CSE 400: Project and Thesis

On Detecting Malicious Code Injection by Monitoring Multi-level Container Activities

Presented By,

1705013 - Md. Olid Hasan Bhuiyan

1705034 - Souvik Das

Supervised By,

Dr. Md. Shohrab Hossain

Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology

Container Issues

Security measure
(Door lock)

A temporary office built
with several containers

Container Issues

Container Issues

Found the weak point of a container!

Advantages

- Portable
- Highly scalable
- Isolated
- Individual security mechanism
- Multi-tenant service

Disadvantages

- Large attack surface
- Communication complexity
- Thin protection layer

What is a container?

What is a container?

“A lightweight OS-level virtualization method”

“Stand-alone piece of executable software”

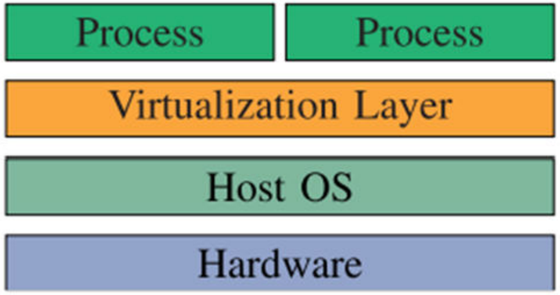
“NOT a virtual machine”

“Process with isolation, shared resources and layered filesystem”

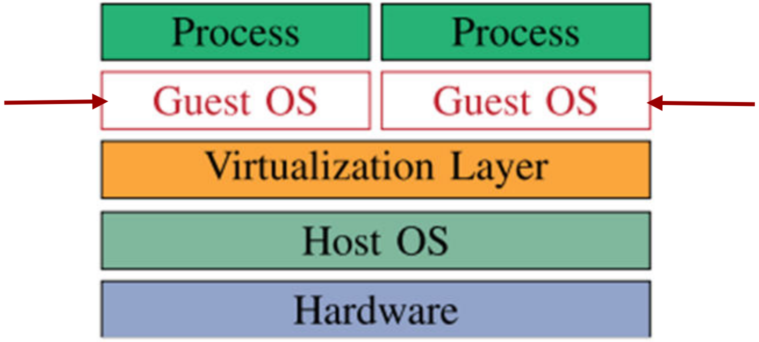
Container vs VM

- Shared OS
- Less secured
- More suitable for microservice

- Separate OS for each process
- More secured
- Less suitable for microservice



Container



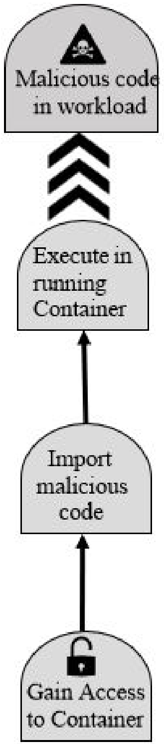
VM

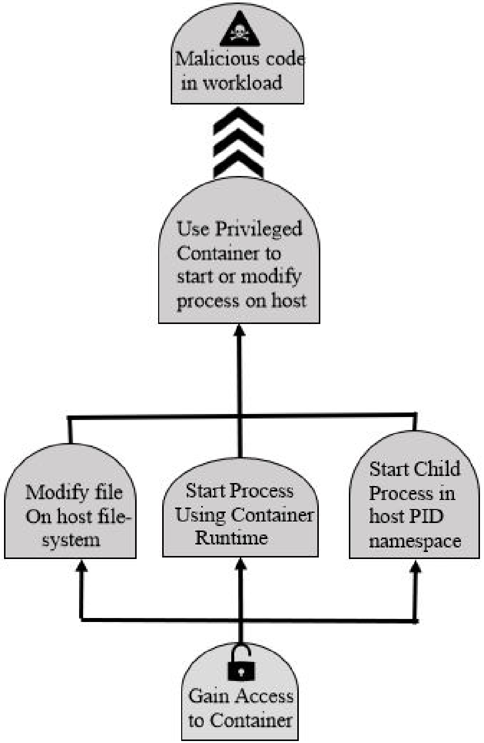
Problem Definition

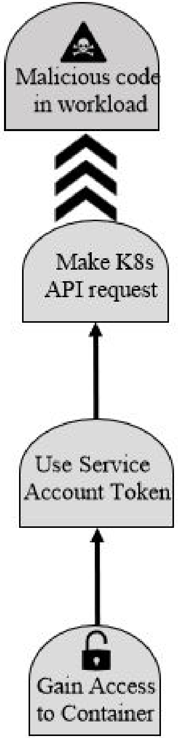
- **Malicious code injection**
  - A type of cyber attack where an attacker inserts malicious code into a vulnerable application or system, with the intent of compromising the security or functionality of the target.

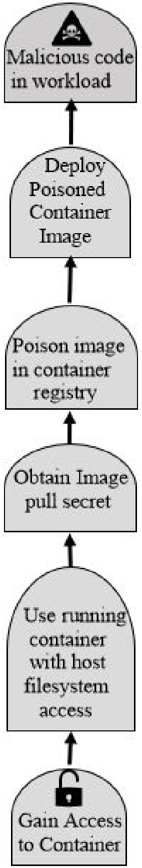
- There are several potential **attack paths** to insert malicious code into a container.

Problem Definition









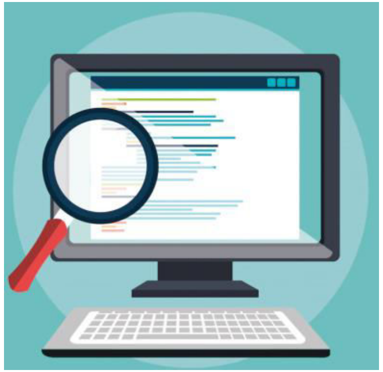
Examples: XSS attack SQLi attack RBAC attack CSRF attack

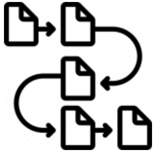
Our Goal

Our aim: Detecting malicious code injection in container for all **potential attack paths**

Achieved by:

1. Monitoring different levels of container
2. Fetching different features(**name, sequence, frequency)** of system calls
3. Using different tools like **sysdig, strace and kubernetes** dashboard





Motivation

- Increasing usage of container technology and its weak security system
- Malicious code injection is initiator to many other attacks
- Malicious code injection is difficult to detect and fix due to complexity in container layering

Related Works

Limitations

- Proposed a common solution - mostly using sequence of system calls
- Did not cover all potential attack paths of malicious code injection

Related Works

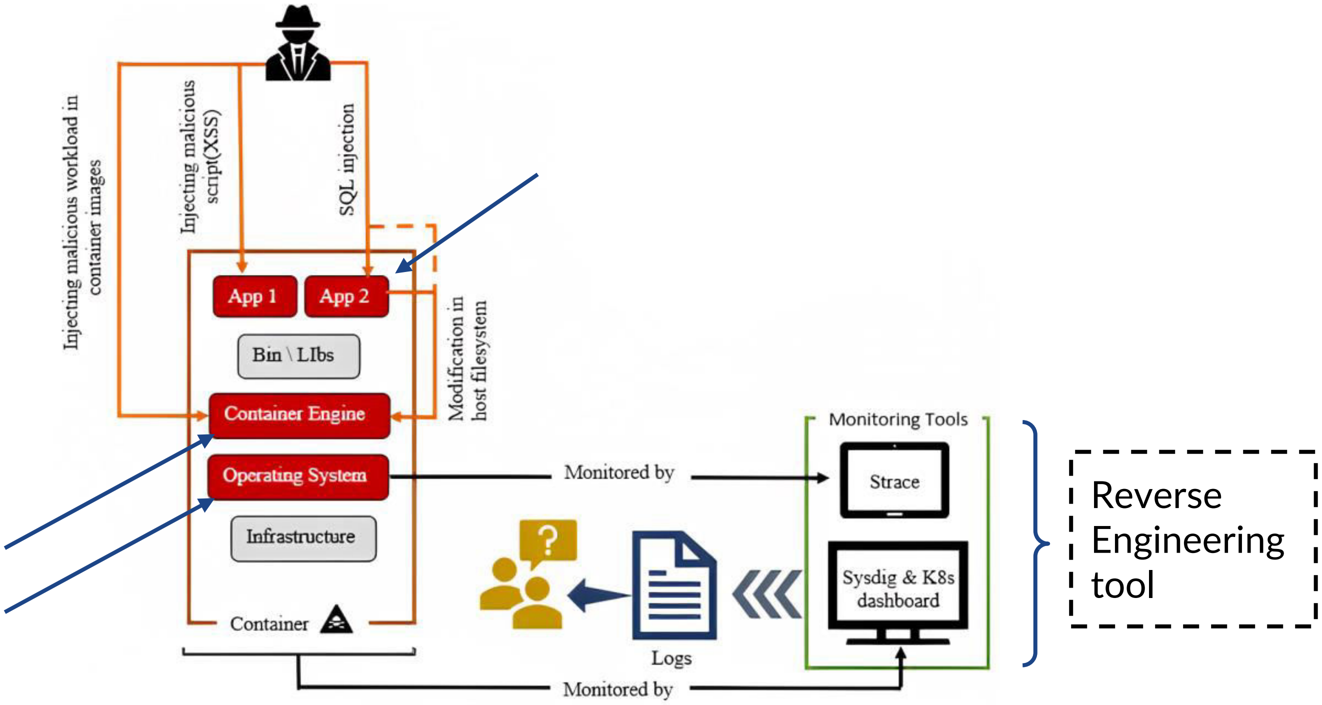
Limitation

- Did not implement their suggested methodology and evaluated it using any kind of performance metrics, let alone work on a specific attack signature

Gap Analysis

- Using a general solution or detection mechanism for different types of attacks
- Absence of attack specific detection mechanism
- Some of the research works only proposed a methodology, did not even test their suggested method
- None of these research works are related to multi-level protection in containerized environment

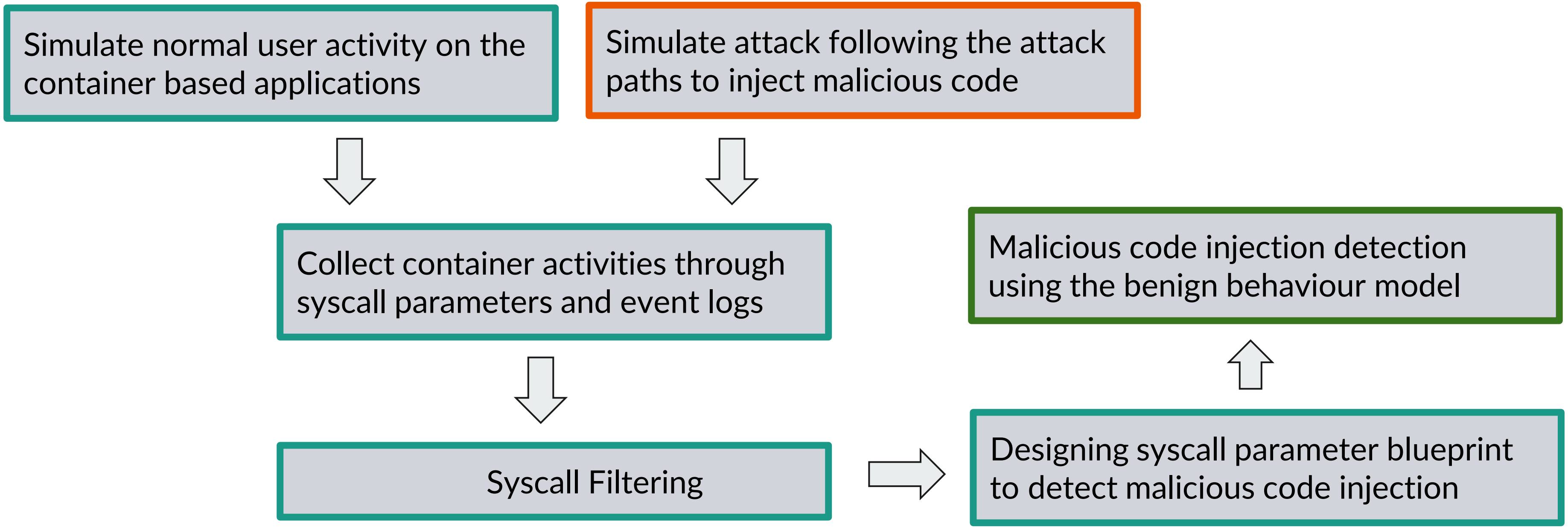
Proposed Approach



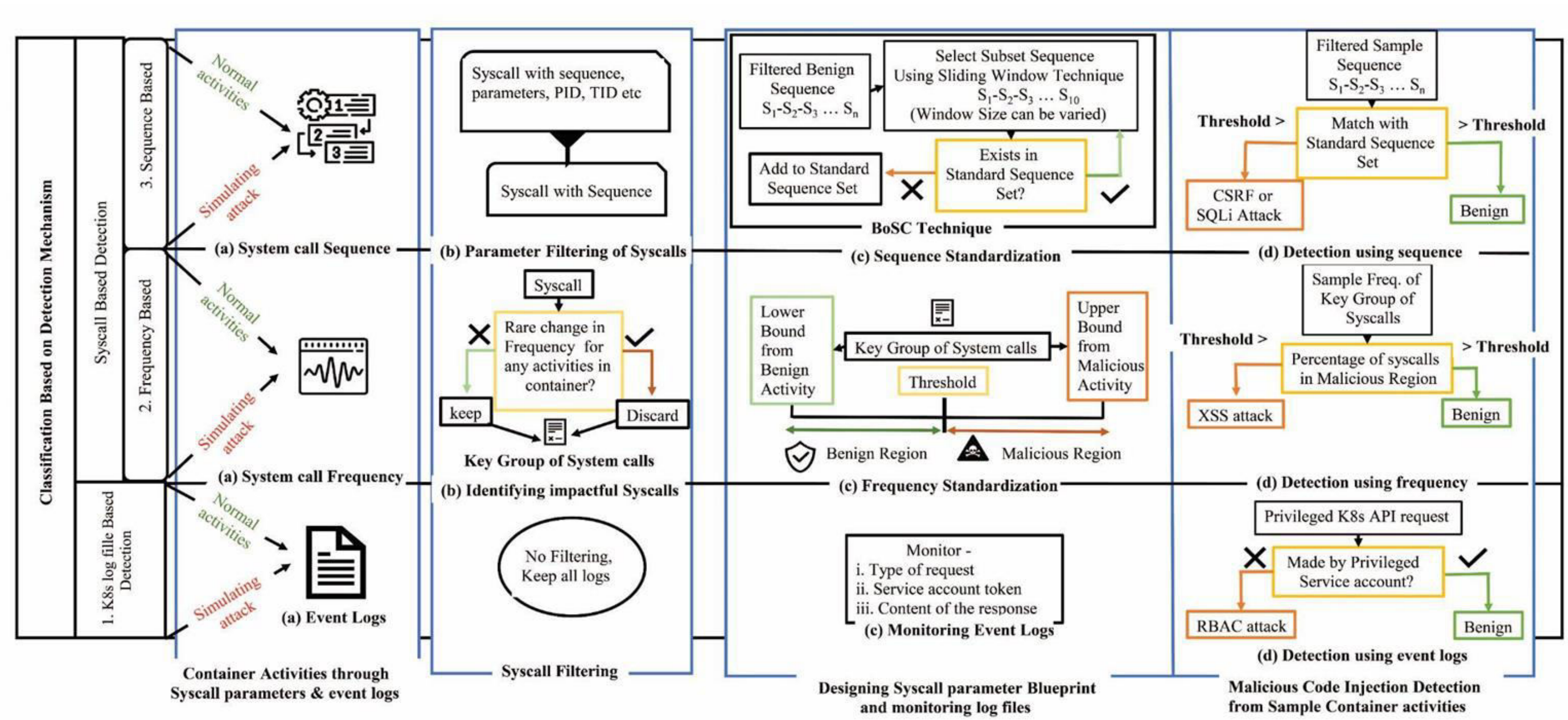
Our proposed container monitoring system

Methodology

Steps



Methodology Overview



Methodology

Simulate normal user activity on the container based applications

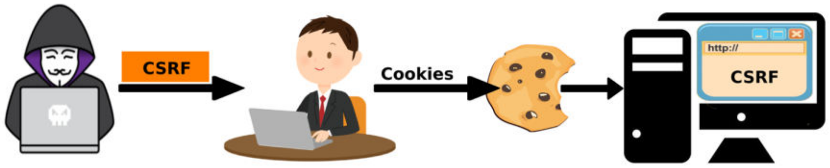
1. Have simulated 4 different container based applications(based on seed-ubuntu)
    a. To implement 4 different attacks(XSS, CSRF, SQLi and RBAC attack)
    b. To generate data for benign activities

2. Tools and platform we used for our simulation are:

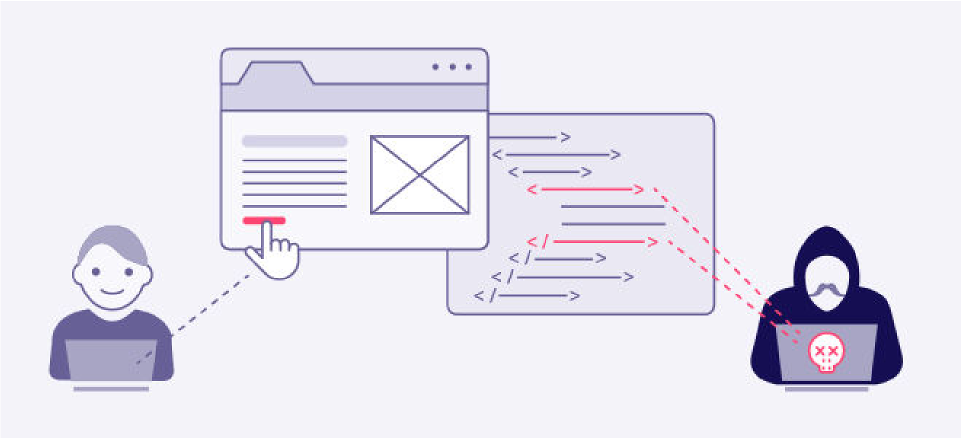
- **Operating system**: Seed Ubuntu 20.04
- **Local kubernetes orchestrator**: Minikube v1.28.0
- **Reverse engineering tools**: Strace, Sysdig, Minikube dashboard

Methodology

Simulate attack following the attack paths to inject malicious code



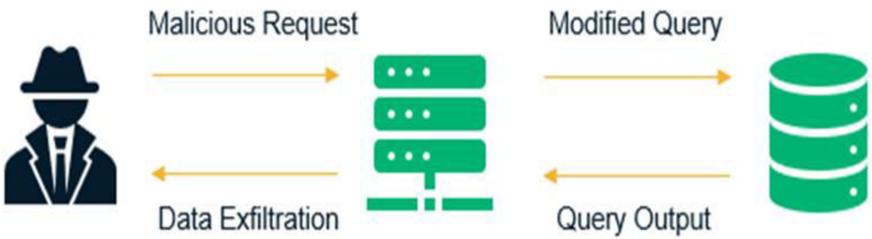
CSRF attack



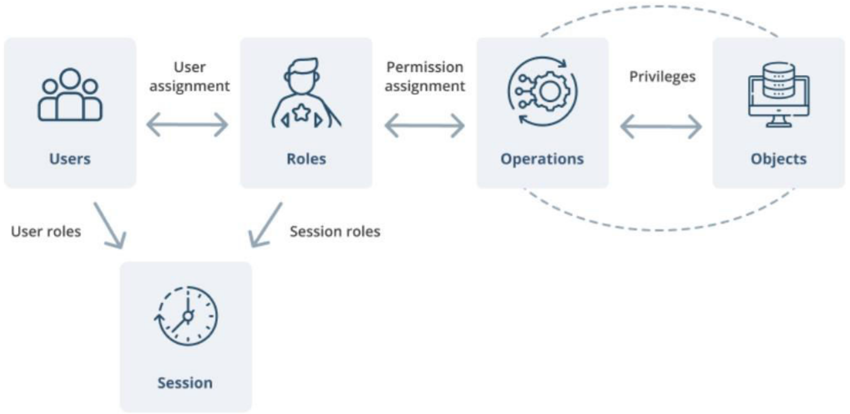
XSS attack

Methodology

Simulate attack following the attack paths to inject malicious code



SQLi attack

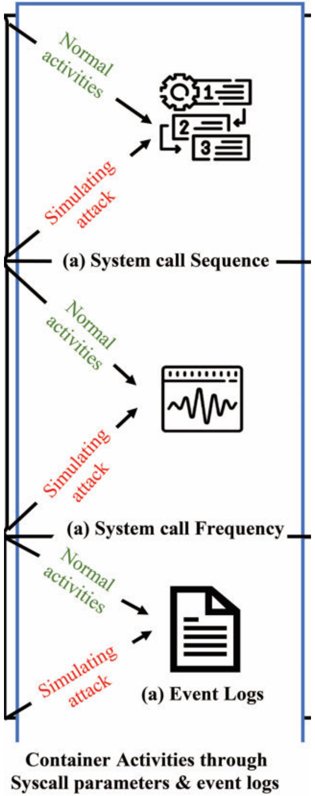


RBAC attack

Methodology

Collect container activities through syscall parameters and event logs

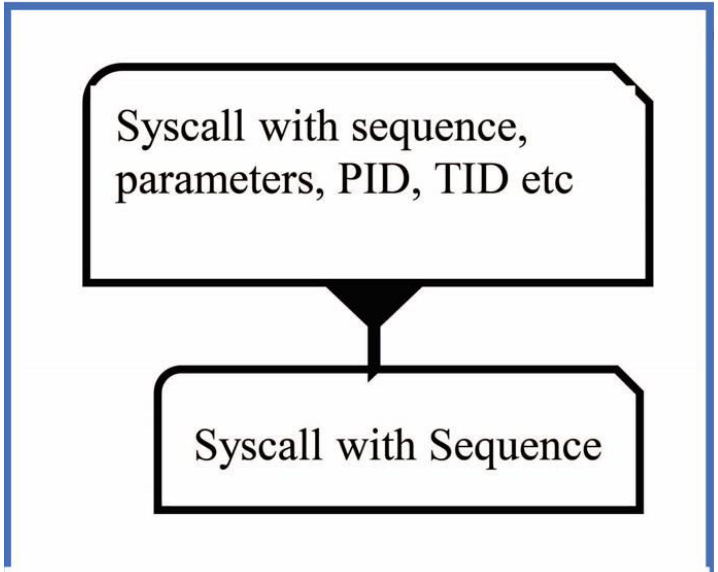
Collect container activities expressed through different features of system calls like frequency, name and sequence



Gather event log files using minikube dashboard to track user activity

Methodology

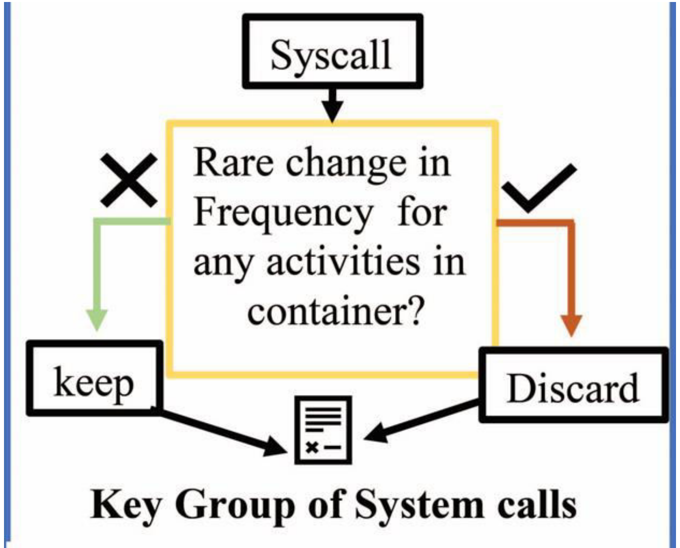
Syscall Filtering → **Using Sequence**



(b) Parameter Filtering of Syscalls

Methodology

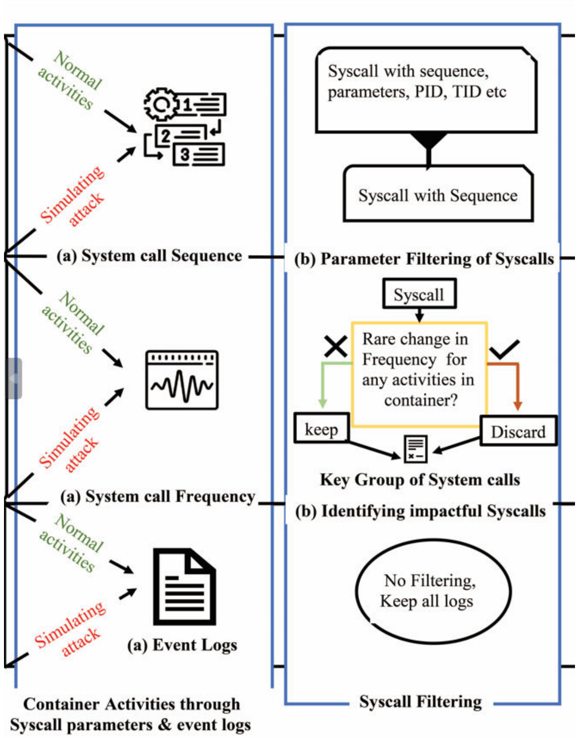
Syscall Filtering → **Using Frequency**



(b) Identifying impactful Syscalls

Methodology

Syscall Filtering



Methodology

Designing syscall parameter blueprint to detect malicious code injection

Monitoring Event logs

Monitor -

i. Type of request

ii. Service account token

iii. Content of the response

(c) Monitoring Event Logs

Methodology

Designing syscall parameter blueprint to detect malicious code injection

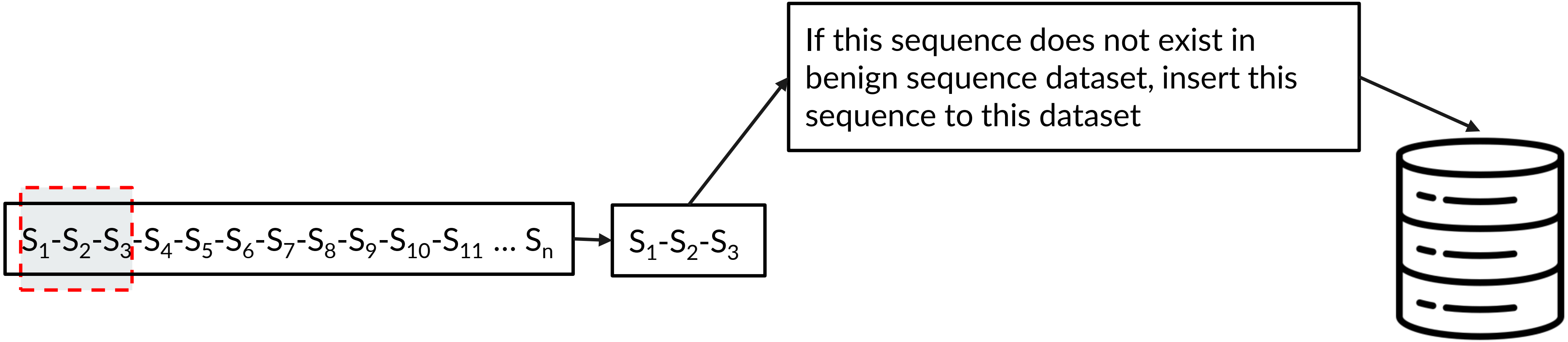
BoSC technique

S_1-S_2-S_3-S_4-S_5-S_6-S_7-S_8-S_9-S_{10}-S_{11} … S_n

Methodology

Designing syscall parameter blueprint to detect malicious code injection

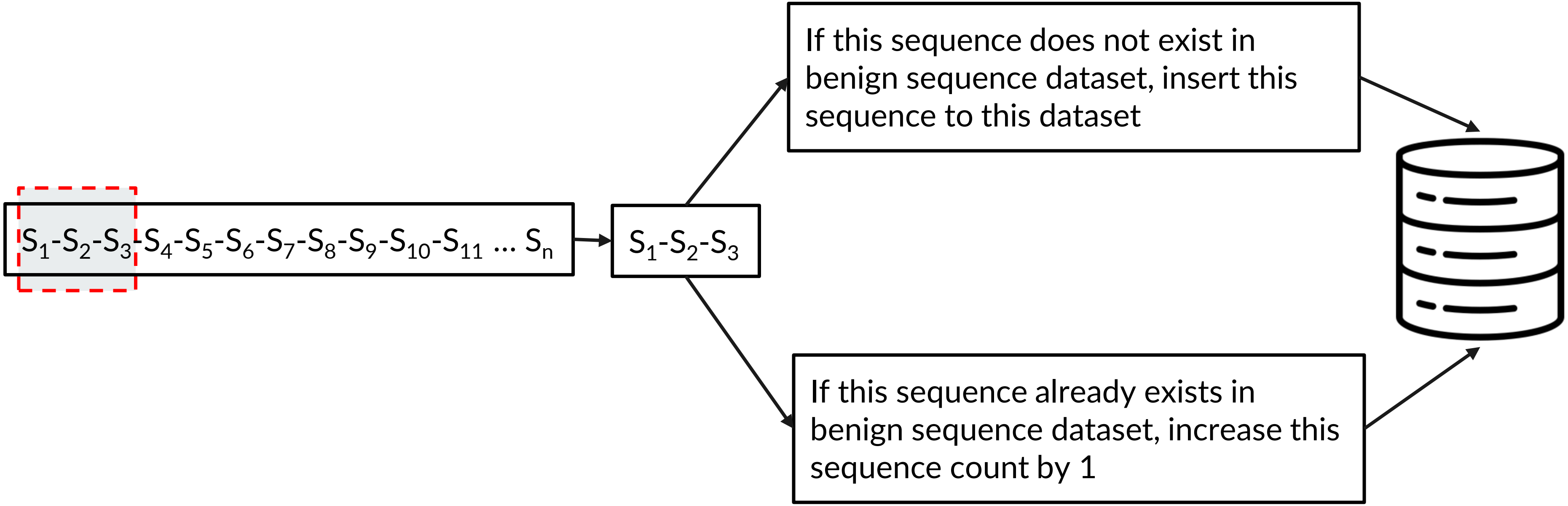
BoSC technique



Methodology

Designing syscall parameter blueprint to detect malicious code injection

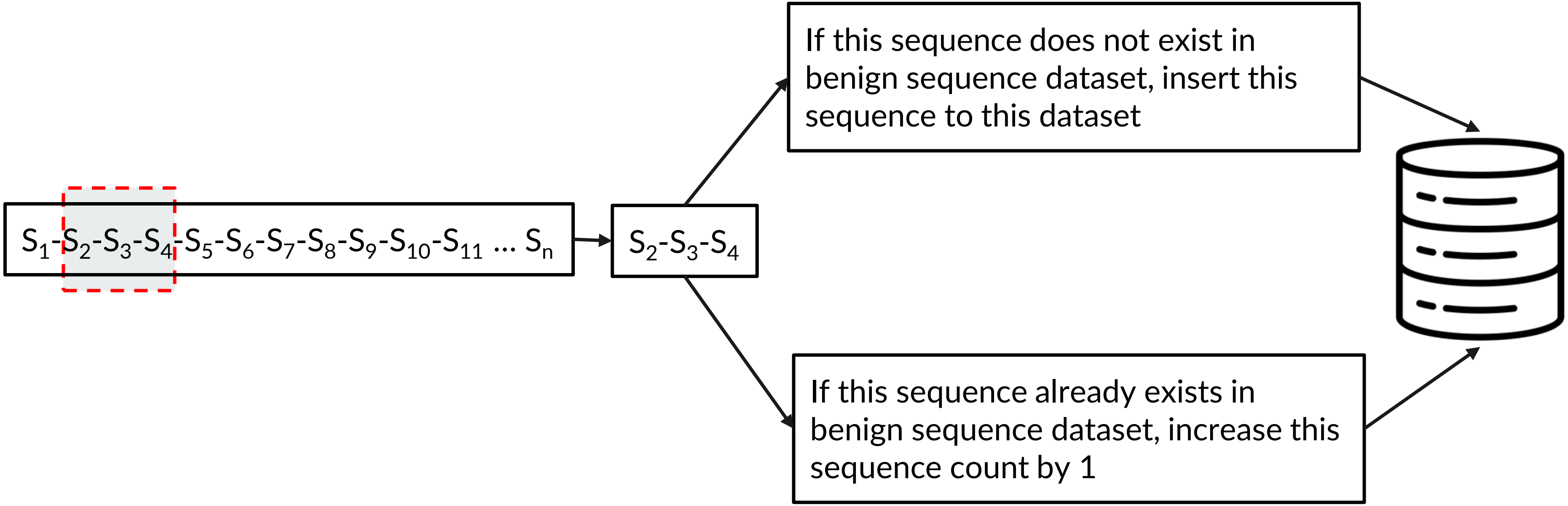
BoSC technique



Methodology

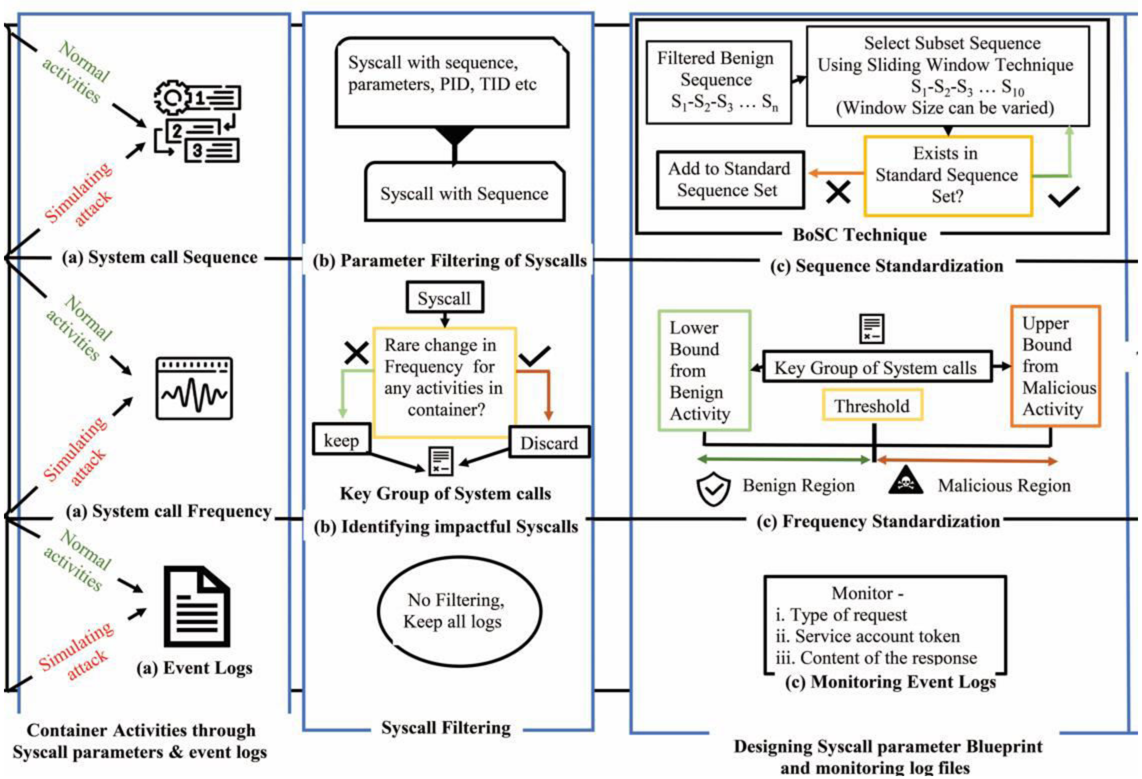
Designing syscall parameter blueprint to detect malicious code injection

BoSC technique



Methodology

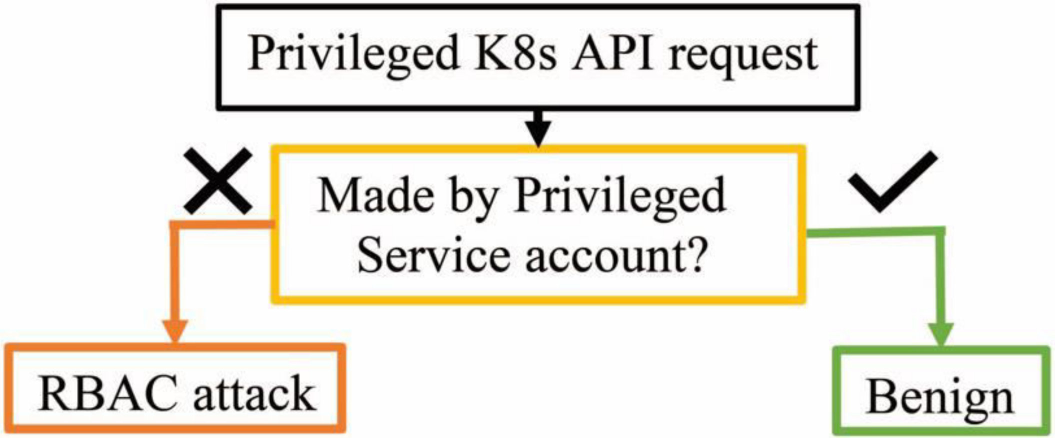
Designing syscall parameter blueprint to detect malicious code injection



Methodology

Malicious code injection detection using the benign behaviour model

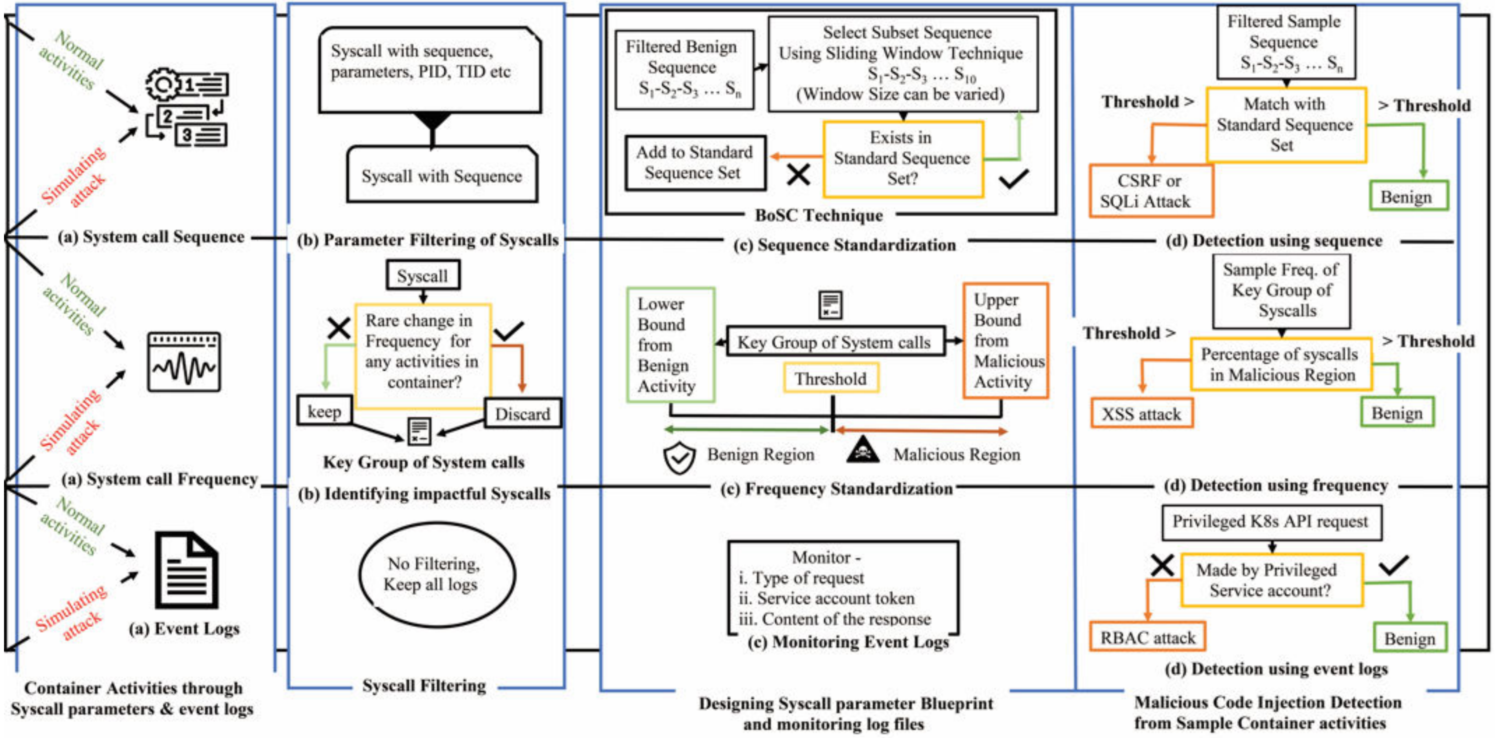
Detection using event logs



(d) Detection using event logs

Methodology

Malicious code injection detection using the benign behaviour model



Methodology

Classification of our detection mechanism

We can classify our detection mechanism in two main classes:

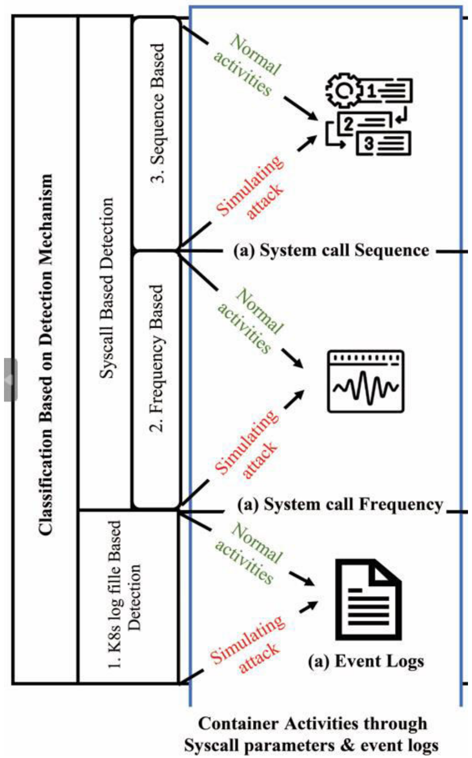
- K8s log file based detection
- System call based detection

System call based detection can be divided into two more classes:

- Sequence based detection
- Frequency based detection

Methodology

Classification of our detection mechanism



Performance Evaluation

We are able to achieve decent score compared to the solution provided by **Cavalcanti et al. [3]**. The following performance metrics are used to evaluate our detection mechanism-

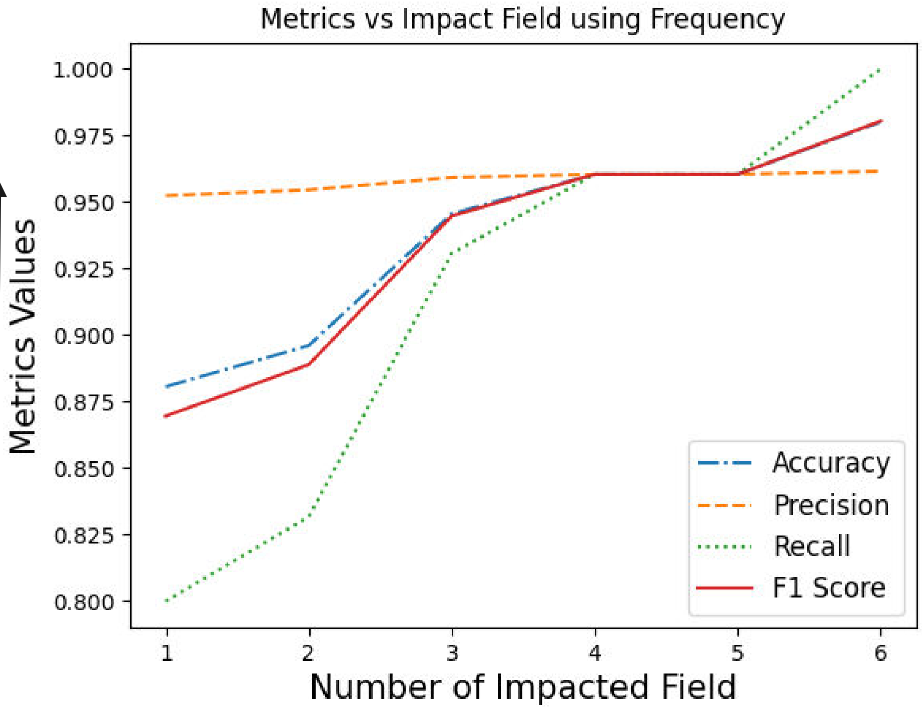
Metrics	Ours	Cavalcanti et al.
Precision	91.21%	79.5%
Recall	94.73%	85.5%
Accuracy	92.06%	-
F1-score	93.81%	83.2%

Performance Evaluation

Performance metrics using Frequency

# of Impacted field(i.e,attack severity)

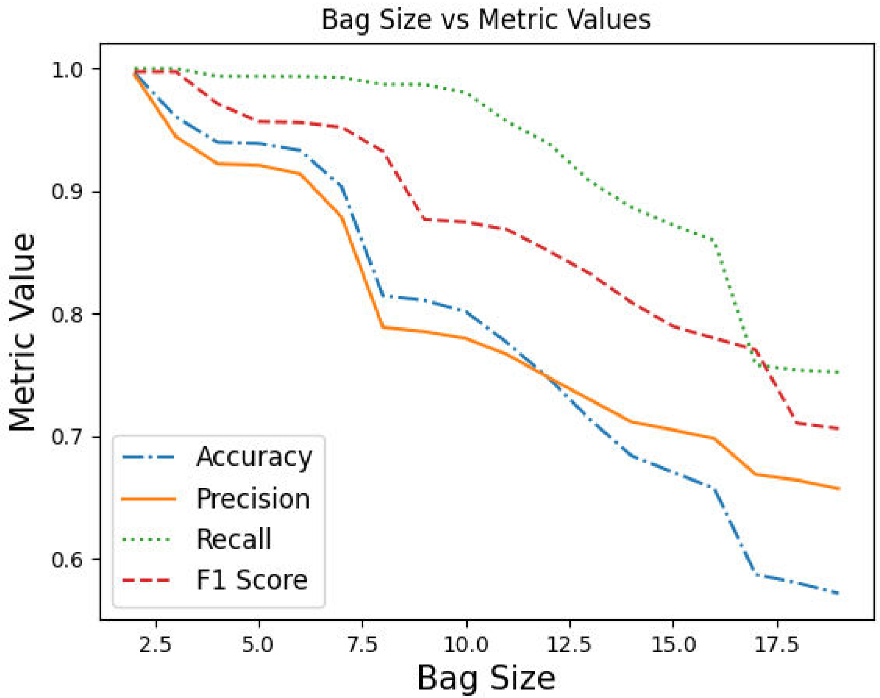
Performance



Performance Evaluation

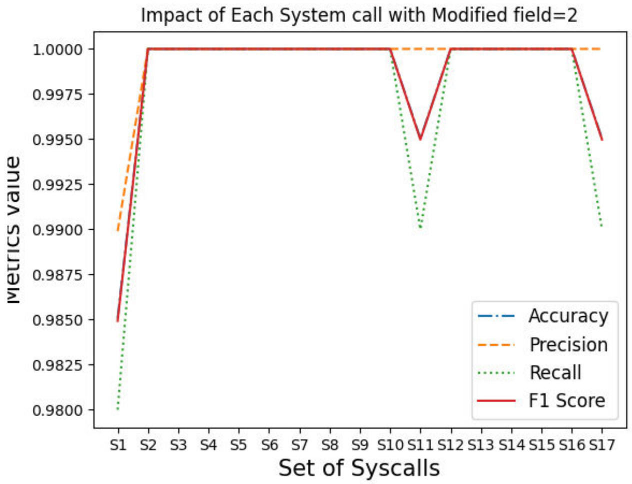
Performance metrics using Sequence

Performance metrics show decent scores for a bag size of **2 to 6**



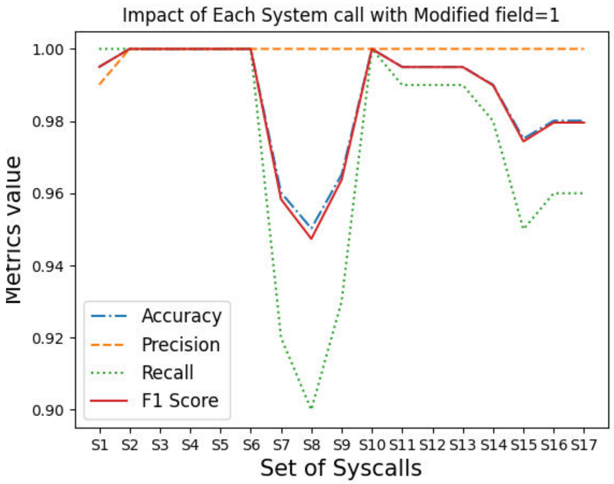
Performance Evaluation

Identifying key group of system calls



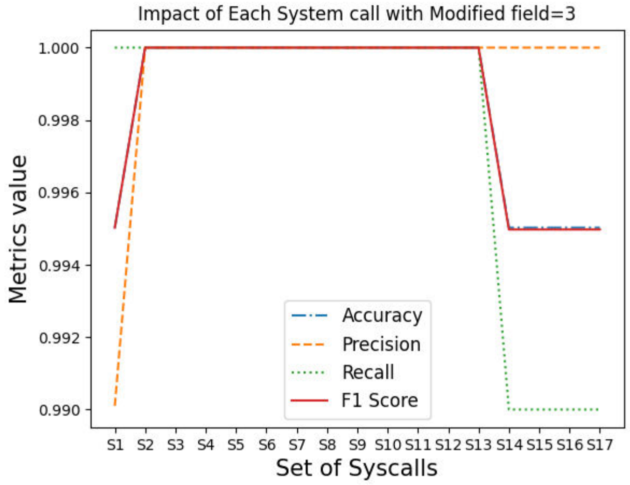
Values along X-axis		
Set	Modified Field=1	Modified Field=2
S1	write	pwrite
S2	S1+fdatasync	S1+write
S3	S2+pwrite	S2+fdatasync
S4	S3+pread	S3+fsync
S5	S4+fsync	S4+io_submit
S6	S5+io_submit	S5+sched_yield
S7	S6+sched_yield	S6+futex
S8	S7+nanosleep	S7+nanosleep
S9	S8+futex	S8+unknown
S10	S9+munmap	S9+mmap
S11	S10+unknown	S10+munmap
S12	S11+mmap	S11+sendto
S13	S12+mprotect	S12+recvfrom
S14	S13+sysdigevent	S13+io_getevents
S15	S14+io_getevents	S14+ppoll
S16	S15+sendto	S15+getrusage
S17	S16+recvfrom	S16+sched_getaffinity

TABLE I: Definition of sets of system calls



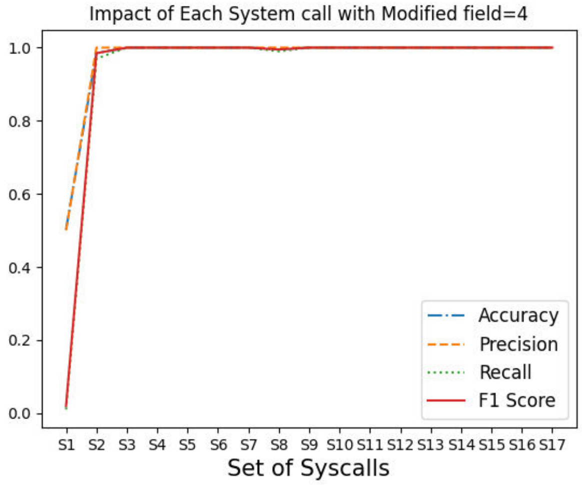
Performance Evaluation

Identifying key group of system calls



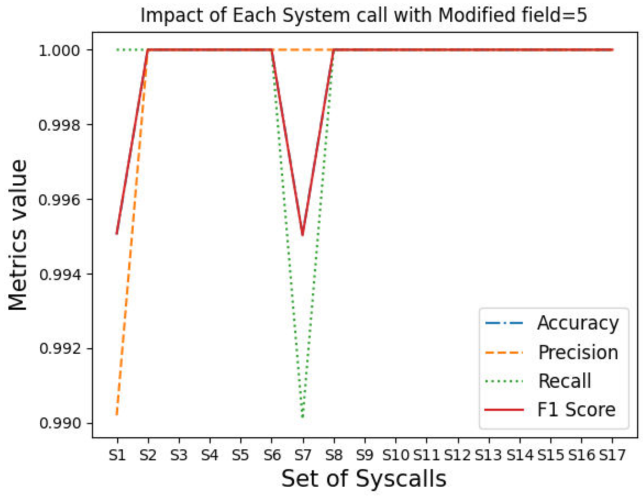
Values along X-axis		
Set	Modified Field=3	Modified Field=4
S1	fdatasync	pread
S2	S1+write	S1+fdatasync
S3	S2+pwrite	S2+write
S4	S3+fsync	S3+pwrite
S5	S4+io_submit	S4+fsync
S6	S5+sched_yield	S5+io_submit
S7	S6+futex	S6+sched_yield
S8	S7+nanosleep	S7+nanosleep
S9	S8+munmap	S8+futex
S10	S9+unknown	S9+unknown
S11	S10+mmap	S10+mmap
S12	[illegible]	[illegible]
S13	S12+recvfrom	S12+ppoll
S14	S13+io_getevents	S13+recvfrom
S15	S14+ppoll	S14+sendto
S16	S15+getrusage	S15+io_getevents
S17	S16+sysdigevent	S16+mprotect

TABLE II: Definition of sets of system calls



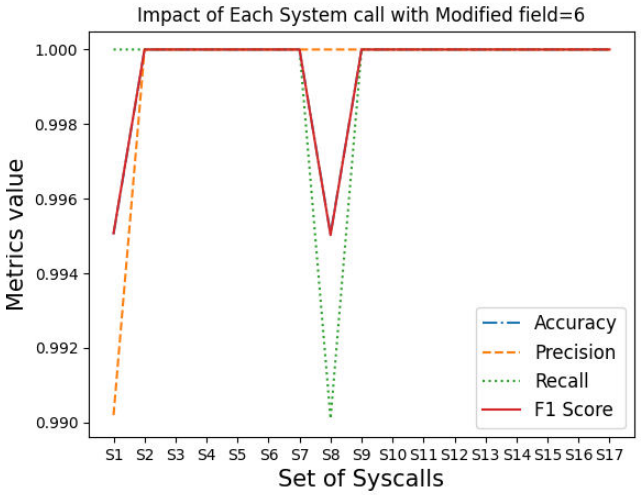
Performance Evaluation

Identifying key group of system calls



Values along X-axis		
Set	Modified Field=5	Modified Field=6
S1	write	fdatasync
S2	S1+fdatasync	S1+write
S3	S2+pwrite	S2+pwrite
S4	S3+fsync	S3+do_submit
S5	S4+io_submit	S4+fsync
S6	S5+sched_yield	S5+io_submit
S7	S6+nanosleep	S6+sched_yield
S8	S7+futex	S7+nanosleep
S9	S8+unknown	S8+futex
S10	S9+mmap	S9+ppoll
S11	S10+munmap	S10+recvfrom
S12	S11+ppoll	S11+munmap
S13	S12+recvfrom	S12+unknown
S14	S13+sendto	S13+sendto
S15	S14+io_getevents	S14+mmap
S16	S15+getrusage	S15+io_getevents
S17	S16+sched_getaffinity	S16+pread

TABLE III: Definition of sets of system calls



Our Contribution

- ❖ Multi-level monitoring of container using different monitoring tools
  - ➢ Makes it feasible to identify an attack in a different level if an attacker manages to get past a single layer of a container without being detected

- ❖ Path Specific detection mechanism
  - ➢ Covers lots of potential ways to inject malicious code in a container

- ❖ Identifying Key group of system calls
  - ➢ Have increased the efficiency of our detection mechanism as we are no longer concerned about unnecessary system calls

References

1. A. S. Abed, T. C. Clancy, and D. S. Levy, “Applying bag of system calls for anomalous behavior detection of applications in linux containers,” in 2015 IEEE globecom workshops (GC Wkshps). IEEE, 2015, pp. 1–5
2. G. R. Castanhel, T. Heinrich, F. Ceschin, and C. Maziero, “Taking a peek: An evaluation of anomaly detection using system calls for containers,” in 2021 IEEE Symposium on Computers and Communications (ISCC). IEEE, 2021, pp. 1–6
3. M. Cavalcanti, P. Inacio, and M. Freire, “Performance evaluation of container-level anomaly-based intrusion detection systems for multi-tenant applications using machine learning algorithms,” in Proceedings of the 16th International Conference on Availability, Reliability and Security, 2021, pp. 1–9
4. M. Souppaya, J. Morello, and K. Scarfone, “Application container security guide,” National Institute of Standards and Technology, Tech. Rep., 2017
5. X. Lin, L. Lei, Y. Wang, J. Jing, K. Sun, and Q. Zhou, “A measurement study on linux container security: Attacks and countermeasures,” in Proceedings of the 34th Annual Computer Security Applications Conference, 2018, pp. 418–429
6. J. Chelladhurai, P. R. Chelliah, and S. A. Kumar, “Securing docker containers from denial of service (dos) attacks,” in 2016 IEEE International Conference on Services Computing (SCC). IEEE, 2016, pp. 856–859

References

7. O. Tunde-Onadele, J. He, T. Dai, and X. Gu, “A study on container vulnerability exploit detection,” in 2019 ieee international conference on cloud engineering (IC2E). IEEE, 2019, pp. 121– 127.
8. T. Bui, “Analysis of docker security,” arXiv preprint arXiv:1501.02967, 2015.
9. T. Yarygina and C. Otterstad, “A game of microservices: Automated intrusion response,” in Distributed Applications and Interoperable Systems: 18th IFIP WG 6.1 International Conference, DAIS 2018, Held as Part of the 13th International Federated Conference on Distributed Computing Techniques, DisCoTec 2018, Madrid, Spain, June 18-21, 2018, Proceedings 18. Springer, 2018, pp. 169–177.
10. V. V. Sarkale, P. Rad, and W. Lee, “Secure cloud container: Runtime behavior monitoring using most privileged container (mpc),” in 2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud). IEEE, 2017, pp. 351–356.
11. S. A. Hofmeyr, S. Forrest, and A. Somayaji, “Intrusion detection using sequences of system calls,” Journal of computer security, vol. 6, no. 3, pp. 151–180, 1998
12. L. Kuang and M. Zulkernine, “An intrusion-tolerant mechanism for intrusion detection systems,” in 2008 Third International Conference on Availability, Reliability and Security. IEEE, 2008, pp. 319–326.

This is a collaborative work with
Concordia University, Canada

This research is funded by RISE Internal Research Grant

THANK YOU