Breaking the Bulkhead: Demystifying Cross-Namespace Reference Vulnerabilities in Kubernetes Operators

Andong Chen, Ziyi Guo, Zhaoxuan Jin, Zhenyuan Li, Yan Chen

Zhejiang University, China Northwestern University, USA

Kubernetes is Everywhere

- Kubernetes has become the default application infrastructure in modern cloud-native environments.
- Over 82% container users are adopting Kubernetes.[1]
- Almost all major clouds are providing Kubernetes services.
- Reports estimate the market size to be USD 3.13B in 2026.[2]

Google Kubernetes Engine

Azure Kubernetes Service (AKS)

Amazon EKS

[1] CNCF Annual Cloud Native Survey - The infrastructure of AI's future
[2] Kubernetes Market Size & Share Analysis - Growth Trends and Forecast (2026 - 2031) Source: https://www.mordorintelligence.com/industry-reports/kubernetes-market

…But it can be insecure

We Unveiled a New Privilege Escalation Class in Kubernetes

Cross-Namespace Reference Vulnerability

And It Has Been Hiding in Operators for Years:

- First Operator-specific vulnerability study
- 14%+ Operators affected
- Confirmations and CVEs from major vendors:
  - Google – The inventor of Kubernetes
  - Red Hat – The inventor of Operator
  - NVIDIA, Grafana, …
- Practical detector and mitigation delivered

Isolation in Kubernetes

- A single Kubernetes cluster often hosts:
  - Multiple applications
  - Multiple teams
  - Multiple users
  - Multiple tenants
- **Namespace & Role-Based Access Control (RBAC)** isolate users and applications

Isolation by Namespace & RBAC

Namespaces and RBAC form the *foundation* of isolation:

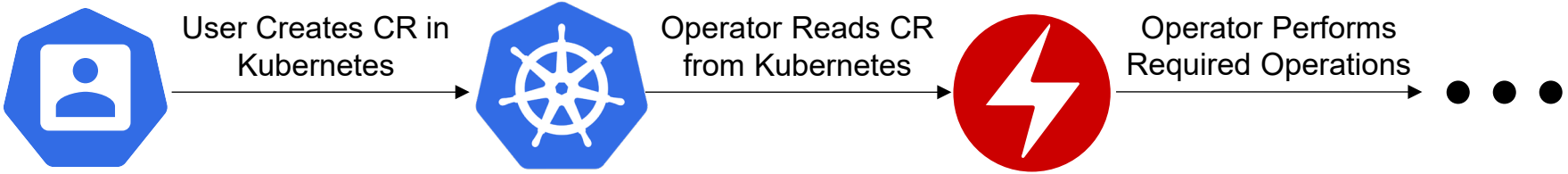
- **Namespaces** serve as the primary logical isolation boundary in Kubernetes.
  - Resources such as Pods, Services, and Secrets are scoped to namespaces.
  - Resources in different namespaces are independent and do not interfere with each other.
- **Role-Based Access Control (RBAC)** limits user access.
  - Namespace-level roles can restrict users to their **AUTHORIZED** namespaces.
  - Cluster-level roles allow users to access **ALL** namespaces

Kubernetes Operator

- Use Kubernetes can be a tough work. 😨
  - **Manually** application lifecycle management like scaling, upgrades……
  - **Manually** deal with hundreds lines of YAML manifests.
  - **Manually** handle multiple kinds of Kubernetes resources.
- Kubernetes Operator automates the lifecycle management of complex applications. 🤗
  - Encapsulate common operations.
  - Users interact with Operators by simpler interfaces.
  - Let Operator do all remaining dirty manual tasks.

Understand Kubernetes Operator

- An Operator consists of:
  - **Custom Resource Definitions (CRD),** which define new types of **Custom Resource (CR)**, essentially input formats of the Operator
  - **Controllers**, which continuously watch Custom Resources and perform operations as declared in Custom Resources
- To use an Operator:
  - Users interact with Operators by manipulating Custom Resources in Kubernetes, declaring their purpose
  - The Operator reads Custom Resources and performs required operations



Understand Kubernetes Operator

- An Operator consists of:
  - **Custom Resource Definitions (CRDs),** which define new types of **Custom Resource (CR)**, essentially the input format of the Operator
  - **Controllers**, which continuously watch Custom Resources and perform operations as declared in Custom Resources
- Each resource has an explicit scope:
  - **Namespace-scoped resource** exists within a **SINGLE** namespace
    - **Regular users** are allowed to manipulate such resources **in their assigned namespaces**.
  - **Cluster-scoped resource** exists at the cluster level, applies to **ALL** namespaces
    - Only users (typically **cluster administrators**) with cluster-level permissions may manipulate such resources.

What Goes Wrong?

- Operators can accept namespace-scoped resources as user input.
  - Therefore, **regular user** restricted in their own namespaces may create such resources and invoke Operators
- However, upon user requests, Operator controller logic may:
  - Reference, access or modify resources in **OTHER** namespaces
  - Reference, access or modify cluster-scoped resources, which may affect **ALL** namespaces
- **PRIVILEGE ESCALATION: An attacker may leverage such logic to access or modify others' resources!**

Threat Model

- Adversary – Any attacker gains namespace-level initial access:
  - A malicious tenant in a multi-tenant cluster who is only authorized to access their assigned namespace.
  - A compromised application running in a namespace with namespace-level permissions mounted.
  - An attacker who steals credentials of Kubernetes accounts with namespace-level permissions.
- Assumption:
  - The attacker has legitimate access to one namespace.
  - **NO** container escape, shell access or direct control of vulnerable applications is assumed.
- Goal: Break namespace isolation. Perform operations in unauthorized namespace

Threat Model

- Adversary – Any attacker gains namespace-level initial access:
  - A malicious tenant in a multi-tenant cluster who is only authorized to access their assigned namespace.
    - Major cloud platforms like GKE, AKS, EKS provide Multi-Tenant Kubernetes Support
    - Many cloud services are operated based on multi-tenant Kubernetes (e.g. Red Hat Developer Sandbox)
    - **Attackers may subscribe such cloud services and affect other tenants!**
  - A compromised application running in a namespace with namespace-level
- Assumption:

Multi-tenant Kubernetes clusters are common in practice.

Threat Model

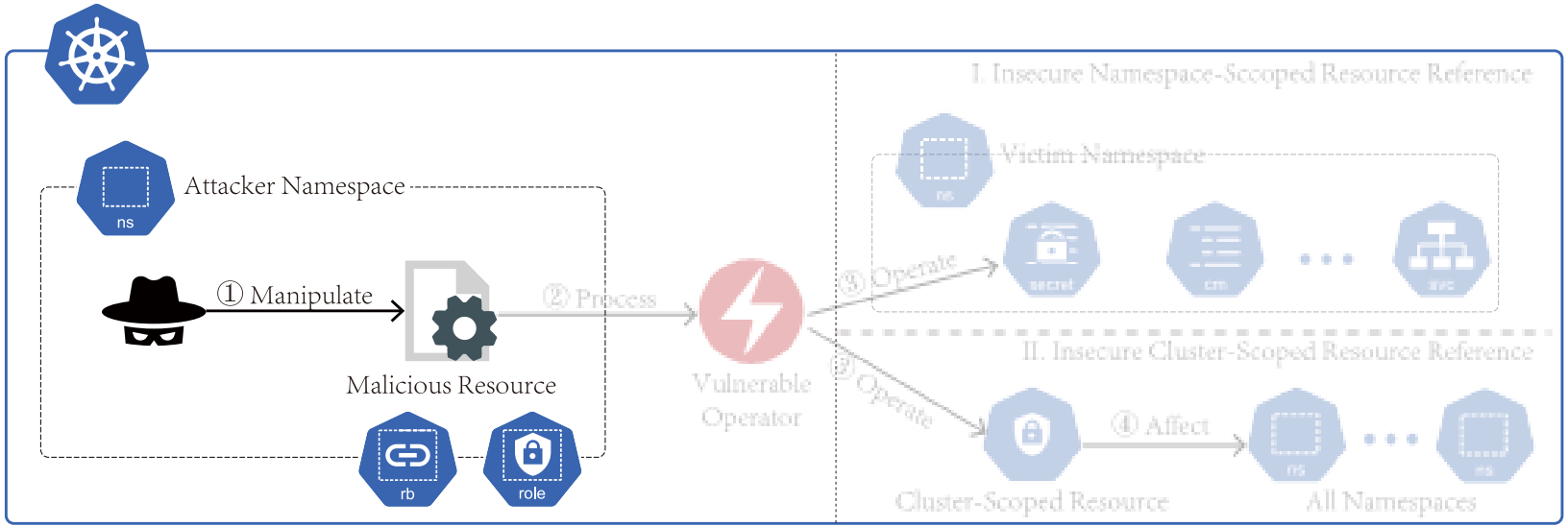
- Adversary – Any attacker gains namespace-level initial access:
- Assumption:
  - The attacker has legitimate access to one namespace.
  - **NO** container escape, shell access or direct control of vulnerable applications is assumed:
    - Existing works[1-3] typically assume somehow full control of vulnerable applications, which can be a strong assumption

More Reasonable Assumption

[1] N. Yang, W. Shen, J. Li, X. Liu, X. Guo, and J. Ma, “Take over the whole cluster: Attacking Kubernetes via excessive permissions of third-party applications,” in Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security
[2] Y. Gu, X. Tan, Y. Zhang, S. Gao, and M. Yang, “ EPScan: Automated Detection of Excessive RBAC Permissions in Kubernetes Applications ,” in 2025 IEEE Symposium on Security and Privacy (SP).
[3] Y. Avrahami and S. B. Hai, “Kubernetes privilege escalation: Container escape== cluster admin,” Black Hat USA, 2022

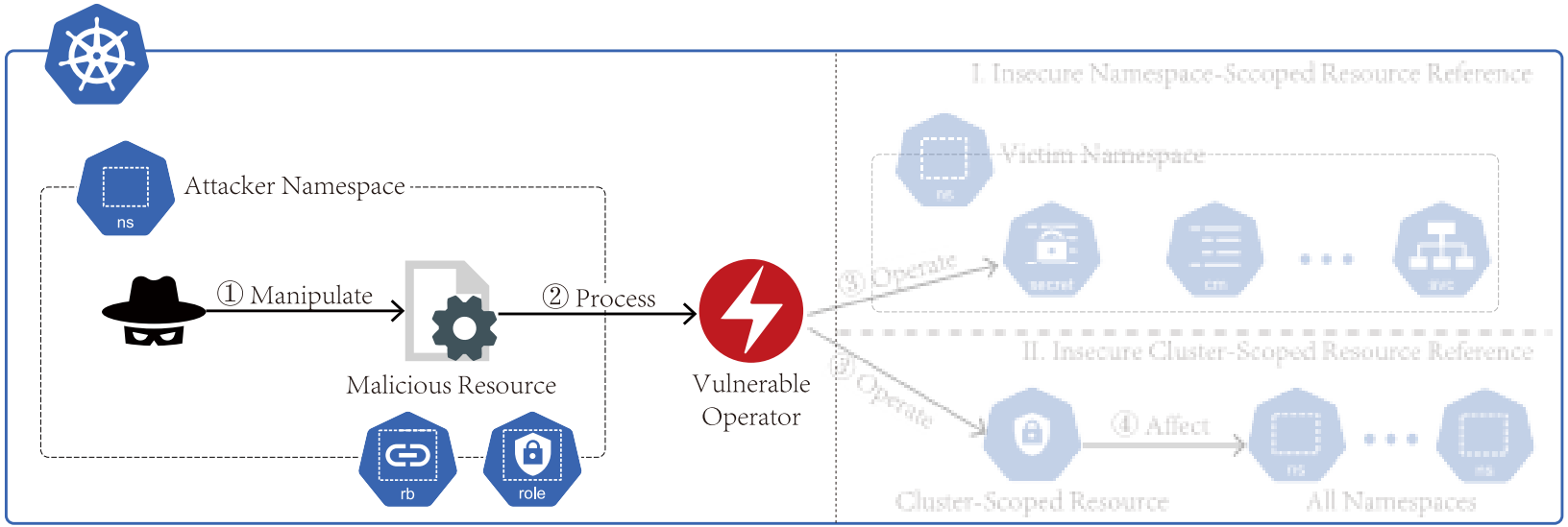
Cross-Namespace Reference Vulnerability

- Attack Flow
  - ① The attacker creates a crafted namespace-scoped resource in his authorized namespaces.
  - ② The Operator observes and processes the resource.
  - ③ The Operator performs actions that affect other namespaces.



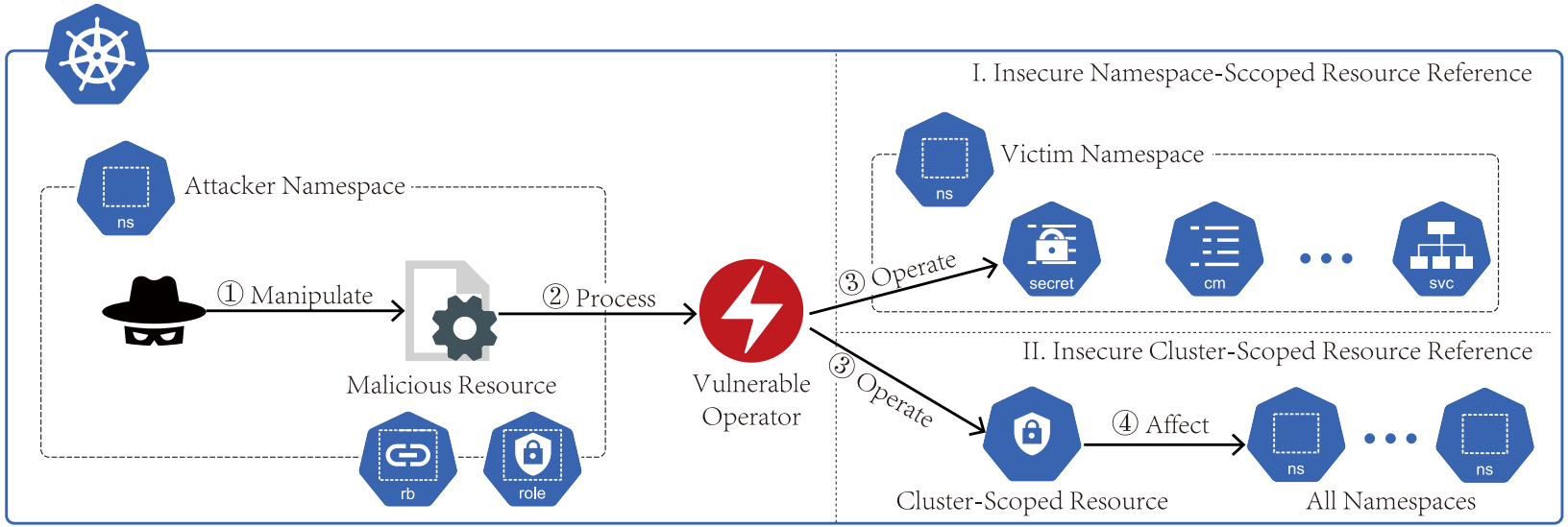
Cross-Namespace Reference Vulnerability

- Attack Flow
  - ① The attacker creates a crafted namespace-scoped resource in his authorized namespaces.
  - ② The Operator observes and processes the resource.
  - ③ The Operator performs actions that affect other namespaces.



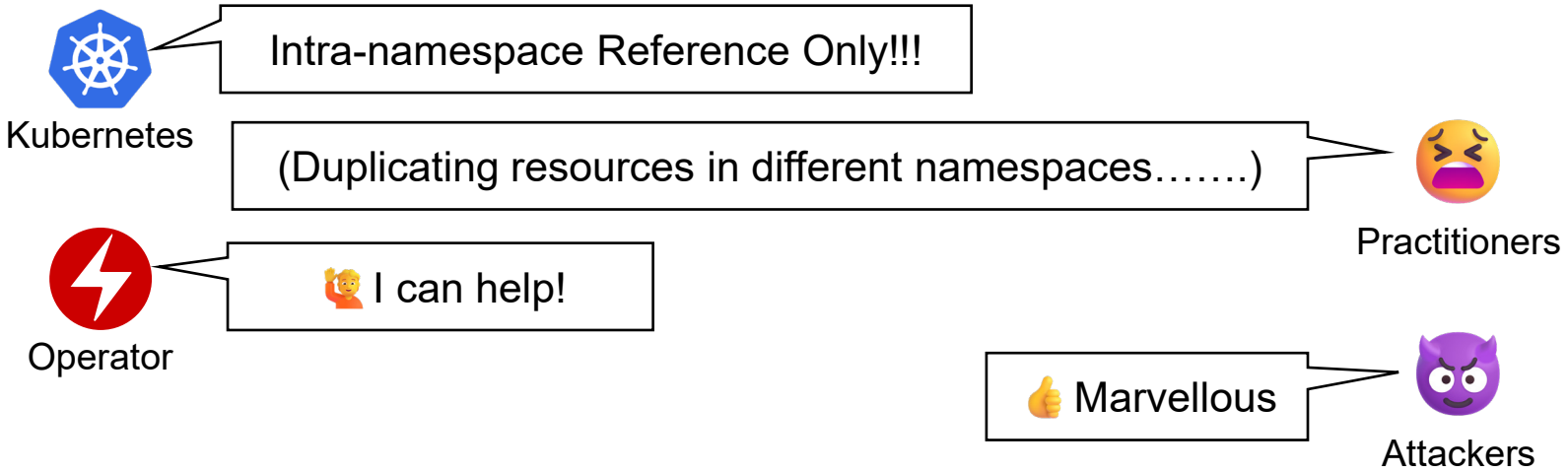
Cross-Namespace Reference Vulnerability

- Attack Flow
  - ① The attacker creates a crafted namespace-scoped resource in his authorized namespaces.
  - ② The Operator observes and processes the resource.
  - ③ The Operator performs actions that affect other namespaces.



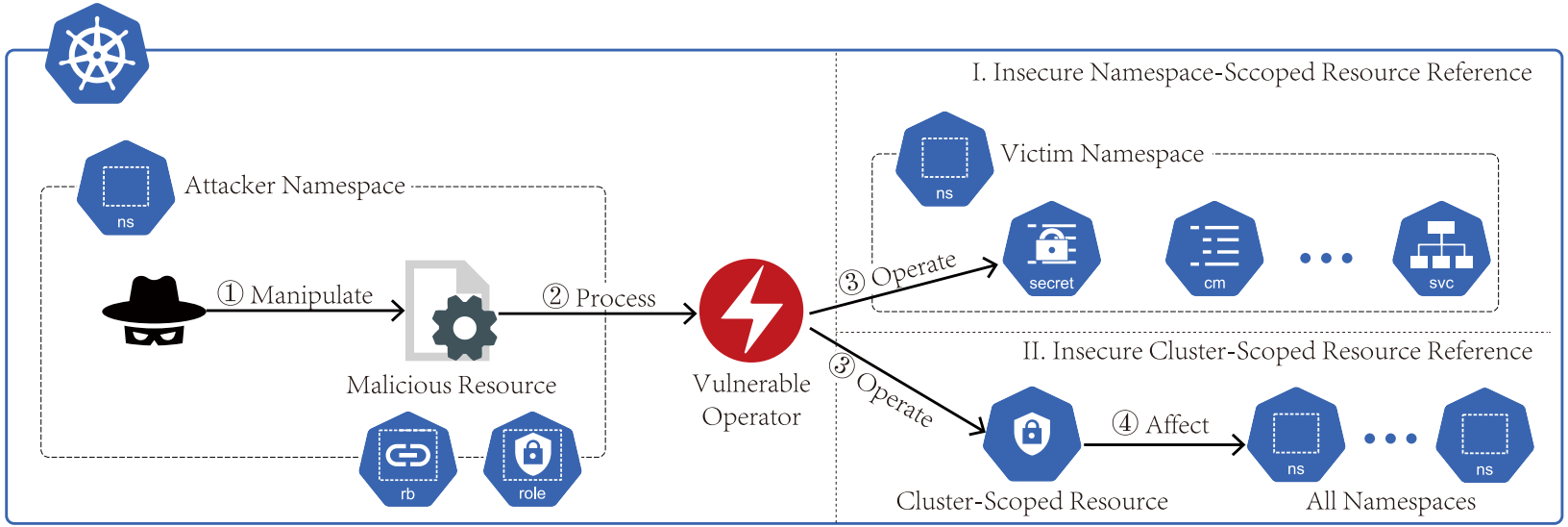
Cross-Namespace Reference Vulnerability

- Root Cause:
  - Mismatch between **resource scope** and **operation scope**:
    - Operators accept namespace-scoped resources. Regular tenant can invoke
    - The process of the resource may affect other namespaces
  - Insecure trade-off between **convenience** and **security**:



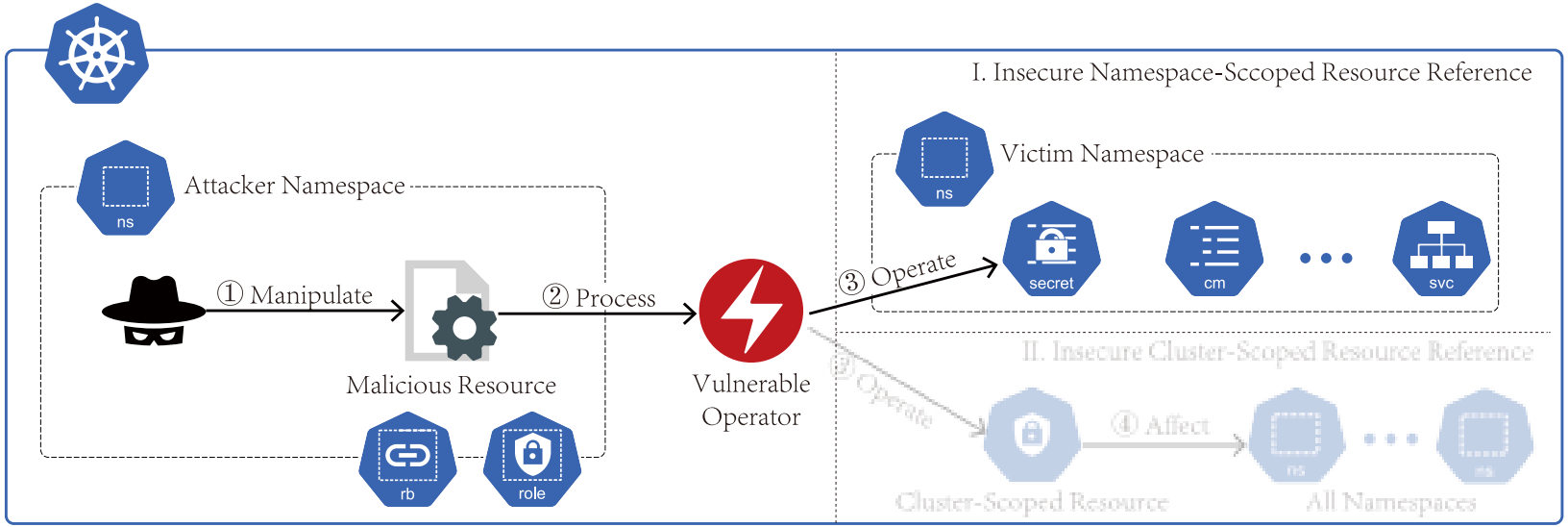
Cross-Namespace Reference Vulnerability

- Tactics:
  1. **Insecure Namespace-Scoped Resource Reference**
     - Attacker input controls references to resources in other namespaces.
  2. **Insecure Cluster-Scoped Resource Reference**
     - Attacker input references cluster-scoped resource, affecting all namespaces



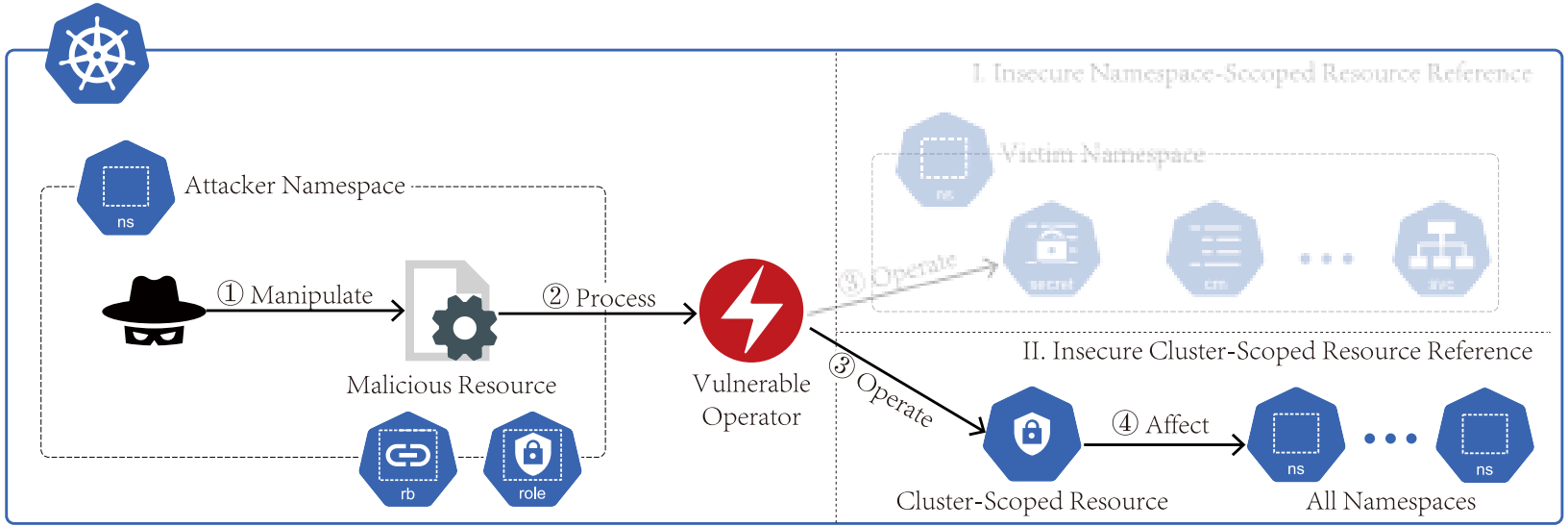
Cross-Namespace Reference Vulnerability

- Tactics:
  1. **Insecure Namespace-Scoped Resource Reference**
     - Attacker input controls references to resources in other namespaces.
  2. **Insecure Cluster-Scoped Resource Reference**
     - Attacker input references cluster-scoped resource, affecting all namespaces



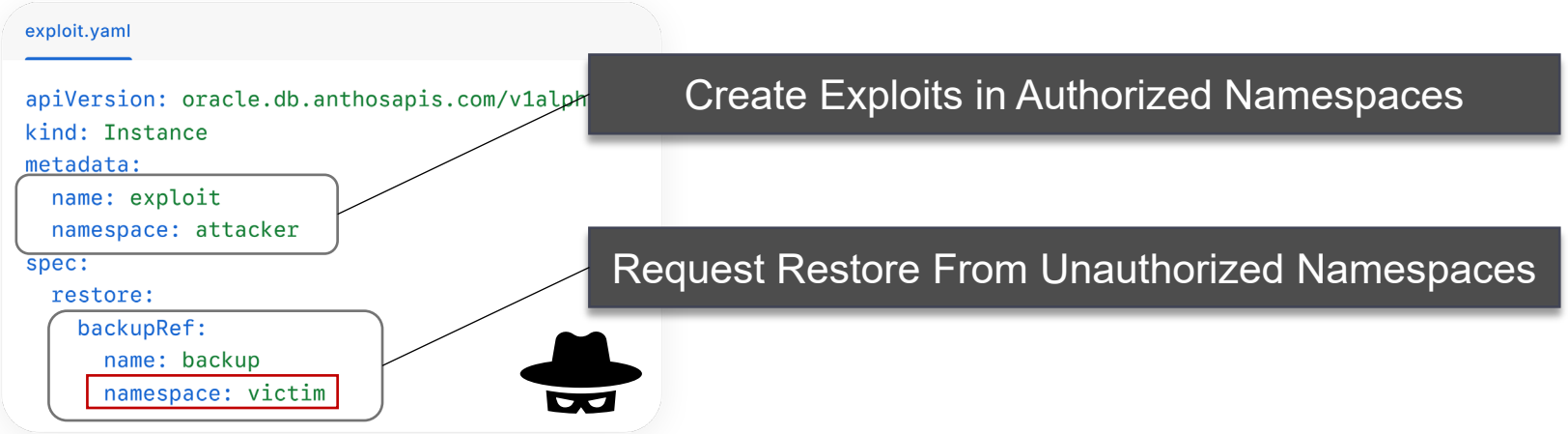
Cross-Namespace Reference Vulnerability

- Tactics:
  1. **Insecure Namespace-Scoped Resource Reference**
     - Attacker input controls references to resources in other namespaces.
  2. **Insecure Cluster-Scoped Resource Reference**
     - Attacker input references cluster-scoped resource, affecting all namespaces



Insecure Namespace-Scoped Resource Reference

- Example: GoogleCloudPlatform/elcarro-oracle-operator
  - An Operator for Oracle Database Lifecycle Management Automation
  - Automate database backup and restore
  - **ALLOW RESTORING FROM BACKUPS IN OTHER NAMESPACE.**
    - **Leak databases of other tenants!**
    - **Further exploitation can lead to data tampering and data loss!**



Insecure Cluster-Scoped Resource Reference

- Example: Grafana/tempo-operator
  - Automate deployment and configuration for Grafana Tempo
  - Reference *ClusterRoleBinding* upon Namespace-scoped Resources
    - *ClusterRoleBinding* is a cluster-scoped Kubernetes resource, which assigns **CLUSTER-LEVEL** permission to a Kubernetes entity.
    - **An attacker can invoke the Operator and get CLUSTER-LEVEL PERMISSION!**

exploit.yaml

```yaml
apiVersion: tempo.grafana.com/v1alpha1
kind: TempoStack
metadata:
  name: exploit
  namespace: attacker
spec:
  template:
    queryFrontend:
      jaegerQuery:
        enabled: true
        monitorTab:
          enabled: true
          prometheusEndpoint: "..."
```

Condition for ClusterRoleBinding Creation

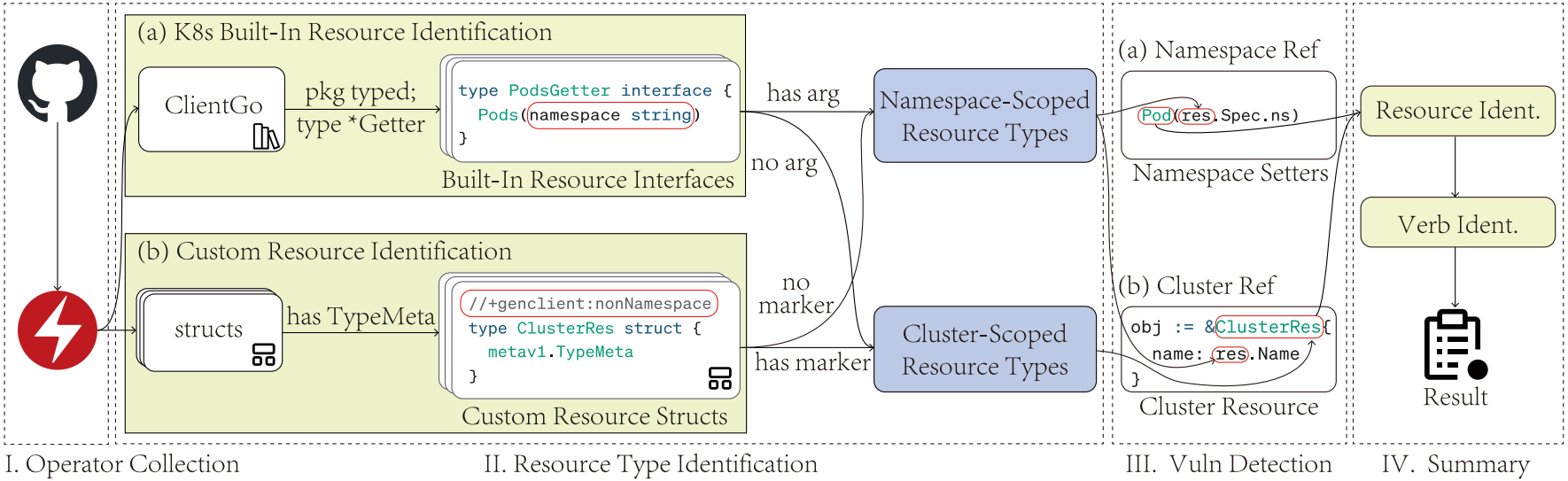
query_frontend.go

```
  if tempo.Spec.Template.QueryFrontend.JaegerQuery.Enabled &&
tempo.Spec.Template.QueryFrontend.JaegerQuery.MonitorTab.Enabled &&
    tempo.Spec.Template.QueryFrontend.JaegerQuery.MonitorTab.PrometheusEndpoint ==
thanosQuerierOpenShiftMonitoring {
    clusterRoleBinding := openShiftMonitoringClusterRoleBinding(tempo, d)
    manifests = append(manifests, &clusterRoleBinding)
  }
```

Operator Creates ClusterRoleBinding

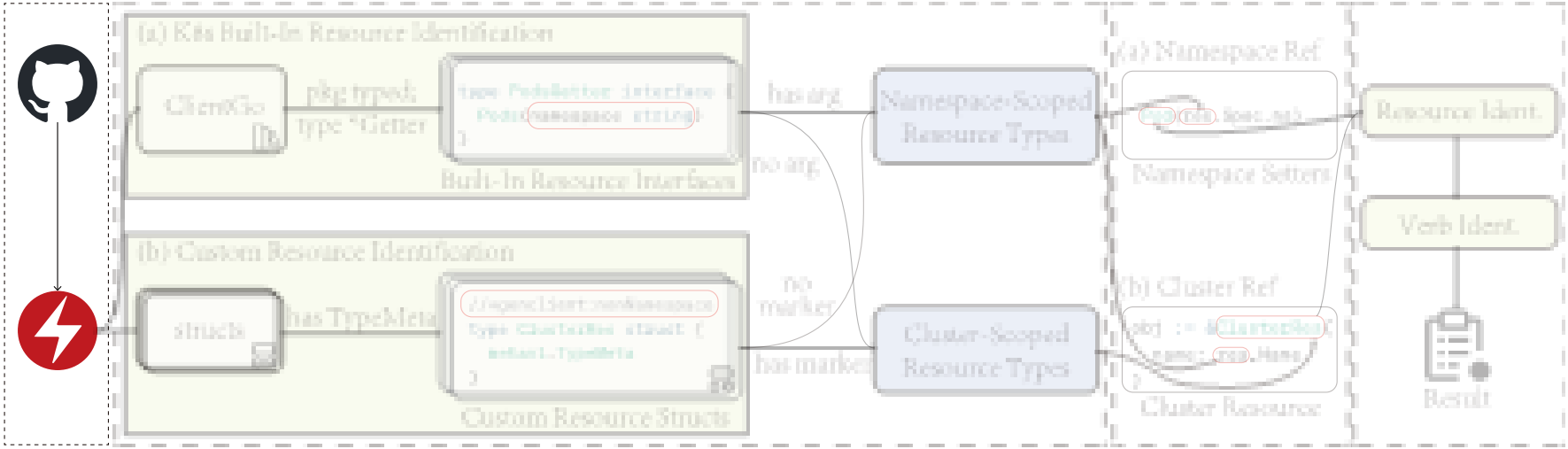
Measurement

- To understand how widespread the vulnerabilities are in real-world Kubernetes Operators, we perform a systematic measurement.
- CodeQL used for static analysis



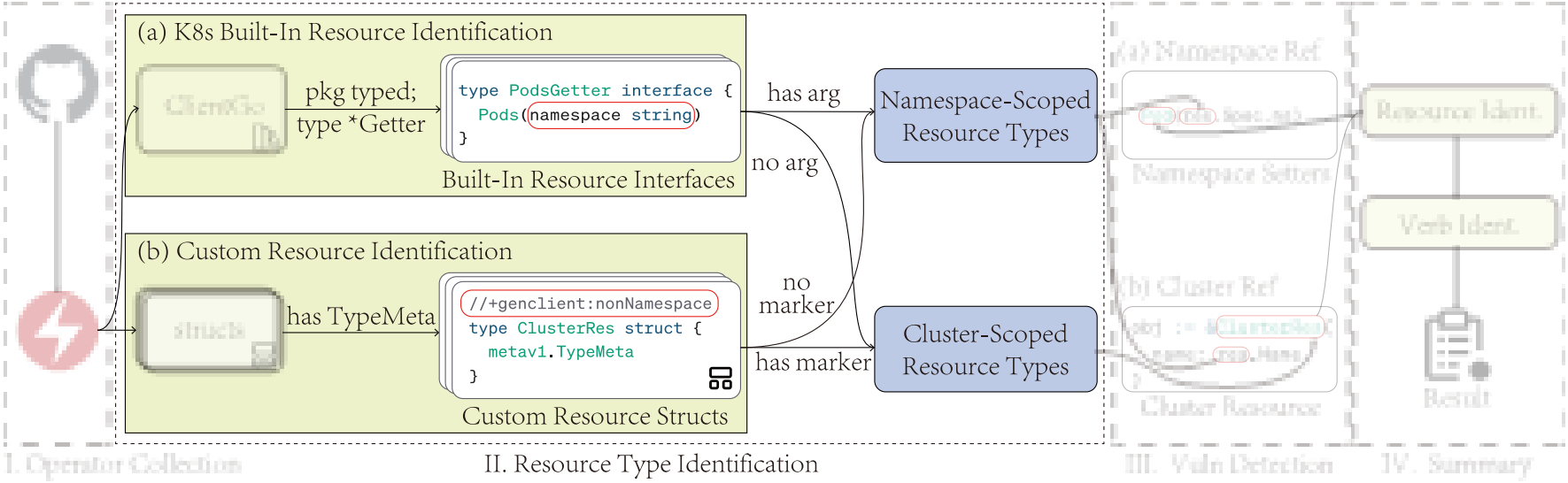
Measurement - Operator Collection

- Focused on Golang Operators due to ecosystem dominance
- Crawled and analyzed 2,268 real-world Kubernetes Operators
  - By GitHub searching *'Kubernetes operator language:go'*



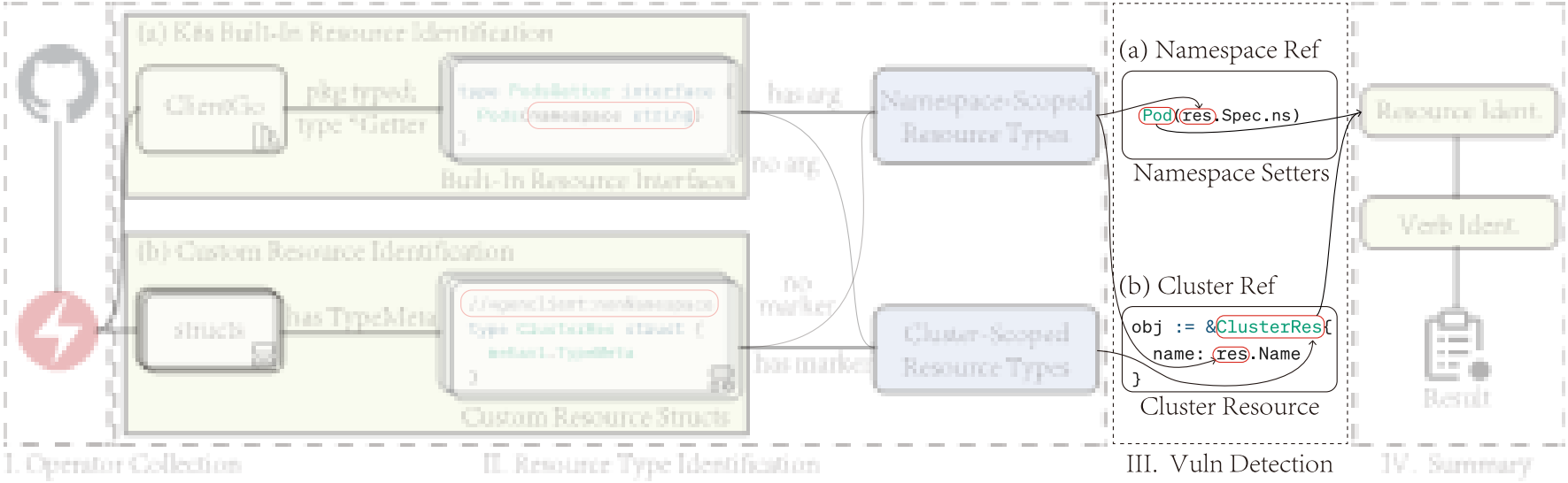
Measurement - Resource Type Identification

- Extract all involved Resources and their declared scopes (Namespace-scoped vs Cluster-scoped) based on specifications of major Operator-related libraries.



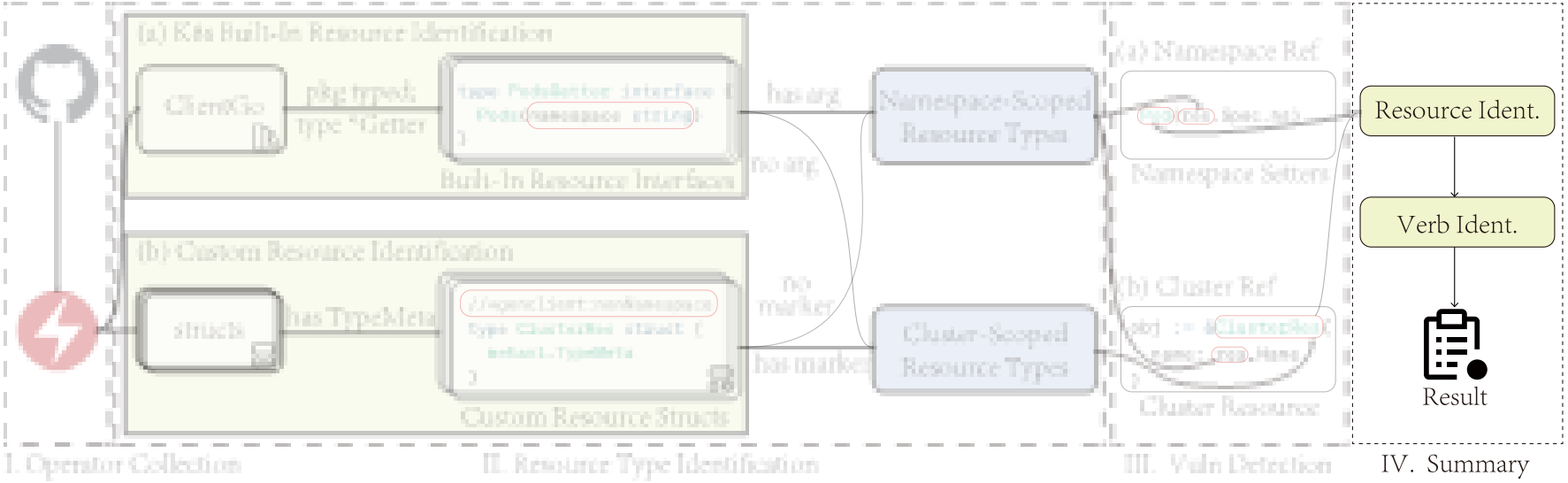
Measurement – Vuln Detection

- Perform inter-procedural taint tracking from namespace-scoped resource fields to:
  - namespace setters of the other namespace-scoped resources
  - any cluster-scoped resources



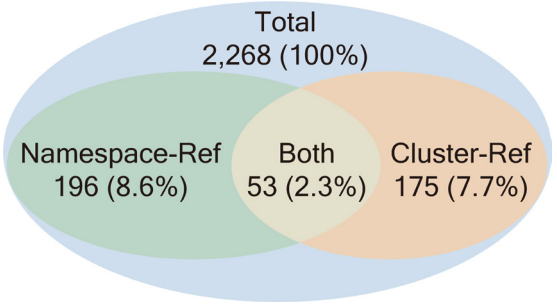
Measurement - Summary

- Aggregate result by tactic type (namespace or cluster ref).
- Analyze affected resource types and API verbs (Get/Create/Update/Delete/...) via another inter-procedural taint tracking.



Measurement – Key Result

- **RQ1:** How many operators are potentially vulnerable to insecure cross-namespace reference?
  - 318 (14%) Potentially Vulnerable Operators
    - 196 (8.6%) involve namespace-scoped resource references
    - 175 (7.7%) involve cluster-scoped resource references
    - 53 (2.3%) involve both



- **RQ2:** What resources can be cross-namespace referenced by attackers?

Most Referenced Namespace-scoped Resources		
Resource	**# Op.**	**Description**
Secret	102	Sensitive Data Storage
ConfigMap	29	Application Configuration
Deployment	29	Workload Management

Most Referenced Cluster-scoped Resources		
Resource	**# Op.**	**Description**
Namespace	62	Logical Boundary
ClusterRole	40	Cluster-Level Permission Manage.
ClusterRoleBinding	26	

Measurement – Key Result

- **RQ3:** What can attackers do towards cross-namespace referenced resources?

Major Insecure Cross-Namespace Operations			
Scope	**Operation**	**# Op.**	**Impact**
Namespace	Get - Secret	102	Leaking Credential
	Get - ConfigMap	29	Leaking Application Configuration
	Get - Deployment	29	Leaking Running Application Info
Cluster	Get – Namespace	51	Leaking Other Tenant Info
	Create – ClusterRoleBinding	33	Gaining Cluster-Level Permission
	Create - Namespace	29	Creating Unmanaged Namespaces

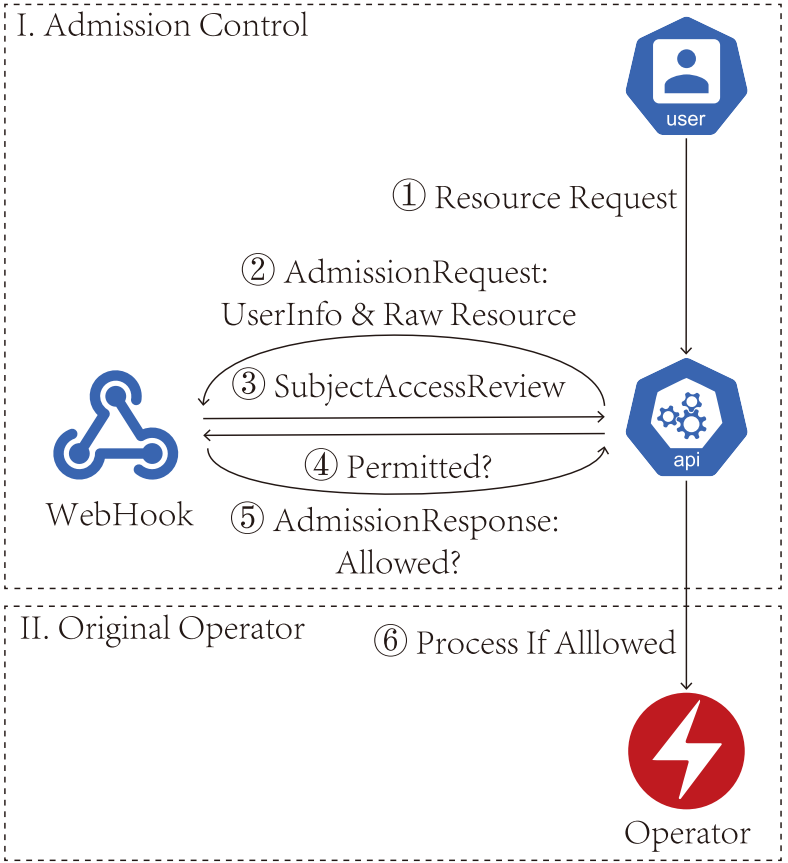
Measurement – Real World Impact

- **RQ4:** How can insecure cross-namespace references impact the real world?
  - 8 Confirmations and 7 CVEs

Confirmed Vulnerabilities		
Operator	**Vendor**	**Status**
elcarro-oracle-operator	Google – *Inventor of Kubernetes*	Confirmed
gateway-operator	Kong - *Leading API Gateway*	CVE Assigning
baremetal-operator	Metal3-io – *Leading Bare Metal Management*	CVE-2025-29781
observability-operator	Red Hat – *Inventor of Operator*	CVE-2025-2843
gateway-operator	Kong	CVE Assigning
tempo-operator	Grafana & Red Hat	CVE-2025-2842
tempo-operator	Grafana & Red Hat	CVE-2025-2786
ais-k8s	NVIDIA	CVE-2025-23260

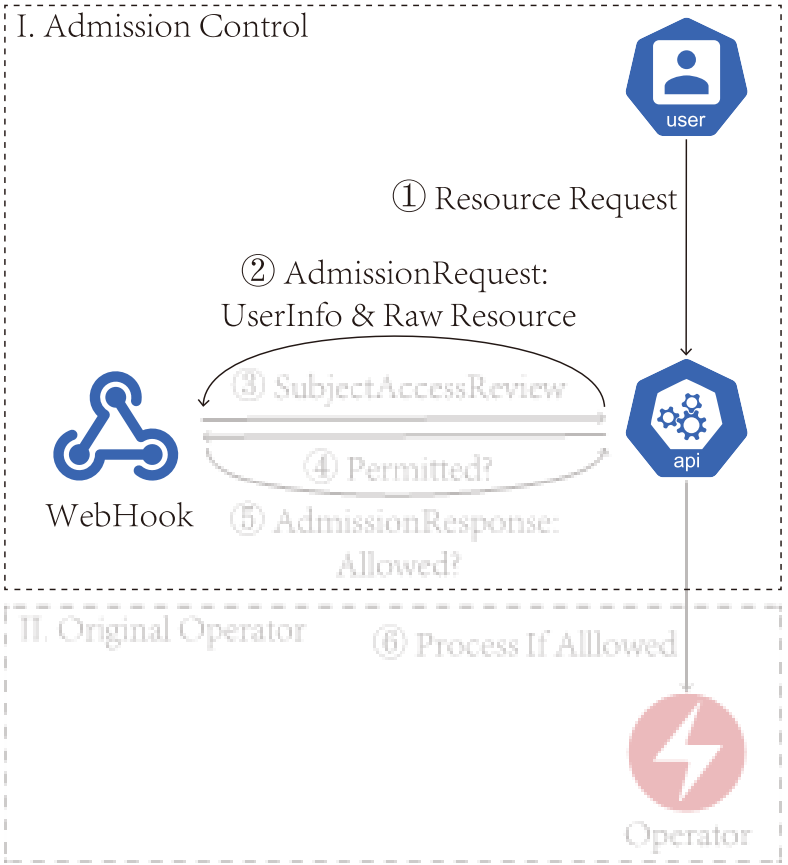
Mitigation

- Admission Control / Webhook
  - Intercepting resource before it enters the cluster and reaches Operator
  - Independent of existing Operator
  - No pain of refactoring code
  - Prototype:
    1. Extract referenced namespaces and resources from the request
    2. Use *SubjectAccessReview* to check: Is the user already authorized?
    3. If not authorized, reject the Resource immediately.



Mitigation

- Admission Control / Webhook
  - Intercepting resource before it enters the cluster and reaches Operator
  - Independent of existing Operator
  - No pain of refactoring code
  - Prototype:
    1. Extract referenced namespaces and resources from the request
    2. Use *SubjectAccessReview* to check: Is the user already authorized?
    3. If not authorized, reject the Resource immediately.

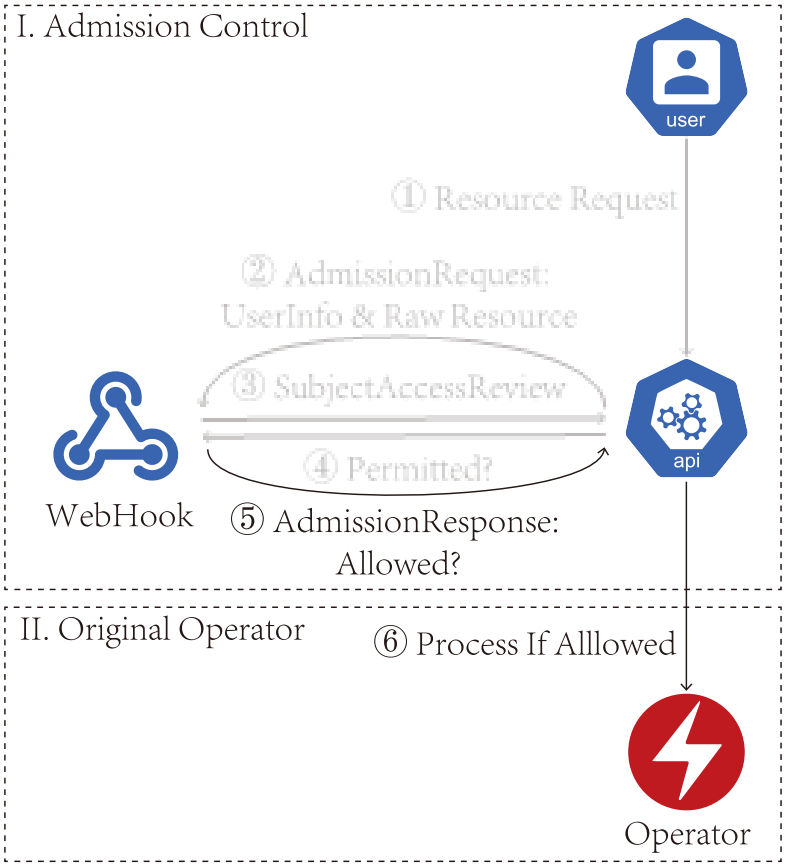


Mitigation

- Admission Control / Webhook
  - Intercepting resource before it enters the cluster and reaches Operator
  - Independent of existing Operator
  - No pain of refactoring code
  - Prototype:
    1. Extract referenced namespaces and resources from the request
    2. Use *SubjectAccessReview* to check: Is the user already authorized?
    3. If not authorized, reject the Resource immediately.

Mitigation

- Admission Control / Webhook
  - Intercepting resource before it enters the cluster and reaches Operator
  - Independent of existing Operator
  - No pain of refactoring code
  - Prototype:
    1. Extract referenced namespaces and resources from the request
    2. Use *SubjectAccessReview* to check: Is the user already authorized?
    3. If not authorized, reject the Resource immediately.



Mitigation

- Mitigate with Kubernetes Admission Control
  - We designed Validating Admisson Webhook for all confirmed vulnerabilities.
  - With scaffold generated by KubeBuilder, only tens of LoC are required
  - Successfully blocked **ALL** malicious requests with ignorable overhead

Validating Admission Webhook Mitigation Evaluation				
Operator	**Vendor**	**Mitigated?**	**Before (ms)**	**After (ms)**
elcarro-oracle-operator	Google	✅	95.7	99.3 (+3.6)
gateway-operator	Kong	✅	94.5	97.0 (+2.5)
baremetal-operator	Metal3-io	✅	91.6	94.5 (+2.9)
observability-operator	Red Hat	✅	96.5	101.2 (+4.7)
tempo-operator	Grafana & Red Hat	✅	106.3	113.7 (+7.4)
ais-k8s	NVIDIA	✅	101.7	105.2 (+3.5)
Average			97.7	101.8 (+4.1)

Conclusion

- We unveiled a hidden attack surface - Operators.
  - Operators are typically highly customized, privileged, and interactive.
- We presented the first systematic study on Operator vulnerability.
  - Cross-Namespace Reference Vulnerability with two tactics
- We conducted a large scale measurement of vulnerabilities.
  - Affect >14% Operators in the wild.
- We responsibly disclosed vulnerabilities to vendors.
  - Acknowledgements from industry leaders, including Google, Red Hat and NVIDIA.
- Tools & Mitigations are open-sourced to the community.
  - Paper: https://dx.doi.org/10.14722/ndss.2026.240761
  - Artifacts: https://doi.org/10.17605/OSF.IO/PWVC4

Thanks

Paper: https://dx.doi.org/10.14722/ndss.2026.240761
Artifacts: https://doi.org/10.17605/OSF.IO/PWVC4

Threat Model

- Adversary – Any attacker gains namespace-level initial access:
  - A malicious tenant in a multi-tenant cluster who is only authorized to access their assigned namespace.
  - A compromised application running in a namespace with namespace-level permissions mounted.
  - An attacker who steals credentials of Kubernetes accounts with namespace-level permissions.
- Ass
  - T
  - applications is assumed.
- Goal: Break namespace isolation. Perform operations in unauthorized namespace

Lateral Movement between Isolated Namespaces

Discussion & Mitigation

- Carefully Using Kubernetes
  - Share cluster with trusted users
  - Take care of applications running on Kubernetes, as they may be exploited to gain initial Kubernetes access and affect other applications / tenants.
- Scope Alignment
  - Ensure the scope of resources reflects the scope of their operational effect.

Discussion & Mitigation

- Principle of Least Privileges (PoLP)? 🤔
  - Not a perfect choice
  - Those privileges are not over-granted
    - They are actually used by Operators.
  - Operators need those privileges to execute operation tasks
    - Simply remove related privileges will interrupt Operators