UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

itee PhD
information technology
electrical engineering

DIETI UNINA

Carmine Cesarano

Reducing Software Attack Surfaces via Security Hardening, Fuzzing and Runtime Enforcement

Tutor: Roberto Natella

Cycle: XXXVIII

Year: Third

itee PhD
information technology
electrical engineering

Candidate's information

- MSc degree in **Computer Engineering** (June 2022)
- **Research group**: Dependable and Secure Software Engineering and Real-Time Systems (DESSERT)
- **PhD start-end date**: 01/11/25 – 31/10/2025
- **Scholarship type**: UNINA
- **Periods abroad**: 6 months at KTH Royal Institute of Technology in Stockholm

Summary of study activities

Ad hoc PhD courses

- IoT Data Analysis
- Virtualization Technologies and their Applications
- Statistical Data Analysis for Science and Engineering Research
- Percorso per il rafforzamento delle competenze sulla progettazione europea
- Strategic Orientation for STEM Research & Writing
- Using Deep Learning Properly
- Innovation and Entrepreunership
- AI Code Generation: Foundations, Evaluation, and Security

Summary of study activities

Conferences attended

- 24th IEEE International Symposium on Software Reliability Engineering (ISSRE 2024)
- 19th European Dependable Computing Conference (EDCC 2024)
- ACM Conference on Computer and Communications Security (CCS2024)
- 36th IEEE International Symposium on Software Reliability Engineering (ISSRE 2025)

Research areas

- **Security Testing** of Industrial Software Systems
  - General-purpose, open-source hypervisors for cloud environments
  - Specialized, closed-source hypervisors for defence
  - Industrial firewall systems for railways
- **Security Hardening** of Container Orchestration Platforms
  - Container Orchestrators like Kubernetes used in cloud
- **Runtime Security** for Open-Source Software Supply Chain
  - Golang programming language widely used for cloud infrastructures

Research results

- Comprehensive analysis of security challenges and mitigation opportunities in edge computing.
- Methodological framework for testing Application Layer Gateways used in critical domains.
- Methodological framework for assessing and certifying isolation properties in partitioning hypervisors.
- Fuzzing framework for testing encrypted network protocols like OpenVPN (**FuEL**).

Thesis

- Fuzzing framework for testing open-source, general purpose hypervisors (**IRIS**).
- Fuzzing framework for testing closed-source, industrial hypervisors (**FuzzBox**).
- API filtering system for reducing the attack surface of Kubernetes (**KubeFence**)
- Language-specific taxonomy of supply chain attack vectors in Go and static analysis tool for identifying them (**GoSurf**)
- Runtime enforcement system mitigating supply chain attacks in Go (**GoLeash**)

Research products (1)

[P1]	C. Cesarano, D. Cotroneo, L. De Simone Towards Assessing Isolation Properties in Partitioning Hypervisors ***33rd IEEE International Symposium on Software Reliability Engineering (ISSRE2022)*** Charlotte, NC, USA, 31 October–03 November 2022, IEEE, DOI: 10.1109/ISSREW55968.2022.00067
[P2]	C. Cesarano, M. Cinque, D. Cotroneo, L. De Simone, G. Farina IRIS: a Record and Replay Framework to Enable Hardware-assisted Virtualization Fuzzing ***53rd IEEE/IFIP International Conference on Dependable Systems and Networks (DSN2023)*** Porto, Portugal, 27-30 June 2023, IEEE, DOI: 10.1109/DSN58367.2023.00045
[P3]	C. Cesarano Security Assessment and Hardening of Fog Computing Systems ***34th IEEE International Symposium on Software Reliability Engineering (ISSRE2023)*** Florence, Italy, 09-12 October 2023, IEEE, DOI: 10.1109/ISSREW60843.2023.00037
[P4]	C. Carmine, R. Natella Securing an Application Layer Gateway: An Industrial Case Study ***19th European Dependable Computing Conference (EDCC2024)*** Leuven, Belgium, 08-11 April 2024, IEEE, DOI: 10.1109/EDCC61798.2024.00025
[P5]	C. Cesarano, V. Andersson, R. Natella, M. Monperrus GoSurf: Identifying Software Supply Chain Attack Vectors in Go ***ACM Workhop on Software Supply Chain Offensive Research and Ecosystem Defenese (SCORED24)*** Salt Lake City, Utah, USA, 14-18 October 2024, ACM, DOI: 10.1145/3689944.3696166

Research products (2)

[P6]	C. Cesarano, A. Foggia, G. Roscigno, L. Andreani, R. Natella GENIO: Synergizing Edge Computing with Optical Network Infrastructures ***IEEE Communication Magazine (COMMAG-IEEE)*** Volume 63, Issue 7, Pages: 154-160, July 2025, IEEE, DOI: 10.1109/MCOM.002.2400382
[P7]	C. Cesarano, R. Natella KubeFence: Security Hardening of the Kubernetes Attack Surface. ***55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*** Naples, Italy, 23-26 June 2025, IEEE, DOI: 10.1109/DSN64029.2025.00054
[P8]	C. Cesarano, A. Foggia, G. Roscigno, L. Andreani, R. Natella Security-by-Design at the Telco Edge with OSS: Challenges and Lessons Learned ***55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*** Naples, Italy, 23-26 June 2025, IEEE, DOI: 10.1109/DSN64029.2025.00054
[P9]	C. Cesarano, R. Natella FuzzBox: Blending Fuzzing into Emulation for Binary-Only Embedded Targets **Springer Cybersecurity Journal** (accepted, to appear)

PhD thesis overview

Problem statement

Modern software systems expose large, multi-layered attack surfaces, that current methods fail to effectively reduce

Objective

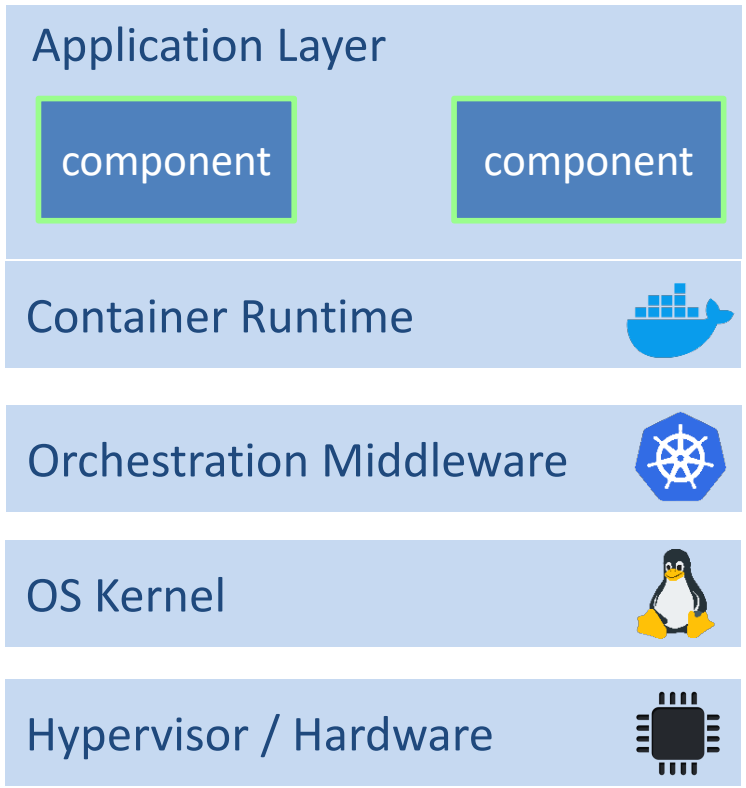
Develop cross-layer, fine-grained, and automated techniques for attack surface reduction that enhance system security without compromising functionality

Methodology

- Characterize attack surfaces and identify layer-specific weaknesses
- Design and prototype five complementary techniques combining static and dynamic analysis
- Evaluate their effectiveness through real-world deployments, benchmarks, vulnerability discovery, and attack mitigation

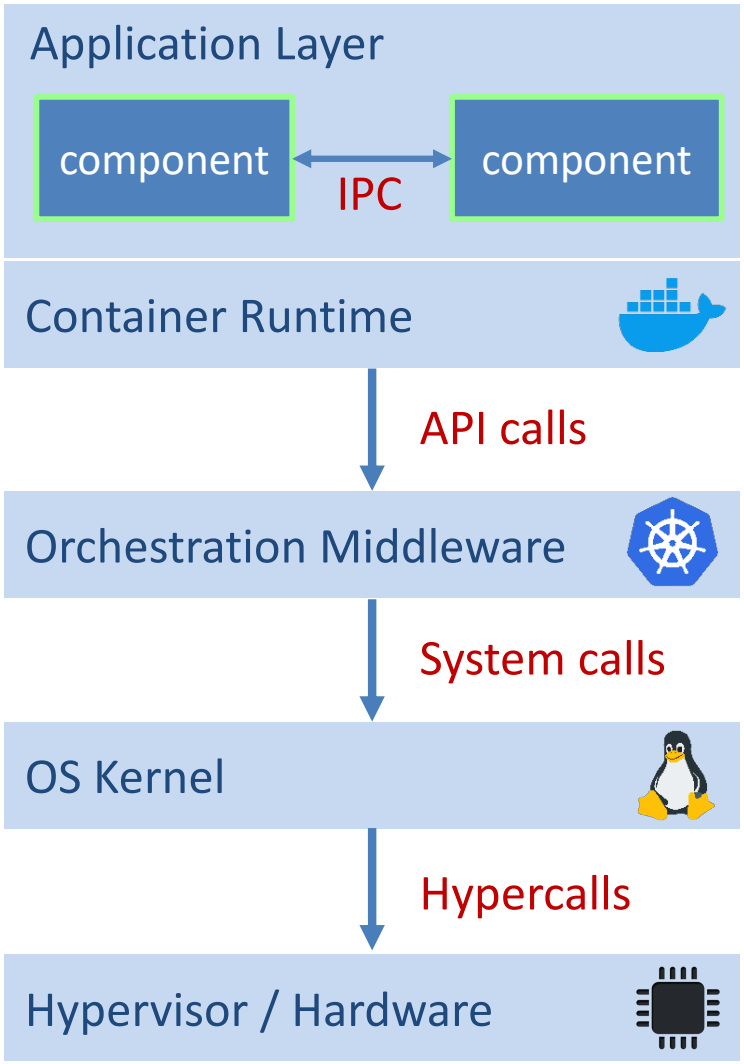
Context

Modern Software Stacks: Complexity and Exposure



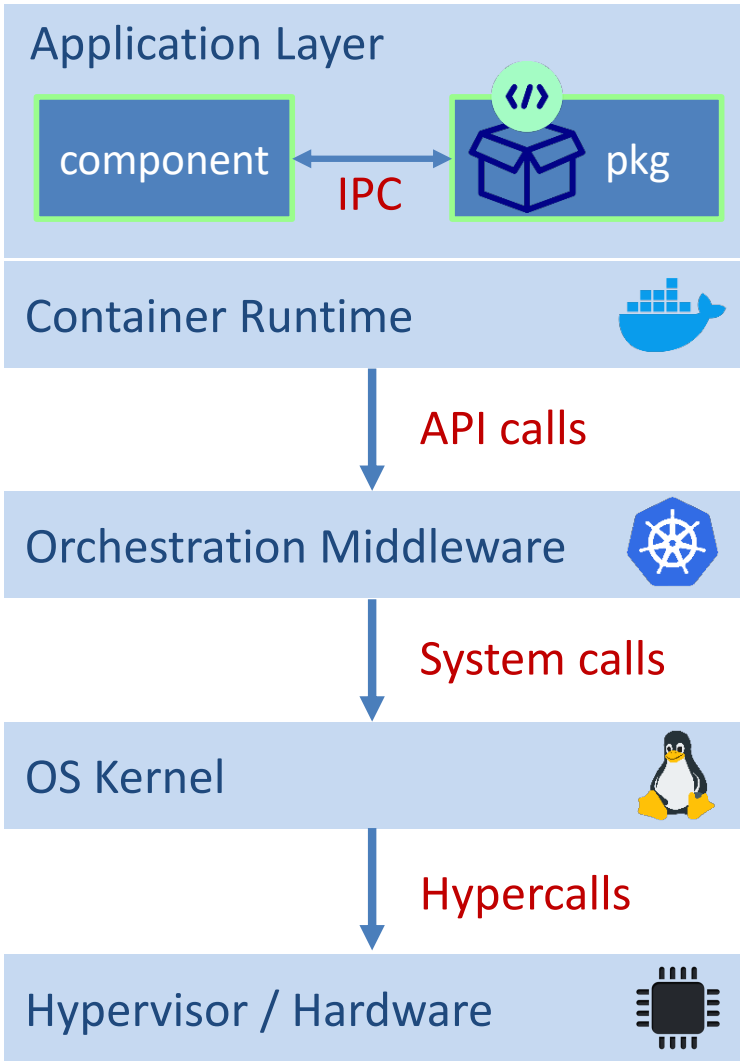
Context

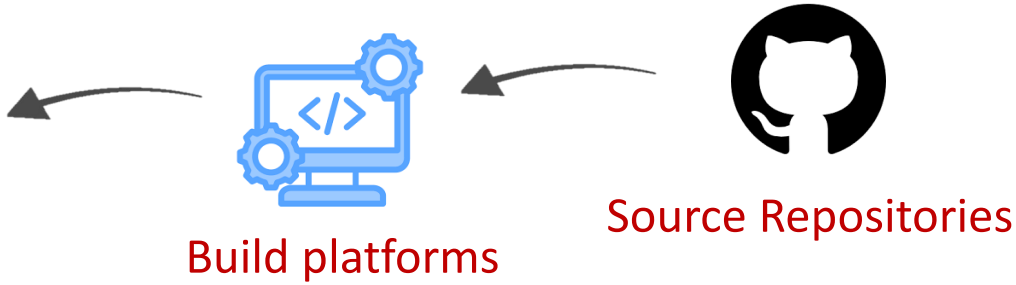
Modern Software Stacks: Complexity and Exposure



Context

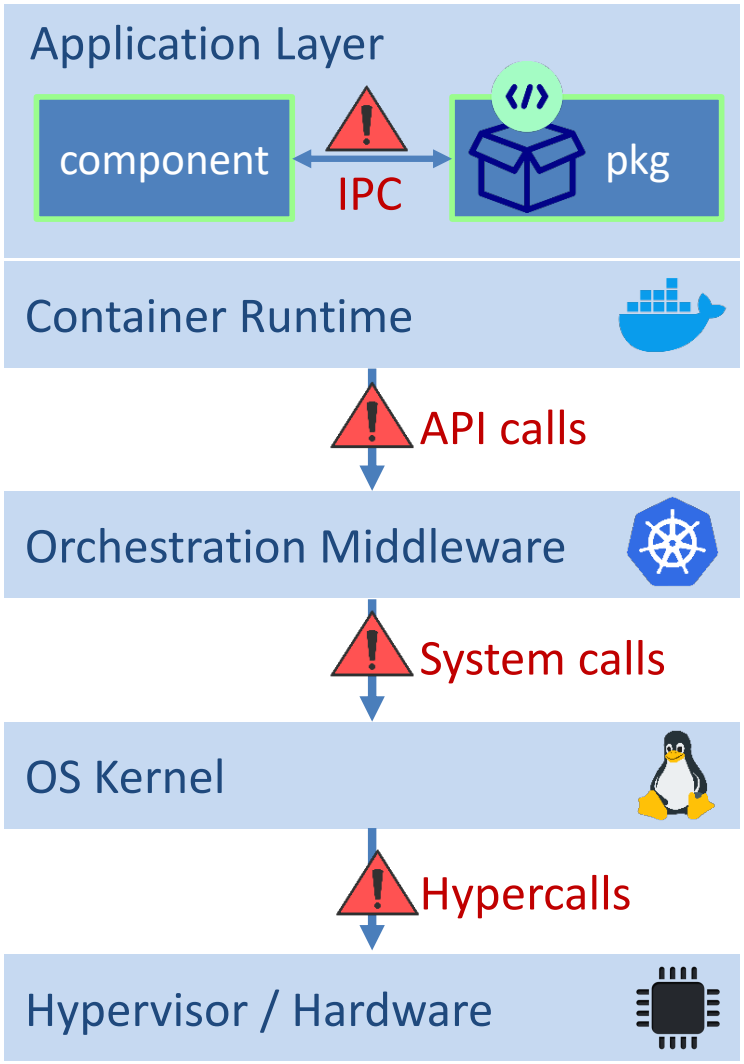
Modern Software Stacks: Complexity and Exposure

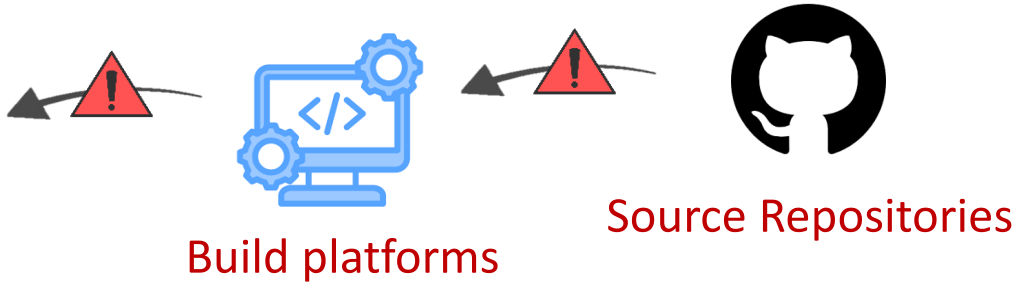




Research Problem

Modern Software Stacks: Complexity and Exposure





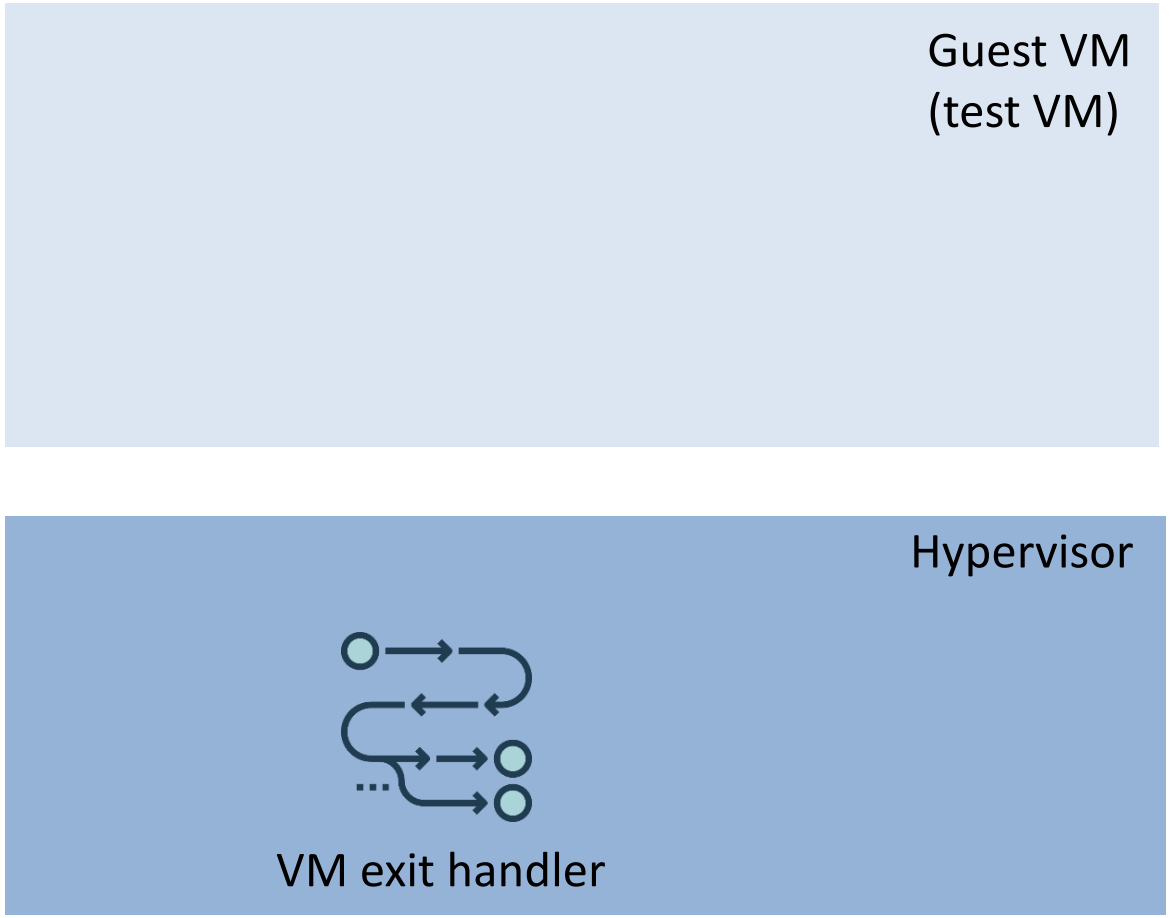
Interfaces represent attack surfaces

How can we systematically reduce them?

Contibution 1

IRIS: Fuzzing the Hypervisor Surface

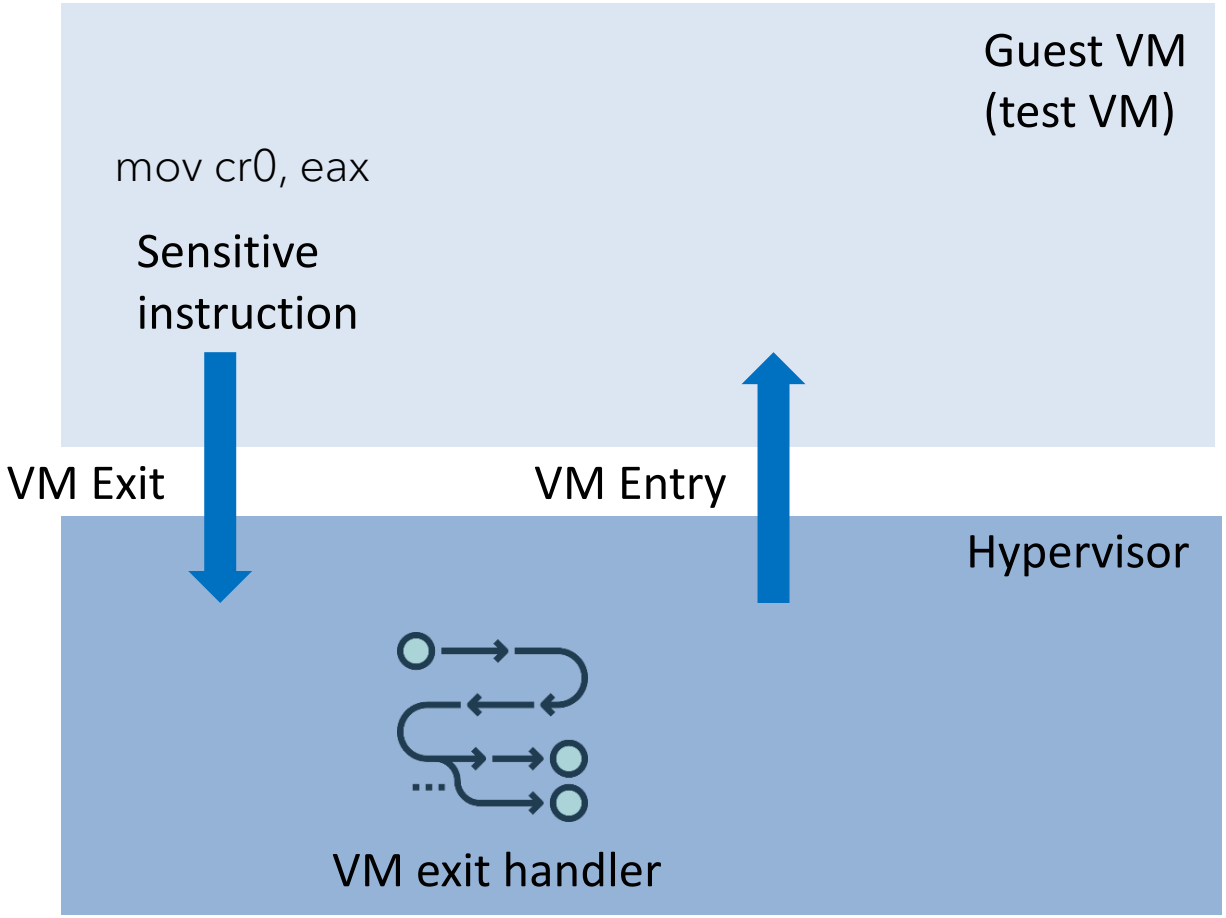
Testing hypervisor logic is stateful and hard



Contibution 1

IRIS: Fuzzing the Hypervisor Surface

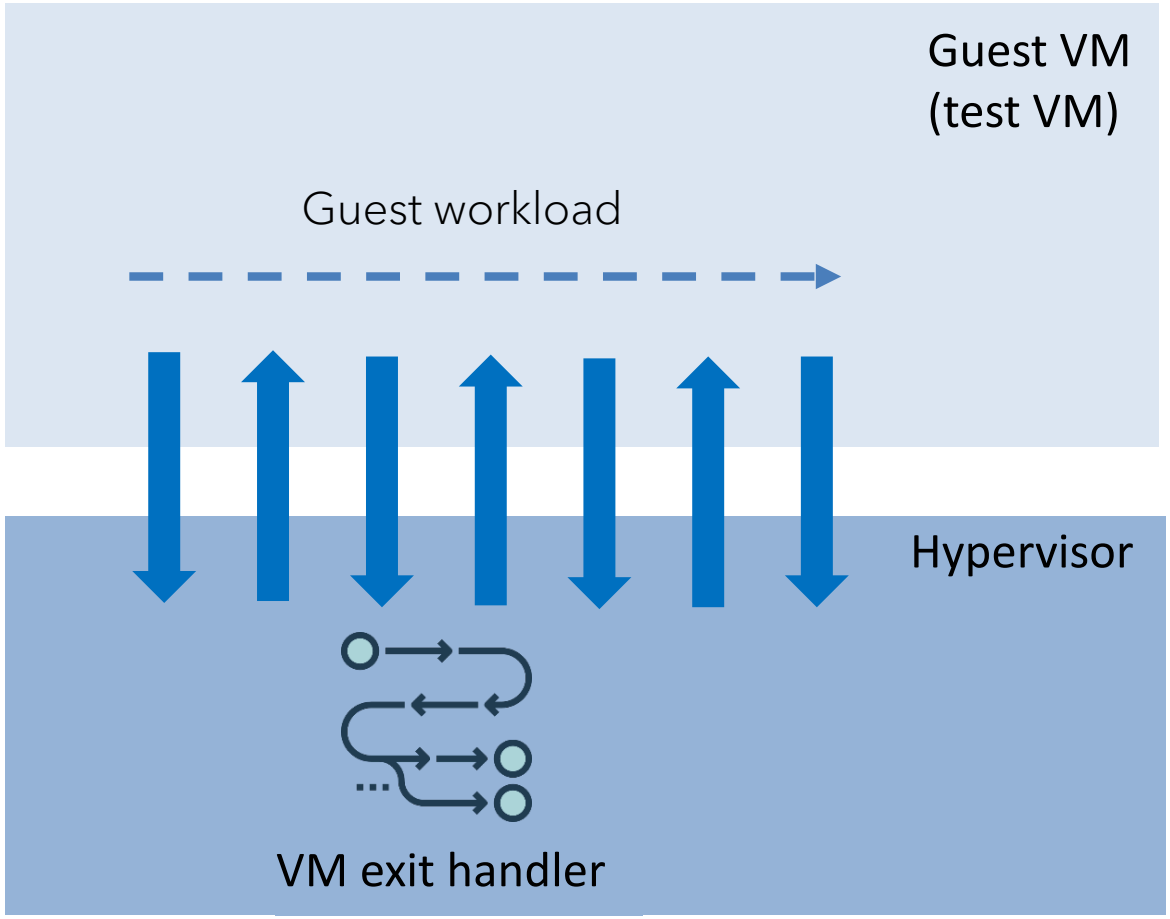
A single VM exit handling depends on the guest VM state, sensitive instruction invoked, hypervisor current state and previous VM exits



Contibution 1

IRIS: Fuzzing the Hypervisor Surface

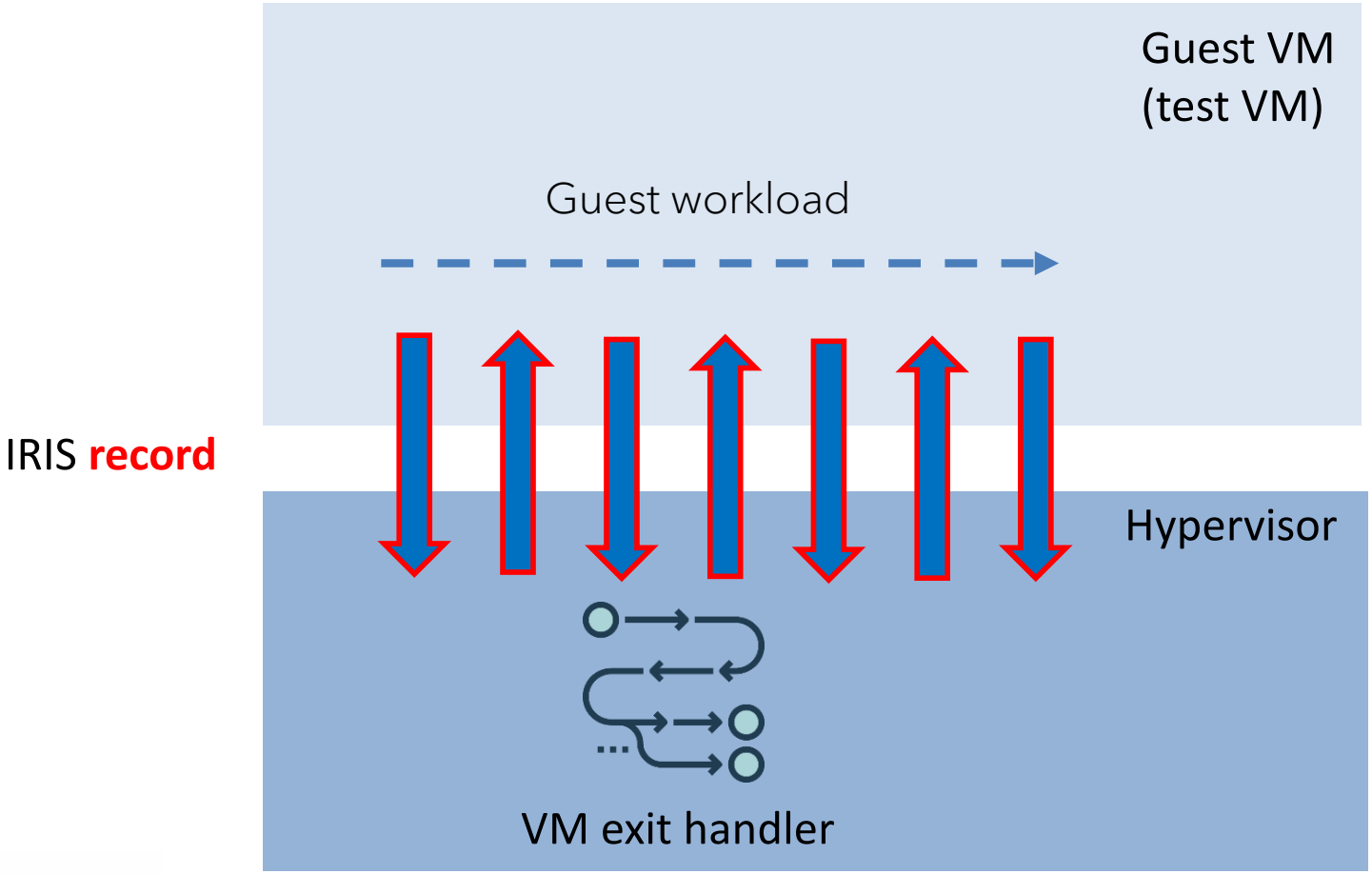
Reproducing that sequence from a test guest requires re-implementing OS-level flows (prohibitive)



Contibution 1

IRIS: Fuzzing the Hypervisor Surface

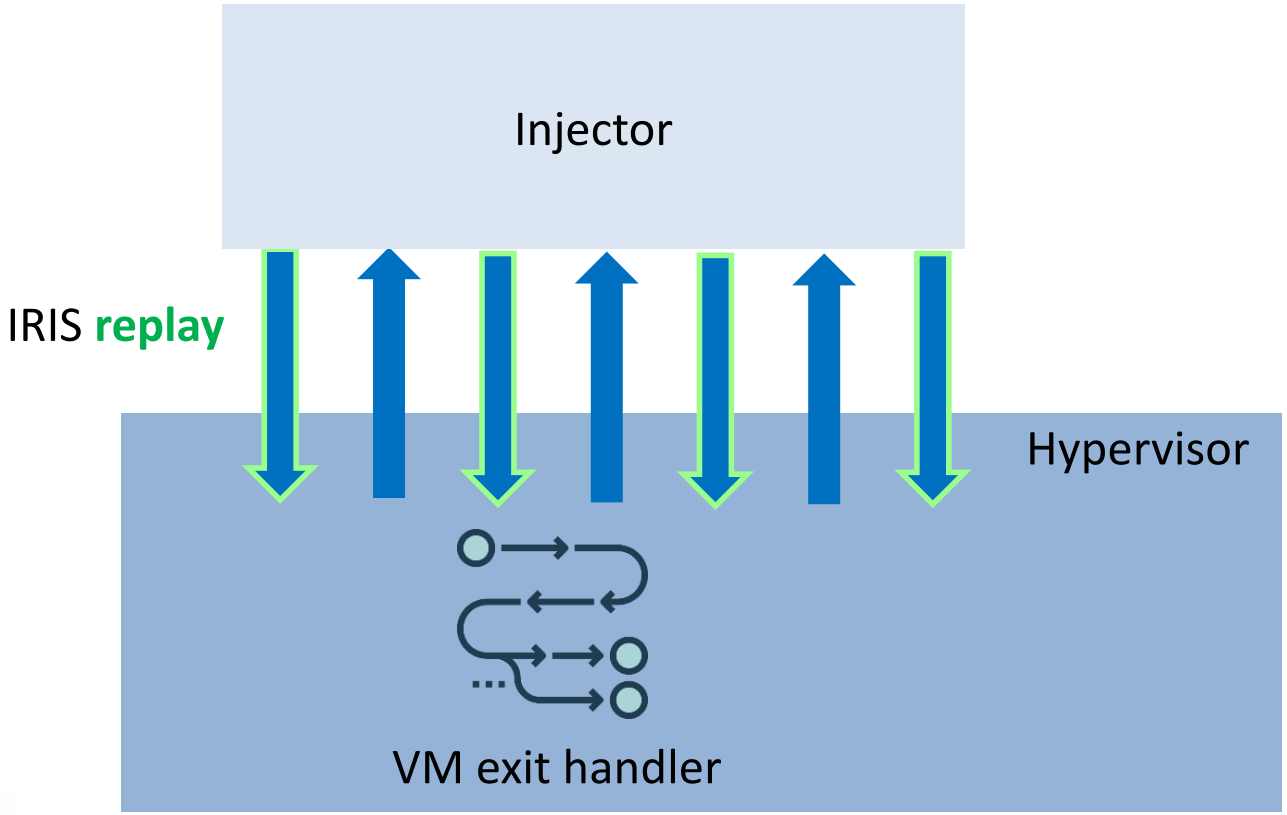
IRIS record VM exit events during hypervisor intervention by executing a nominal guest workload



Contibution 1

IRIS: Fuzzing the Hypervisor Surface

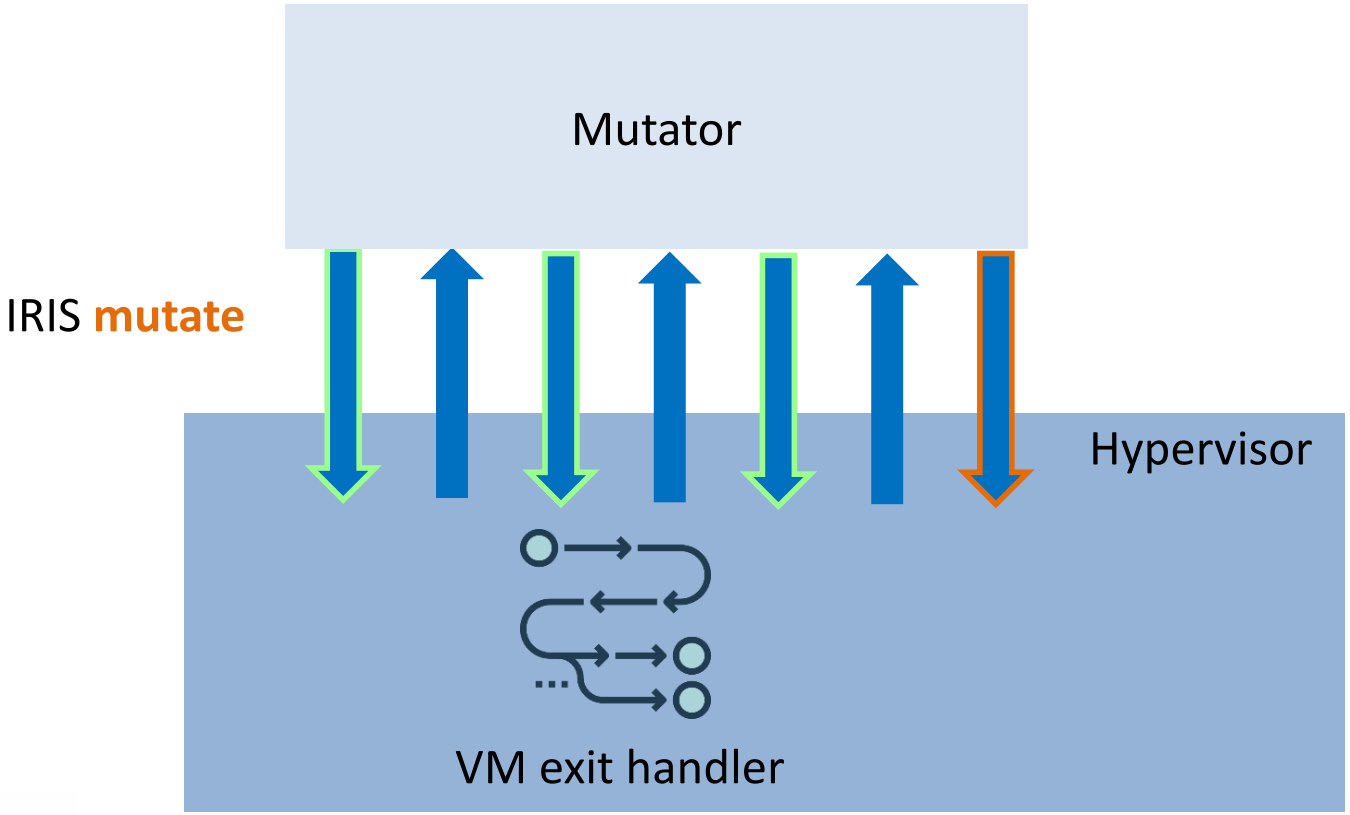
IRIS replay such events to quickly reach deep hypervisor states



Contibution 1

IRIS: Fuzzing the Hypervisor Surface

IRIS can mutate VM exits in the replayed sequence to test corner cases, facilitating fuzzing of the hypervisor logic



Contibution 1

IRIS: Fuzzing the Hypervisor Surface

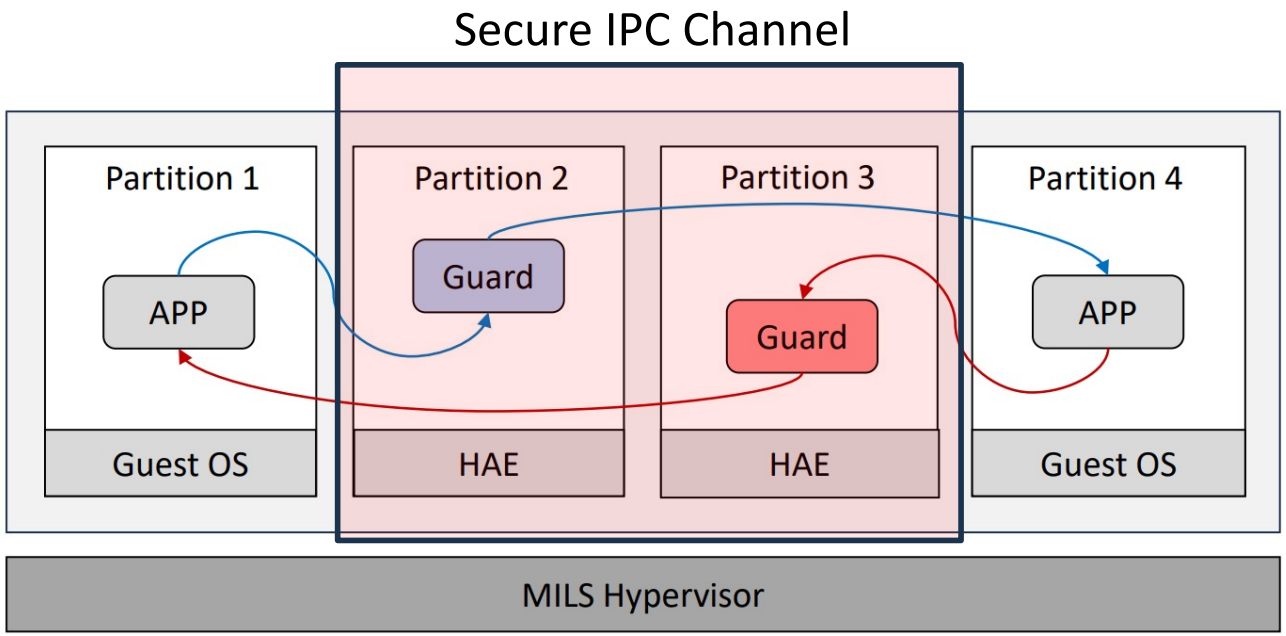
Evaluations on three real guest workloads:

- Reproduces valid hypervisor states identical to real guest executions by replaying VM exit sequences
- Matches 92.2-100% of code coverage of real guest workload
- Achieves 42.5–99.6% faster execution time compared to full guest runs

Contribution 2

FuzzBox: Fuzzing the Syscall/IPC Surface

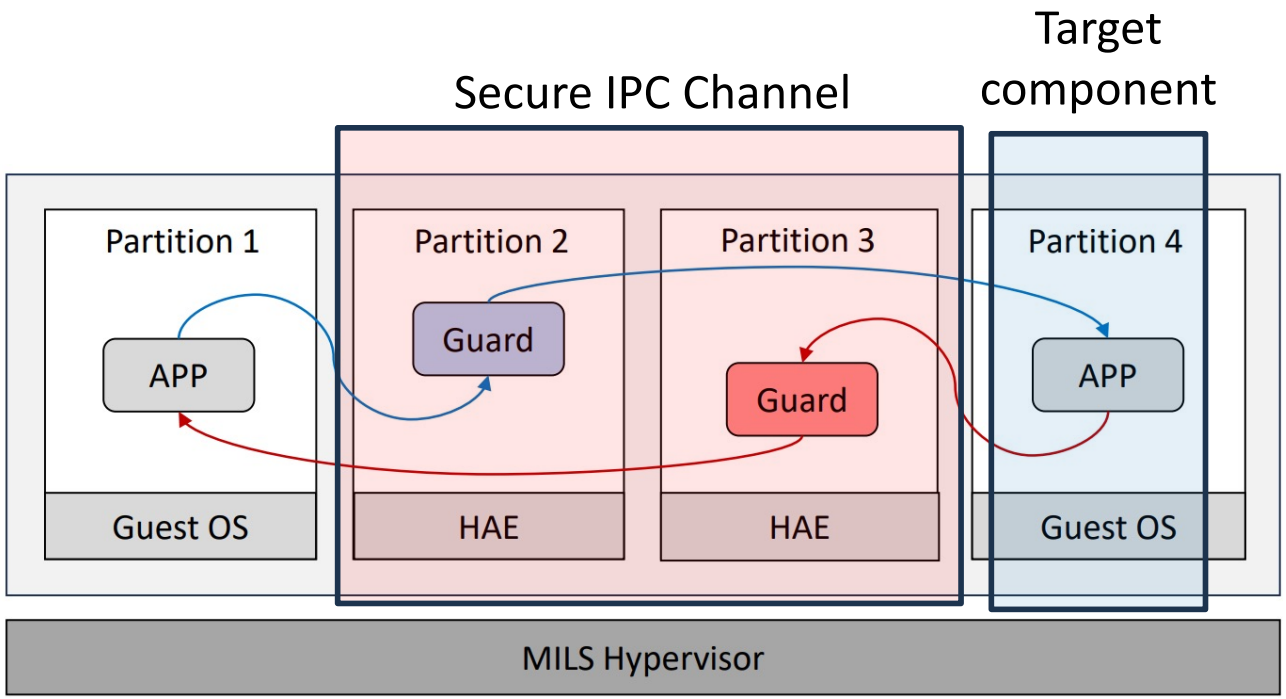
Testing internal opaque components within closed-source binaries is challenging



Contribution 2

FuzzBox: Fuzzing the Syscall/IPC Surface

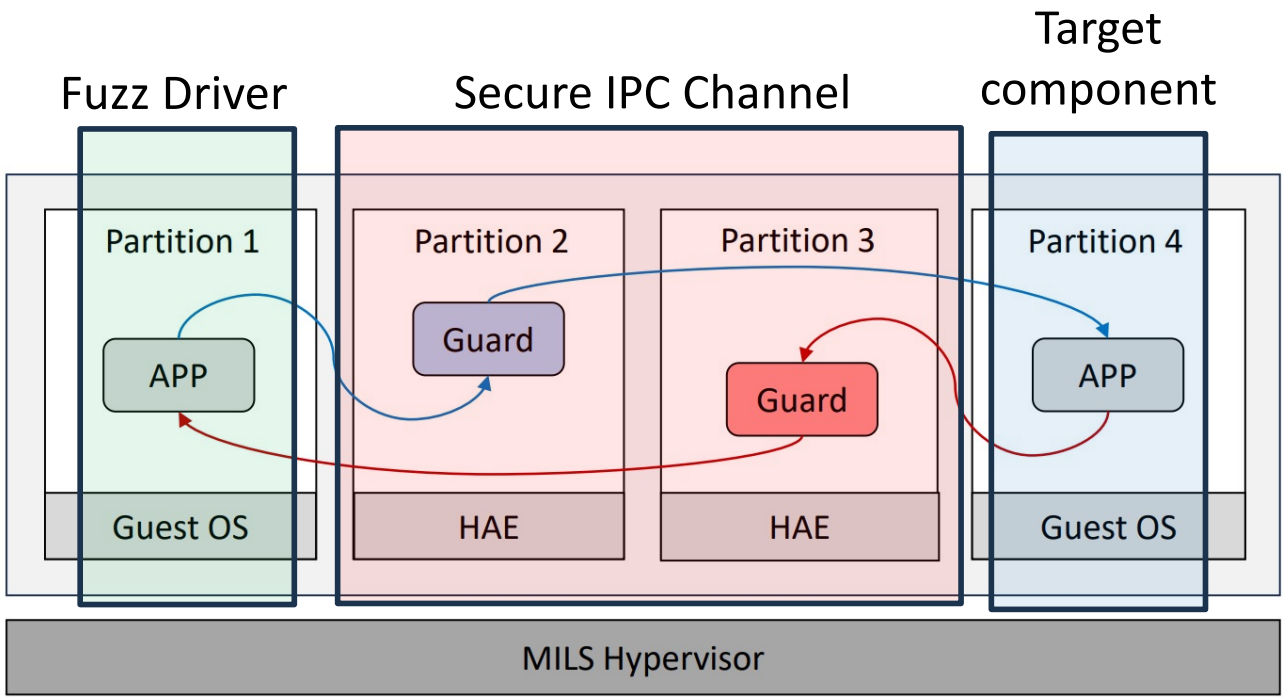
Testing internal opaque components within closed-source binaries is challenging



Contribution 2

FuzzBox: Fuzzing the Syscall/IPC Surface

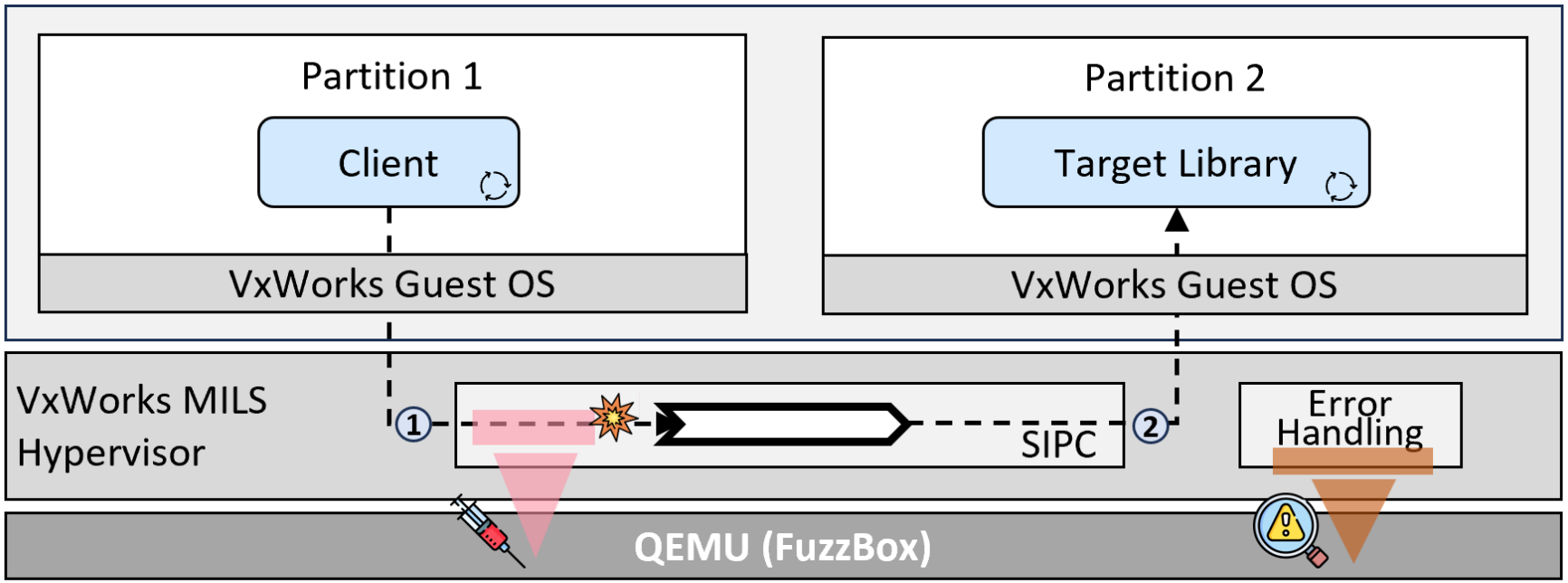
Challenges of Intrusiveness, Toolchain Dependency and Hardware Dependency make fuzzing of industrial, closed-source binaries infeasible



Contribution 2

FuzzBox: Fuzzing the Syscall/IPC Surface

FuzzBox integrate fuzzing with emulation to intercept and mutate internal function calls, gather branch coverage transparently, and monitor error handling



Contribution 2

FuzzBox: Fuzzing the Syscall/IPC Surface

Before FuzzBox fuzzing internal components in MILS-based systems required:

- Adding a fuzz driver inside a partition
- Rebuilding the entire system binary
- Performing black-box fuzzing

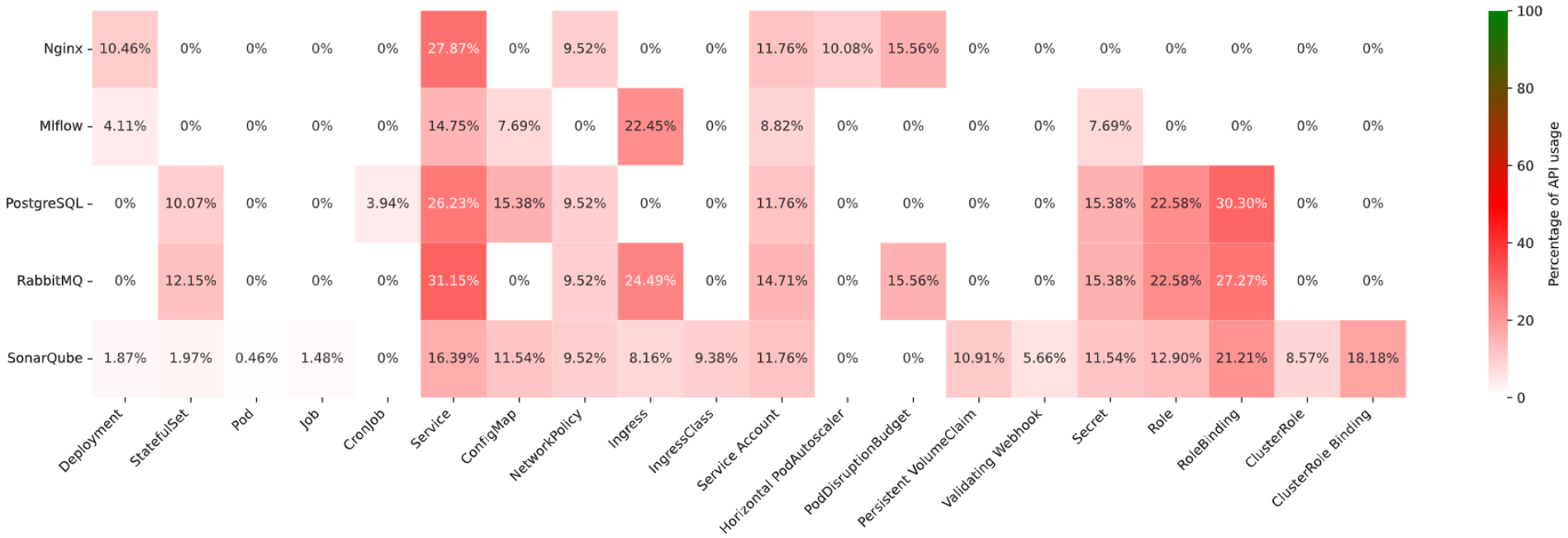
With FuzzBox

- Enabled direct gray-box fuzzing of internal components without rebuilds
- 2X higher fuzzing throughput
- ~58% coverage improvement
- Faster bug discovery

Contribution 3

KubeFence: Hardening the Orchestration Surface

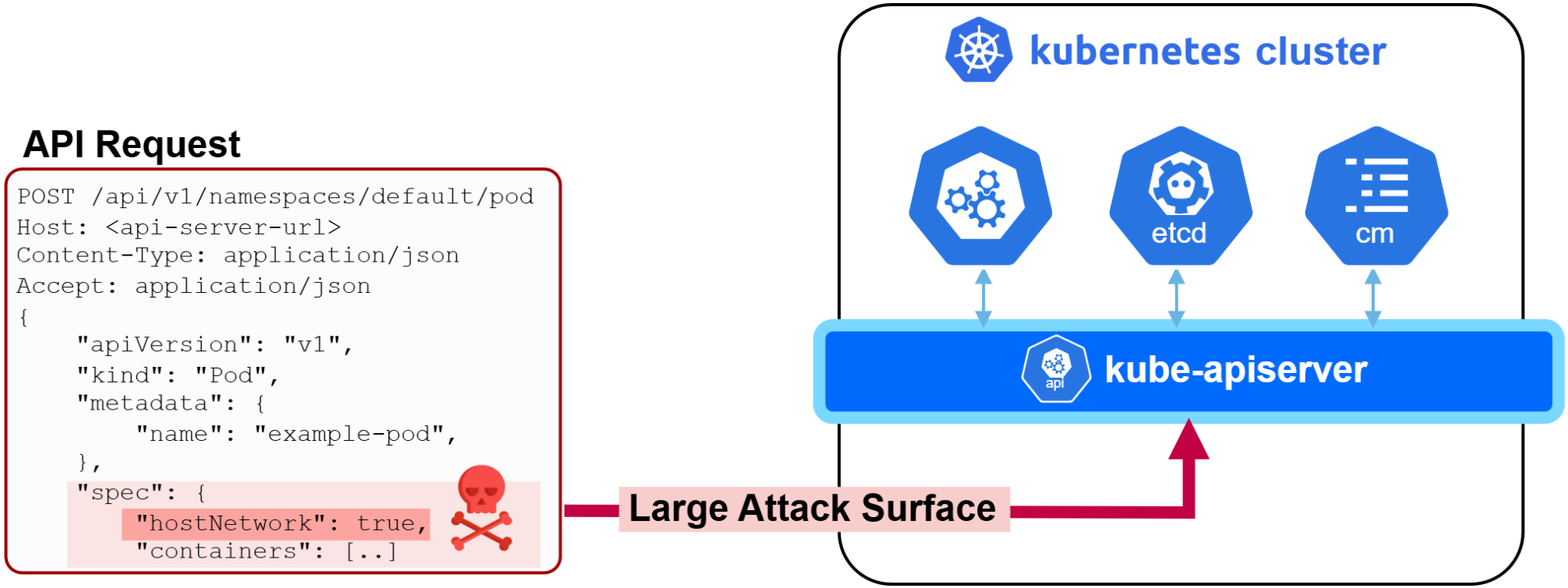
Most of the Kubernetes API is unused by still attackable



Contribution 3

KubeFence: Hardening the Orchestration Surface

Current methods to harden orchestration APIs (e.g., RBAC) are coarse-grained

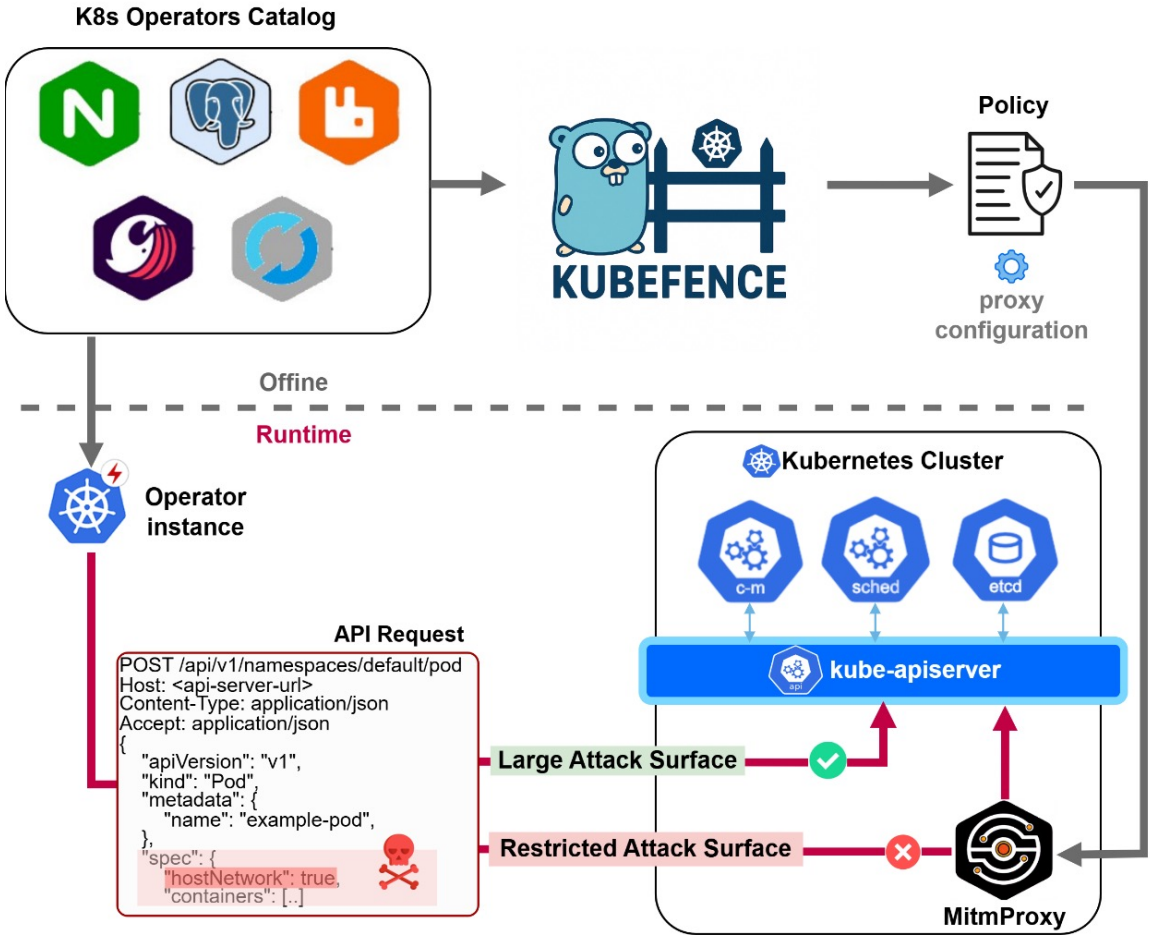


RBAC stops resource kind **not specs!**

Contribution 3

KubeFence: Hardening the Orchestration Surface

KubeFence learns access policies at the level of spec, for a specific K8s app, and apply runtime enforcement allowing only required APIs fields and values



Contribution 3

KubeFence: Hardening the Orchestration Surface

KubeFence vs. RBAC

- KubeFence achieved an average 35% reduction in cluster attack surface compared to RBAC
- Mitigated 8 CVE exploits and 7 misconfigurations, while RBAC prevente none.
- Introduced only ~21% overhead (~50 ms). Overhead applies only to cluster management operations. No impact on workload performance.

Contribution 4

GoSurf: Identifying Software Supply Chain Attack Vectors

How can attackers hide the execution of malicious code in Go dependencies?

Automatic Behavior	**Unexpected Behaviors**	**Dynamic Behaviors**
• Static Code Generators • Testing Functions	• Global Variable Initialization • Initialization Hooks • Constructors Methods	• Reflection • Interfaces • Unsafe Pointers • CGO Functions • Assembly Functions • Dynamic Plugins • External Execution
Pre-Build Phase	*Initialization Phase*	*Execution Phase*

We created a taxonomy of 12 attack vectors across three stages of the Go package lifecycle

Contribution 4

GoSurf: Identifying Software Supply Chain Attack Vectors

How can developers prioritize code reviews of imported packages and/or understand if packages added vectors across new versions?

16 usecase/gosurf-analysis-summary.md

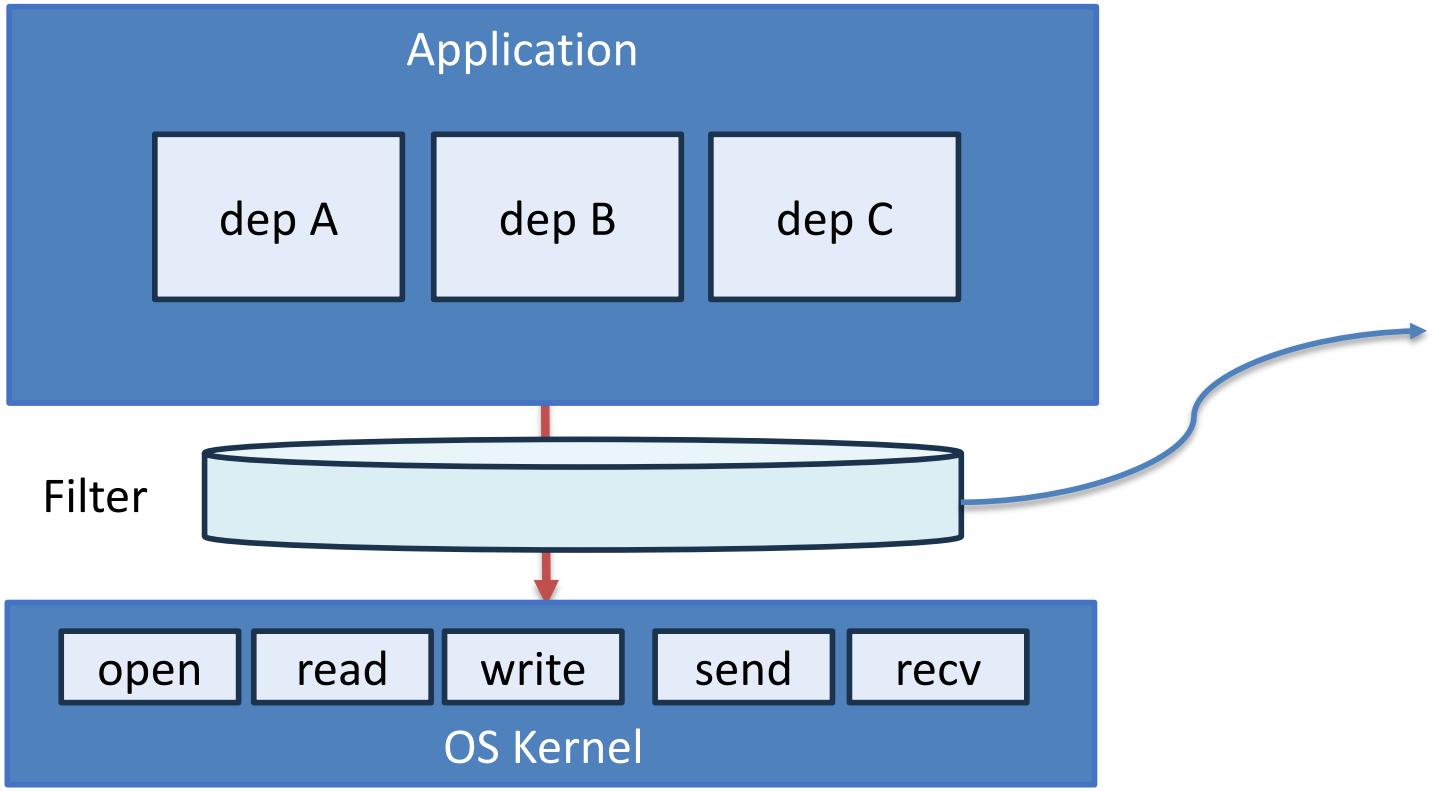
```
@@ -1,13 +1,13 @@
-  Attack Surface Analysis: geth-v1.14.5
+  Attack Surface Analysis: geth-v1.14.6
-  [P1] Static Code Generation:          34
-  [P2] Testing Functions:             2132
-  [I1] Global Variable Initialization: 1116
-  [I2] init() Functions:                76
-  [E1] Constructor Methods:           6470
-  [E2] Reflection:                     128
-  [E3] Interfaces:                    6343
+  [P1] Static Code Generation:          35
+  [P2] Testing Functions:             2134
+  [I1] Global Variable Initialization: 1133
+  [I2] init() Functions:                72
+  [E1] Constructor Methods:           6468
+  [E2] Reflection:                     132
+  [E3] Interfaces:                    6377
   [E4] Unsafe Pointers:                 26
   [E5] CGO Functions:                   21
   [E6] Assembly Functions:              12
```

GoSurf makes actionable the taxonomy finding occurrences of the 12 attack vectors in the code through AST analysis

Contribution 5

GoLeash: Mitigating Software Supply Chain Attacks at Runtime

Application-level sandboxing fails to restrict capabilities of untrusted open source packages that are built within our application

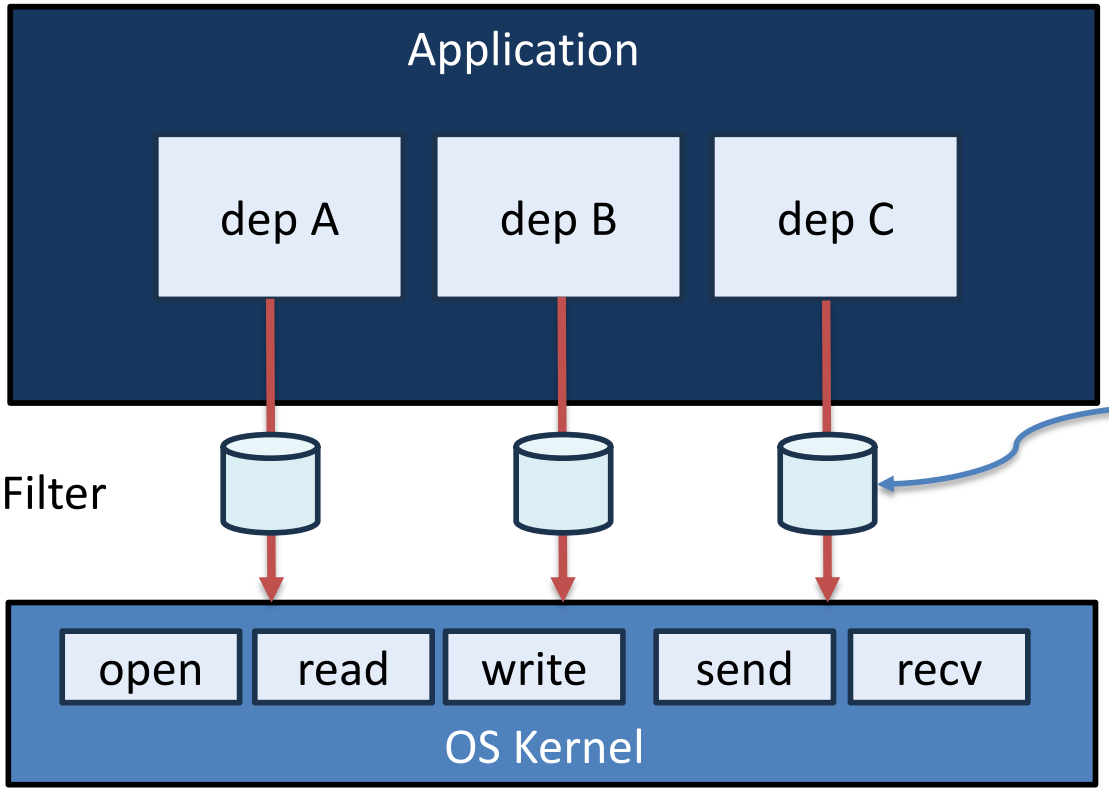


- Seccomp
- AppArmor
- SELinux
- Google gVisor

Contribution 5

GoLeash: Mitigating Software Supply Chain Attacks at Runtime

GoLeash applies package-level enforcement for allowed capabilities at runtime, mitigating supply chain attacks




```
module github.com/mymodule

go 1.20

require (
  github.com/depA v1.1 (network)
  github.com/depB v1.1 (file)
  github.com/depC v1.1 (osStdout)
)
```

Contribution 5

GoLeash: Mitigating Software Supply Chain Attacks at Runtime

Evaluated on 5 real-world Go projects (Kubernetes, etcd, CoreDNS, frp, geth)

- Detected 98% of 3,365 injected supply chain attacks vs 32% with baseline process-level enforcement
- Robust to obfuscation, maintained 92-100% detection across 5,224 obfuscated attacks
- Outperformed Capslock (static analysis) by up to +70 pp detection rate
- Average runtime overhead: **~9%** (4–25%)