The relative importance of system calls used in the exploit codes

Rank	System Call	Weight
1	syscall A	1.5
2	[illegible]	3
3	syscall C	1.0
4	syscall D	0.8

risk-per-syscall

New metric (CSEM)

Measuring system call reachability

→ How many and which system calls reach the host kernel in the container runtime.

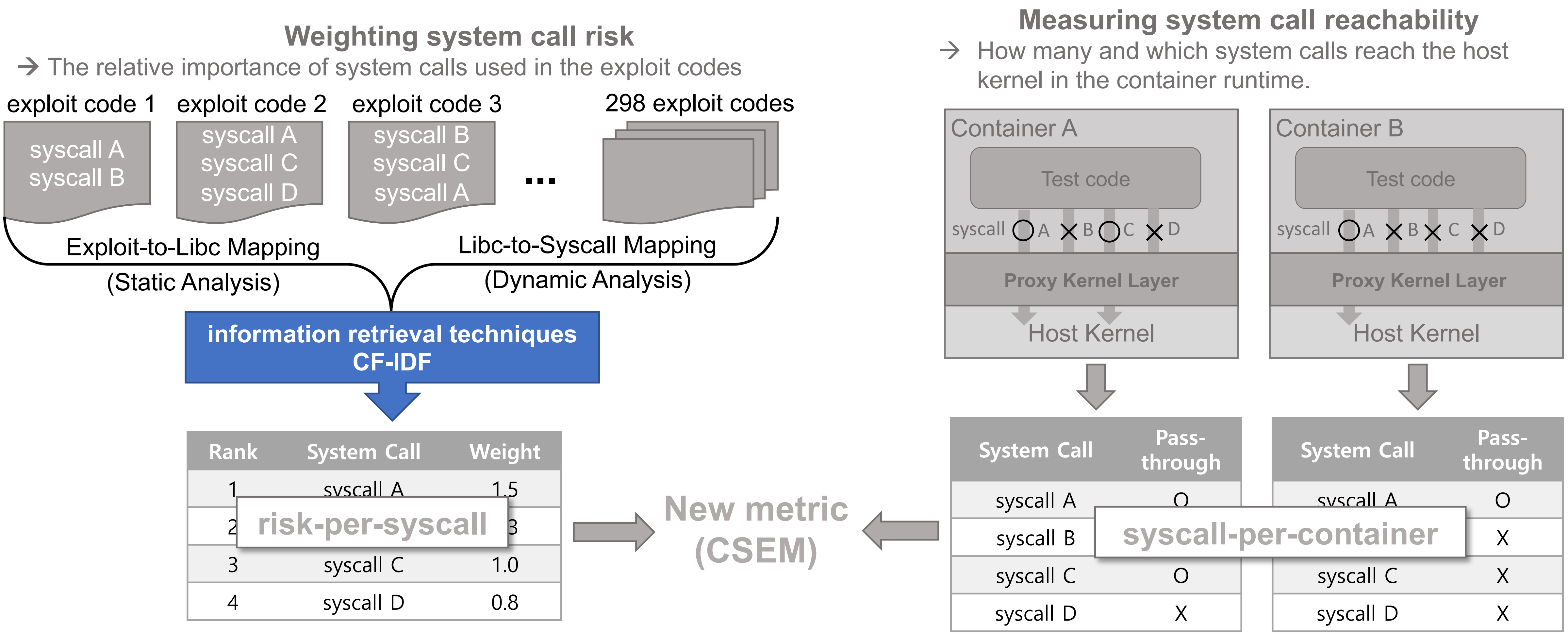
System Call	Pass-through
syscall A	O
syscall B	[illegible]
syscall C	O
syscall D	X

System Call	Pass-through
syscall A	O
syscall B	X
syscall C	X
syscall D	X

syscall-per-container

Approach Overview

- An unified metric for comparing container security
  - Two perspectives: **Risk** and **Reachability** of system call



Weighting system call risk

→ The relative importance of system calls used in the exploit codes

Rank	System Call	Weight
1	syscall A	1.5
2		3
3	syscall C	1.0
4	syscall D	0.8

Measuring system call reachability

→ How many and which system calls reach the host kernel in the container runtime.

System Call	Pass-through
syscall A	O
syscall B	
syscall C	O
syscall D	X

System Call	Pass-through
syscall A	O
	X
syscall C	X
syscall D	X

SCAR: System Call Assessment of Risk

- **System call risk weight assignment**: **CF-IDF**
  - Variation of TF(Term Frequency)-IDF(Inverse Document Frequency) for the security quantification

	Term A	Term B	Term C
Doc 1	0	0	1
Doc 2	2	0	0
Doc 3	1	2	1
DF	2	1	2
IDF	Log(3/**2**)	Log(3/**1**)	Log(3/**2**)

TF: Term frequency within a document

DF: Document frequency that include the Term across documents

SCAR: System Call Assessment of Risk

- **System call risk weight assignment: CF-IDF**
  - Variation of TF(Term Frequency)-IDF(Inverse Document Frequency) for the security quantification

➔ **Term** in **Document** = **System call** in **Exploit code**

	Syscall A	Syscall B	Syscall C
Exploit 1	0	0	1
Exploit 2	2	0	0
Exploit 3	1	2	1
DF	2	1	2
IDF	Log(3/**2**)	Log(3/**1**)	Log(3/**2**)

TF: Frequency in an exploit code may not mean anything.

IDF: Commonly used system calls in exploit codes is less important e.g., `close, brk, exit, nanosleep`

SCAR: System Call Assessment of Risk

- **System call risk weight assignment: CF-IDF**
  - Variation of TF(Term Frequency)-IDF(Inverse Document Frequency) for the security quantification

    ➔ **CF (Class Frequency):** Document Frequency that include the Term within a Class

		Syscall A	Syscall B	Syscall C
Class α	Exploit 1	0	0	1
Class β	Exploit 2	**2**	0	0
	Exploit 3	**1**	2	1
		↓	↓	↓
CF of Class β		2	1	1

Syscall	**Per-syscall risk weight**
Syscall A	0.35
Syscall B	0.48
Syscall C	0.18

CF: The more consistently appearing terms within a class is important to the attack logic of the class

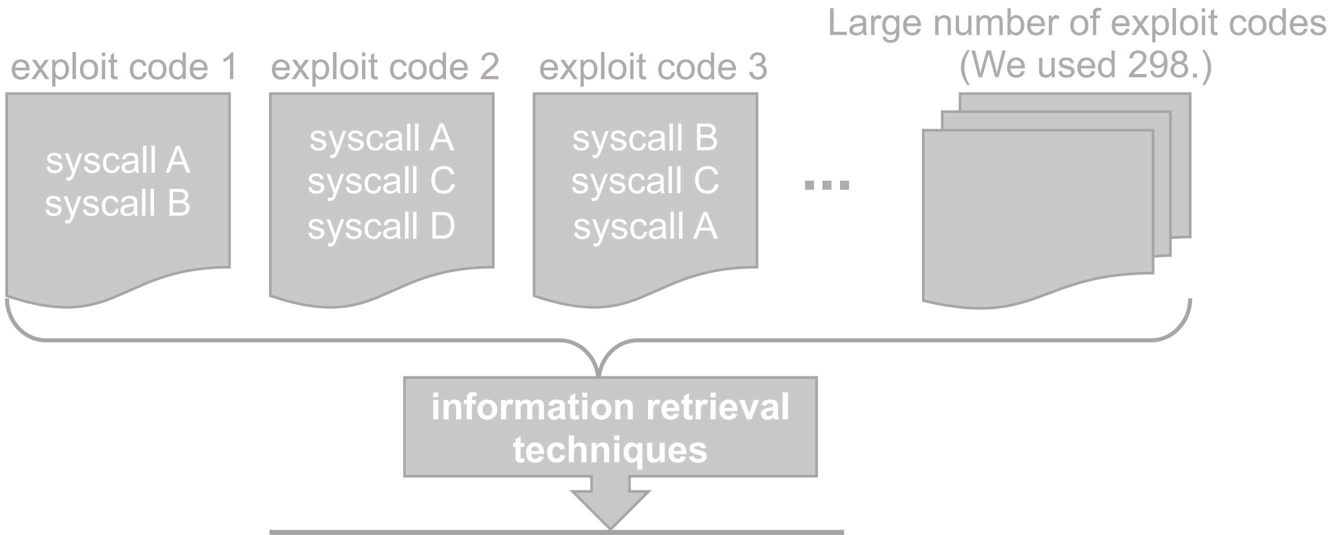
  - Per-syscall risk weight: average CF-IDF values across all exploit codes

Approach Overview

- An unified metric for comparing container security
  - Two perspectives: **Risk** and **Reachability** of system call

Weighting system call risk

→ The relative importance of system calls used in the exploit codes

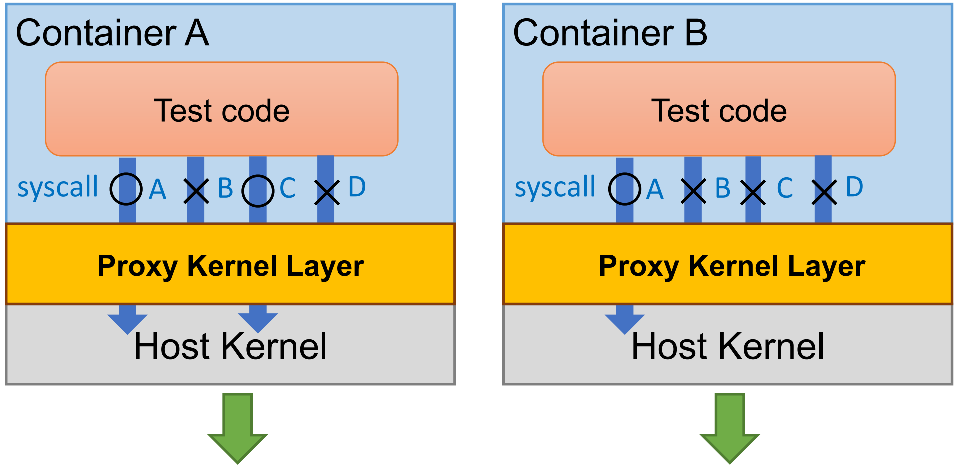


Rank	System Call	Weight
1	syscall A	1.5
2	[illegible]	[illegible]
3	syscall C	1.0
4	syscall D	0.8

risk-per-syscall

Measuring system call reachability

→ How many and which system calls reach the host kernel in the container runtime.



System Call	Pass-through
syscall A	O
syscall B	[illegible]
syscall C	O
syscall D	X

System Call	Pass-through
syscall A	O
[illegible]	X
syscall C	X
syscall D	X

syscall-per-container

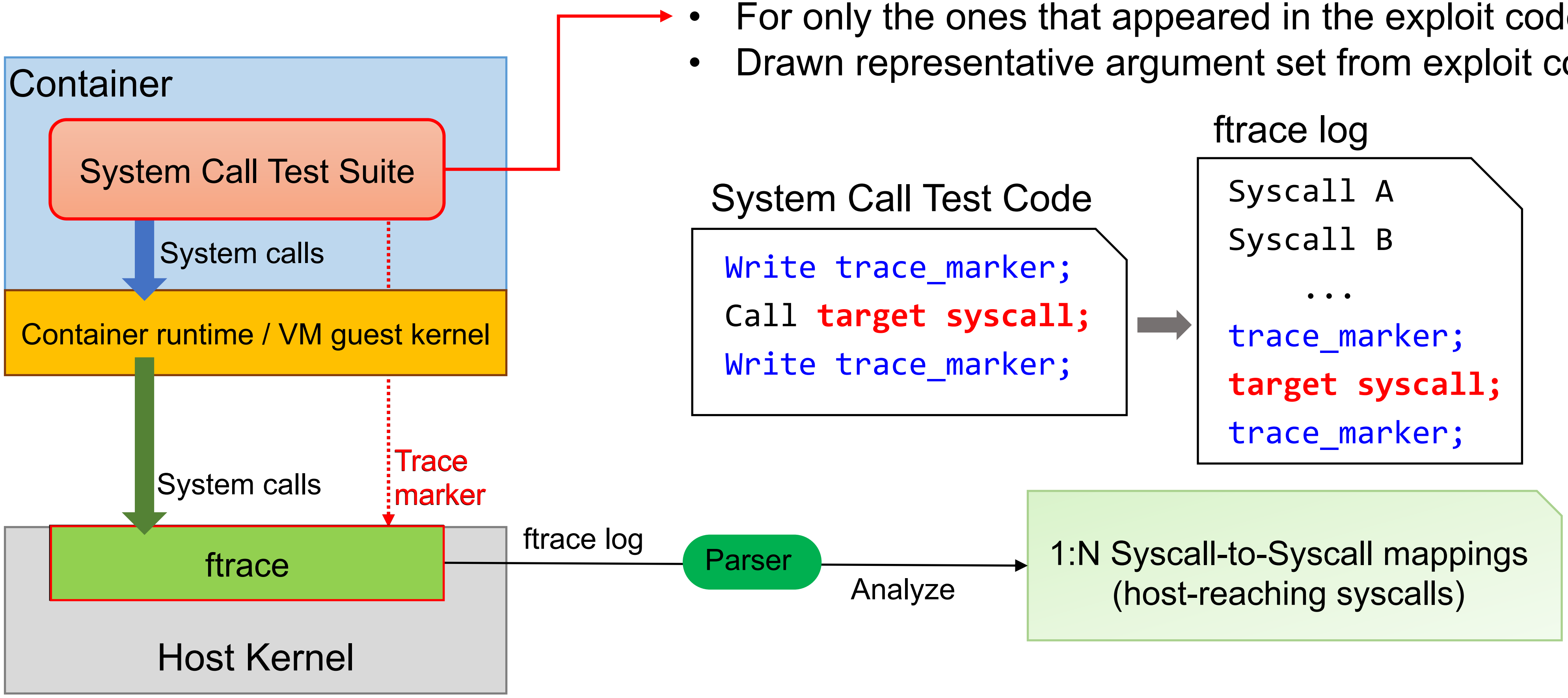
New metric (SecQuant)

System Call Reachability Test Part

SCED (System Call Exposure Discovery)

Custom-built 185 test suite

- For only the ones that appeared in the exploit codes
- Drawn representative argument set from exploit codes



System Call Test Code

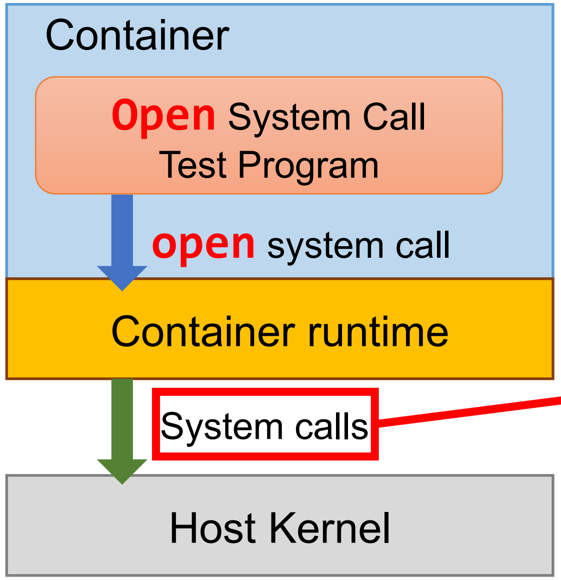
```
Write trace_marker;
Call target syscall;
Write trace_marker;
```

ftrace log

```
Syscall A
Syscall B
    ...
trace_marker;
target syscall;
trace_marker;
```

Example: gVisor vs. Kata Containers

- Set of system calls observed at the host kernel
  - Three kinds: Pass-through, Derived, Blocked



gVisor (runsc-ptrace)	
Host-reaching syscalls	**Count**
futex	20
ptrace	10
openat	4
newfstat	4
wait4	2
fcntl	2
sendmsg	1
recvmsg	1
dup	1
close	1

Kata Containers (kata-qemu)	
Host-reacing syscalls	**Count**
futex	6
write	3
read	3
ppoll	3
openat	3
setresuid	2
setresgid	2
newfstatat	1
geteuid	1
getegid	1

Derived system calls: A syscall is converted into other set of syscalls

Pass-through *(include equivalent syscalls):* A syscall arrives at the host kernel

CSEM: Container Syscall Exposure Measure

- Combining **SCAR**(risk-weights-per-syscall) and **SCED**(syscall-per-container)

per-syscall risk weight vector

$$CSEM(\boxed{W},\boxed{D},\boxed{r}) = \begin{cases} \sum_{s} \boxed{W_s} & \rightarrow \textbf{Pass-through syscalls} \\ \sum_{s} \sum_{d \in D_s} \frac{W_d \boxed{\cdot r}}{\boxed{|D_s|}} & \rightarrow \textbf{Derived syscalls} \end{cases}$$

the whole risk weight values

reduction ratio

observed syscall sets

Average over the set size of D_s

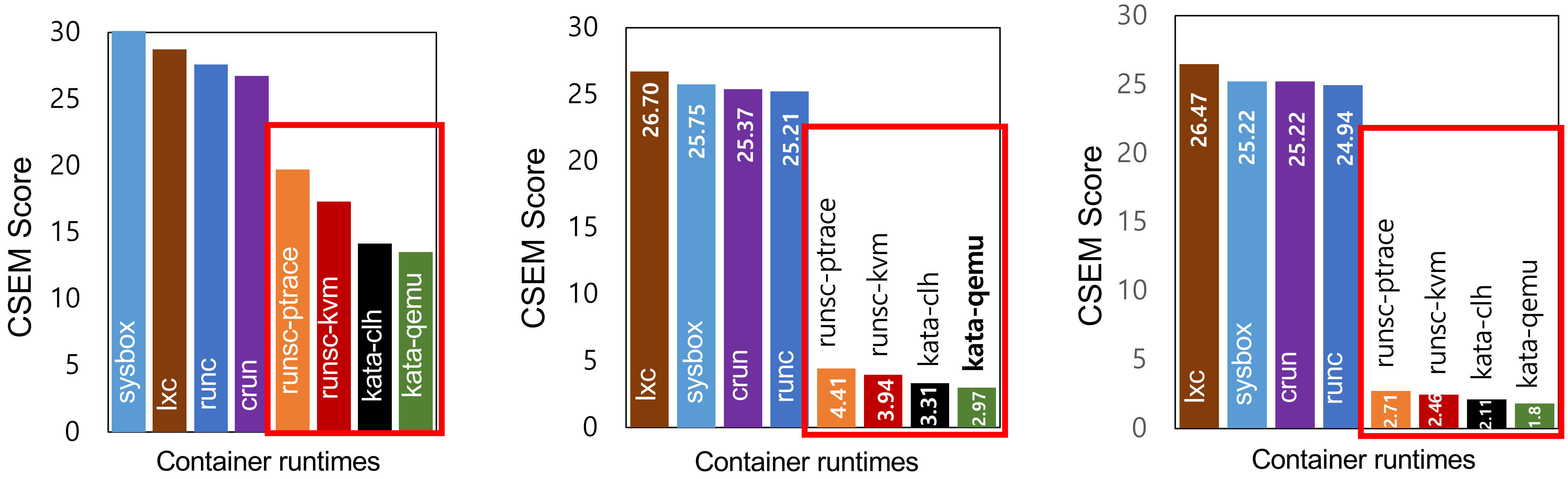
(D_s: a set of derived system calls from a system call s)

reduction ratio ($0 \leq r \leq 1$)
: control the risk-weight contribution of the derived syscalls

Experimental evaluation: Container Runtime Security Analysis

• Container Syscall Exposure Measure Score Comparison

- Baseline Container Runtimes: runc, crun, LXC, and sysbox
- Secure Alternatives: gVisor (runsc-ptrace, runsc-kvm) and Kata containers (kata-qemu, kata-clh)



0% Reduction Ratio

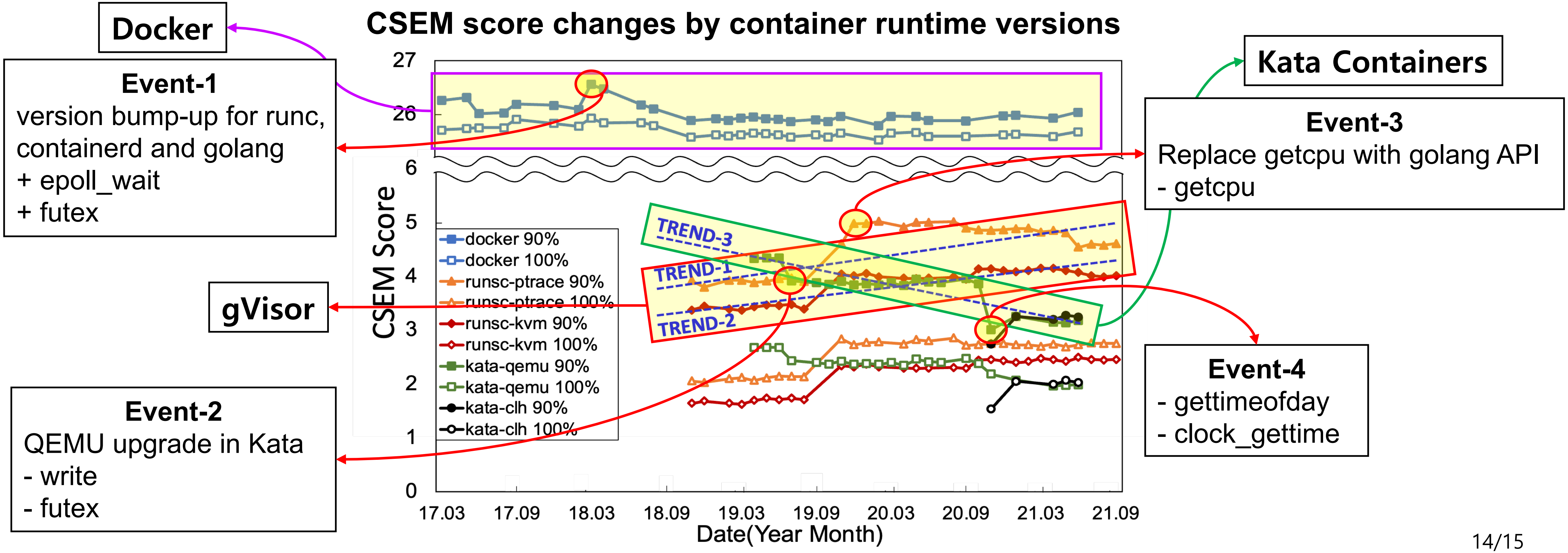
90% Reduction Ratio

100% Reduction Ratio

- Only pass-thru system calls are used in CSEM calculation
- Derived system calls are treated equally as pass-thru system calls
- Secure containers have 4.2~7.5 times less syscall exposure than non-security-oriented ones

Experimental evaluation: Container Runtime Security Analysis

- Historical Trends Across Versions
  - Across a 4.5-year history
  - 31, 35 and 22 versions of Docker (runc), gVisor (runsc-ptrace/kvm), and Kata (kata-qemu/clh)



CSEM score changes by container runtime versions

Docker

Event-1
version bump-up for runc, containerd and golang
+ epoll_wait
+ futex

gVisor

Event-2
QEMU upgrade in Kata
- write
- futex

Kata Containers

Event-3
Replace getcpu with golang API
- getcpu

Event-4
- gettimeofday
- clock_gettime

Conclusion

- First attempt at quantifying the security of secure containers.
  - System call risk weighting + System call reachability testing
- Secure containers still allow a significant number of system calls to reach the host kernel
- Future enhancements
  - Exploit code analysis for broader area
  - Handling of vulnerabilities triggered by non-syscalls such as BufferOverflow, Memory Corruption attacks
  - System call weighting refinement by adding benign code analysis
  - System call argument information in tracing
  - Finding proof of successful attacks using:
    - Pass-through syscalls with modified arguments
    - Derived syscalls

Thank you