Machine Learning-Based Container Security Enhancement Technique Using Kernel Tracing Logs

신현석, 조민정, 탁병철

Kyungpook National University (KNU), Daegu, Republic of Korea

2024.07.09

Container & Kernel Bugs

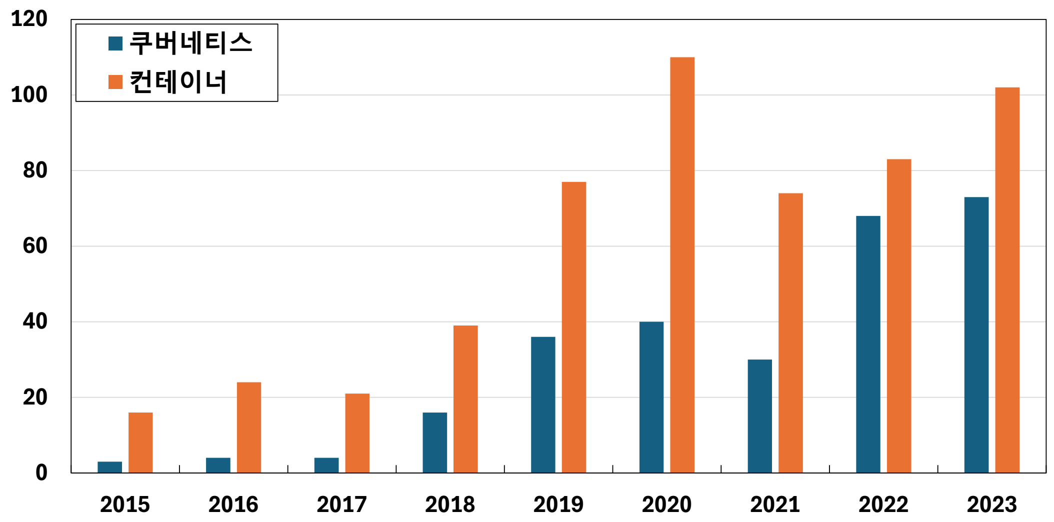
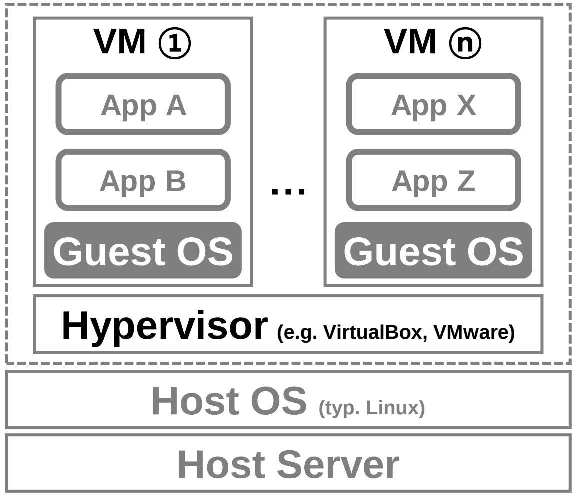
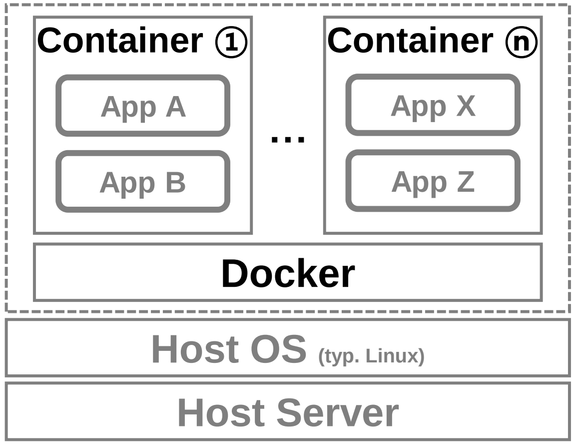


그림1) 연도별 보안 취약점 보고 건수

Steady growth in container utilization

▶ Related security vulnerabilities continue to grow as well





Container & Kernel Bugs

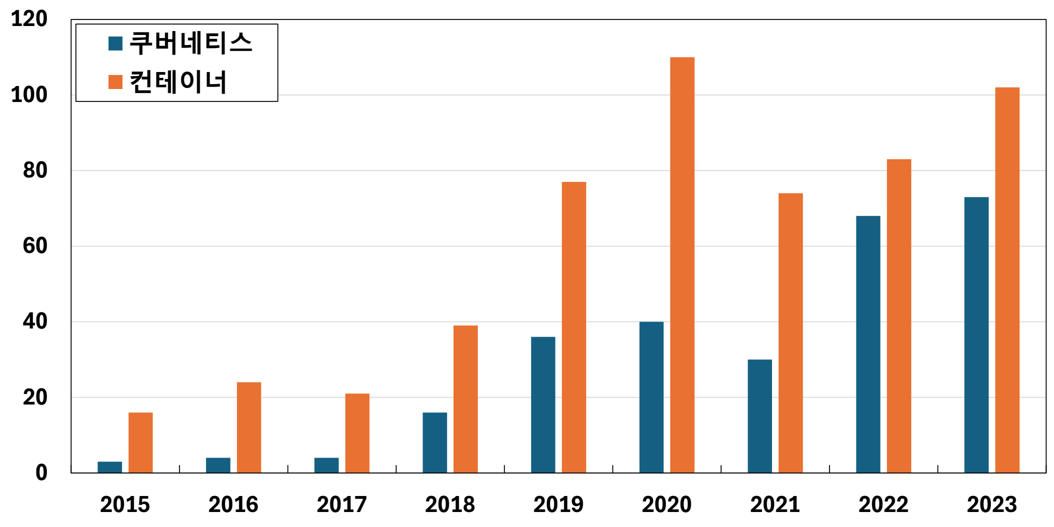


그림1) 연도별 보안 취약점 보고 건수

Steady growth in container utilization

▶ Related security vulnerabilities continue to grow as well

Attackers leverage kernel vulnerabilities to cause **Container Escapes**

▶ Containers ***share a HOST OS***, which can make them much more vulnerable to attacks

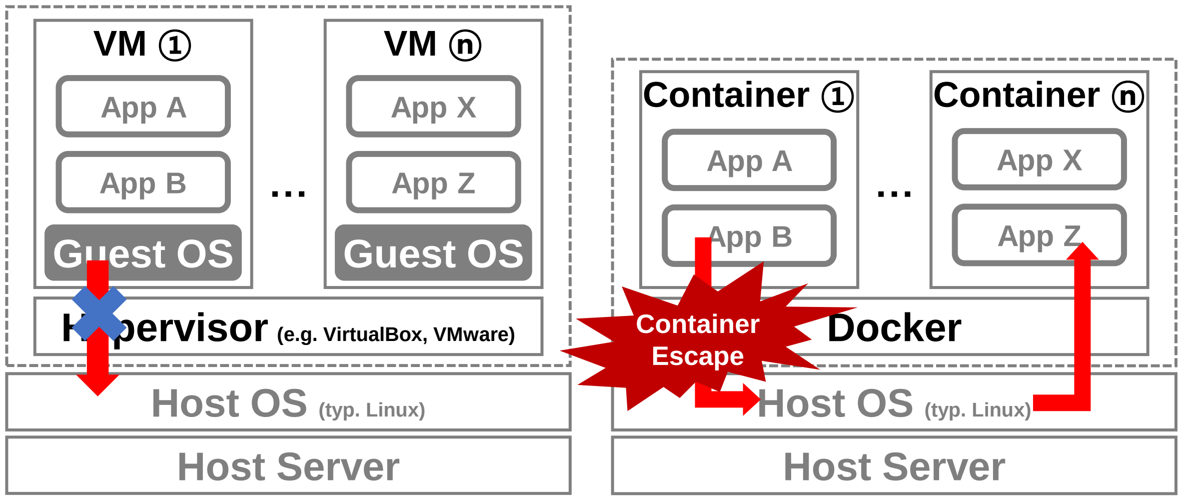


그림1) CVE MITRE 데이터베이스 (https://cve.mitre.org/index.html)

Related Work

sysfilter: Automated System Call Filtering for Commodity Software

- Research on reducing the kernel attack surface by analyzing binary files to limit available system calls

23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)

Reduce the attack surface by creating profiles to **allow** or **restrict specific system calls**

Confine: Automated System Call Policy Generation for Container Attack Surface Reduction

- Research on using static code analysis to create restrictive seccomp syscall policies for docker containers to reduce the kernel attack surface by restricting syscalls

23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)

Related Work

sysfilter: Automated System Call Filtering for Commodity Software

- Research on reducing the kernel attack surface by analyzing binary files to limit available system calls

23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)

Reduce the attack surface by creating profiles to **allow** or **restrict specific system calls**

Confine: Automated System Call Policy Generation for Container Attack Surface Reduction

- Research on using static code analysis to create restrictive seccomp syscall policies for docker containers to reduce the kernel attack surface by restricting syscalls

23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)

Difficult to create profiles for each system

Ongoing maintenance with updates

Attacks are possible with some combination of allowed system calls

Motivation

Past

Requires modifying kernel code and writing kernel modules to perform code inside the kernel

- High level of expertise required
- Very large costs incurred when kernel problems occur

Current

- extended Berkeley Packet Filter (eBPF) technology to write code safely inside the kernel

- There are differences between normal processes (e.g. database) and abnormal processes (e.g. attack purposes such as CVE poc code) while they are running

Motivation

Past

Requires modifying kernel code and writing kernel modules to perform code inside the kernel

- High level of expertise required
- Very large costs incurred when kernel problems occur

Current

- extended Berkeley Packet Filter (eBPF) technology to write code safely inside the kernel

- There are differences between normal processes (e.g. database) and abnormal processes (e.g. attack purposes such as CVE poc code) while they are running

GOAL

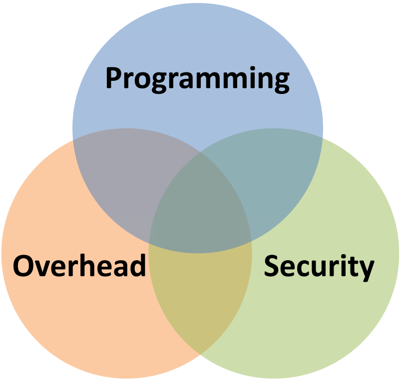
Create a kernel event tracing program using eBPF
to collect kernel tracing logs from normal and abnormal processes

Use the collected logs and machine learning to detect abnormal processes with a **low false positive rate**

Background

- **extended Berkeley Packet Filter (eBPF)**
  - Observe various kernel events without modifying the kernel
  - Hook multiple events occurring within the kernel
  - Enhance security and stability through continuous maintenance
  - Benefit from a growing set of helper functions for safe kernel-level programming

- Programming within the kernel **without kernel modifications**



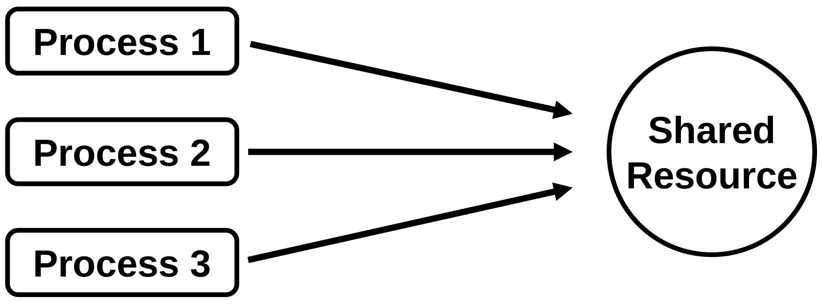
- Code safety check through the verifier
- Provision of various helper functions that do not harm the kernel

- No context switch occurs
- Utilization of Just-In-Time (JIT) compiler

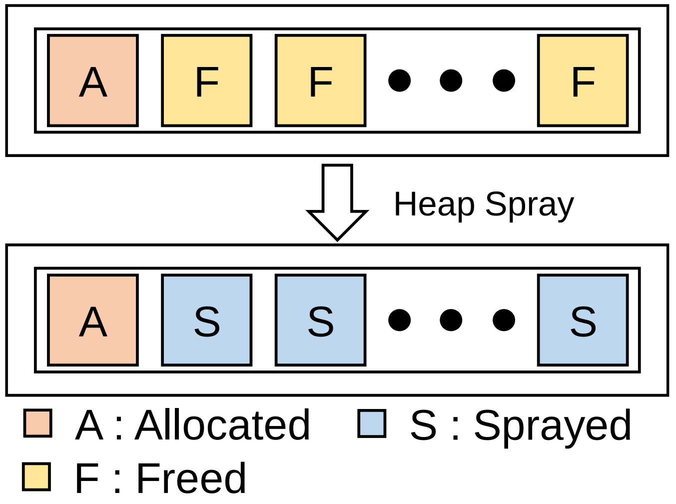
- **Boosting**
  - Ensemble learning technique where weak learners, like decision trees, are sequentially trained and combined to form a stronger model
  - E.g. AdaBoost, XGBoost

Attack Techniques

- **Race condition**
  - A phenomenon in which two or more processes or threads compete to use a shared resource (race)
  - The execution order of processes or threads is unpredictable, leading to repeated execution until a race condition occurs
  - E.g. Dirty COW(CVE-2016-5195), Dirty Cred(CVE-2021-4154, CVE-2022-2588)

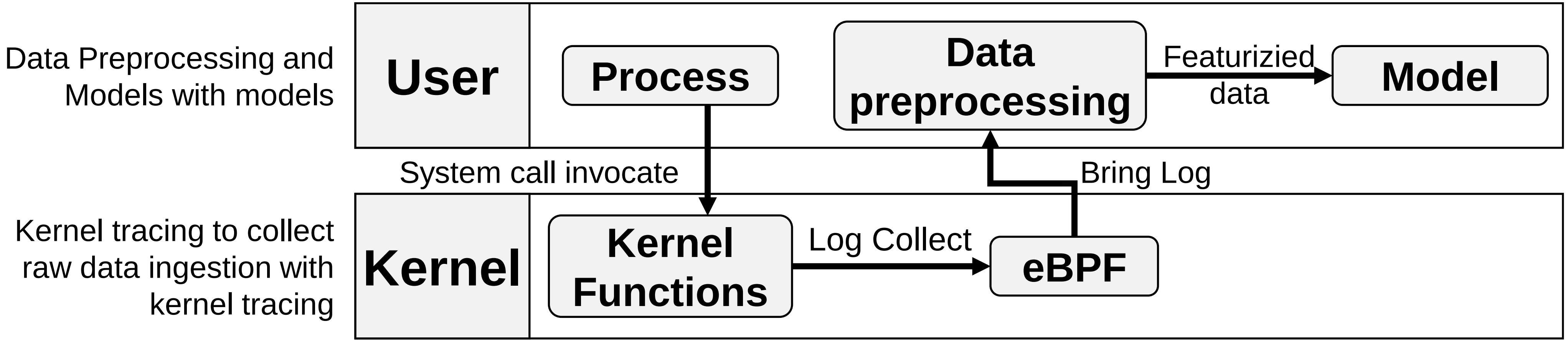


- **Heap spray**
  - A technique for increasing the success rate of a kernel memory-related attack (e.g., Out-Of-Bounds, Use-After-Free) by allocating large amounts of data to targeted kernel memory regions, making the memory predictable to the attacker
    - ► Out-Of-Bounds (OOB): Accessing memory outside the boundaries of an array to create a security issue
    - ► Use-After-Free (UAF): Reusing freed memory to cause unexpected behavior and security vulnerabilities



Architecture

- **Workflow**
  - Processes raise events to the kernel via system calls, and eBPF catches the events and collects the event information into logs
  - The user area periodically fetches logs from eBPF's storage and preprocesses them for use in machine learning models
  - The machine learning model uses the preprocessed data to determine whether a process is normal or abnormal



Feature Extraction

- **Race condition feature**

Feature English	Feature	Race condition Characteristic
SYSCALL_TOTAL_COUNT	시스템콜 총 호출수	호출 수가 큼
SYSCALL_ARGUMENT_SIMILAR	유사한 시스템콜 호출 정도	유사한 시스템콜을 연속적으로 호출
SYSCALL_KIND_SIMILAR	유사한 인자 사용 정도	유사한 인자를 사용
SYSCALL_KIND	사용한 시스템콜 종류	적은 종류의 시스템콜 사용
SYSCALL_TOP1~5	1~5번째로 많이 사용된 시스템콜 호출 횟수	1~3번째 시스템콜 호출 값이 큼

- **Heap spray feature**

Feature English	Feature	Heap spray Characteristic
KMALLOC_TOTAL_COUNT	kfree로 해제되지 않은 kmalloc으로 할당된 slub 개수	해제되지 않는 slub의 개수가 많음
KMALLOC_COUNT	kmalloc 호출 수	호출 수가 큼
KFREE_COUNT	kfree 호출 수	호출 수가 적음
KMALLOC_KIND	kmalloc으로 할당된 slub 크기의 종류	적은 종류의 slub을 할당받음
KMALLOC_TOP_ENTROPY	상위 3개의 할당된 slub 크기의 엔트로피	엔트로피 값이 낮음
KMALLOC_TOP1~3	1~3번째로 많이 할당된 slub 크기의 개수	1번째 값이 큼

Analysis of Feature Observation Results

- **Race condition**
  - Both normal and abnormal processes had a high count of system calls invoked, but the types of system calls used varied for normal processes

Feature	ABNORMAL				NORMAL					
	DirtyCOW ptrace	DirtyCOW write	DirtyCred 2021-4154	DirtyCred 2022-2588	alphine	MongoDB	nginx	MySQL	PostgreSQL	Redis
SYSCALL_TOTAL_COUNT	606.72	2,744.69	1,260.89	73.27	28.73	7,946.05	2,092.69	79.22	6.56	31.75
SYSCALL_ARGUMENT_SIMILAR	1.00	1.00	0.97	0.75	0.45	0.93	0.67	0.47	0.26	0.74
SYSCALL_KIND_SIMILAR	1.00	0.24	0.97	0.03	0.60	0.40	0.09	0.46	0.22	0.47
SYSCALL_KIND	1.00	2.45	1.20	2.03	5.80	21.22	9.83	25.82	3.03	2.19
SYSCALL_TOP1	606.72	1,421.05	1,260.54	50.11	12.15	1,202.56	340.49	17.86	2.75	15.87
SYSCALL_TOP2	0.00	914.86	0.10	23.05	5.73	845.60	340.02	7.08	1.69	11.35
SYSCALL_TOP3	0.00	408.79	0.08	0.04	3.94	826.28	207.75	5.40	1.10	4.48
SYSCALL_TOP4	0.00	0.00	0.03	0.02	3.02	810.10	196.43	4.48	0.72	0.04
SYSCALL_TOP5	0.00	0.00	0.03	0.01	1.51	785.27	186.42	3.87	0.23	0.02

- **Heap spray**
  - Abnormal case, can see high count of unreleased slubs and high count of kmalloc allocations
  - Normal case, can see that the count of kfree invokes is high even when the count of kmalloc invokes is high

Feature	ABNORMAL				NORMAL				
	2022_3910	2023_2008	2023_2598	2023_32233	MongoDB	MySQL	PostgreSQL	Redis	Confusion Data
KMALLOC_TOTAL_COUNT	184.50	437.16	1,683.02	3,197.60	6.00	3.45	6.70	1.50	68.65
KMALLOC_COUNT	184.75	528.61	1,683.05	3,197.60	6.00	4.07	7.30	1.58	351.90
KFREE_COUNT	0.25	91.45	0.03	0.00	0.00	0.62	0.60	0.08	283.25
KMALLOC_KIND	2.38	5.14	1.03	5.00	2.00	1.03	3.72	1.50	1.90
KMALLOC_TOP_ENTROPY	0.96	1.36	0.00	1.03	1.00	0.02	1.38	0.26	0.43
KMALLOC_TOP1	110.13	216.63	1,682.98	1,591.50	3.00	3.44	3.30	1.00	52.85
KMALLOC_TOP2	73.88	130.64	0.03	1,591.40	3.00	0.03	2.21	0.17	14.75
KMALLOC_TOP3	0.63	88.36	0.00	12.70	0.00	0.00	0.86	0.17	1.05

Evaluation

- **Rule-based testing**

	Precision (%)	Recall (%)	F1-score (%)
Race Condition	98.47	33.20	49.65
Heap Spray	98.73	44.39	61.25

- **Model testing**

Race Condition	Precision (%)	Recall (%)	F1-score (%)
AdaBoost	99.99 (▲ 1.52)	99.83 (▲ 66.63)	99.91 (▲ 50.26)
XGBoost	99.97 (▲ 1.5)	99.87 (▲ 66.67)	99.92 (▲ 50.27)

Heap Spray	Precision (%)	Recall (%)	F1-score (%)
AdaBoost	100 (▲ 1.27)	100 (▲ 55.61)	100 (▲ 38.75)
XGBoost	100 (▲ 1.27)	99.28 (▲ 54.89)	99.64 (▲ 38.39)

Model vs. Rule-Based Test Results

- **Precision**: Increase of approximately **2%**
- **Recall**: Increase of approximately **50-70%**
- **F1-Score**: Increase of approximately **40-50%**

Conclusion and Future Research

- **Conclusion**
  - Developed a system using machine learning and eBPF to **detect race conditions and heap spray attacks without modifying the kernel**
  - Achieves a **low false positive rate** near 0 and a high detection rate near 1
  - Provides **near real-time detection with eBPF**

- **Limitation**
  - Lower kernel versions have limitations in running eBPF, making it difficult to detect attacks
  - Models are not available in the kernel area

- **Future research**
  - Detect other attack techniques