BASTION: A Security Enforcement Network Stack for Container Networks

Jaehyun Nam[1], Seungsoo Lee[1], Hyunmin Seo[1],
Phillip Porras[2], Vinod Yegneswaran[2], and Seungwon Shin[1]
KAIST[1] and SRI International[2]

The state of Container Security

- Containers
  - Abstraction at the application layer Include everything needed to run an application
  - Sharing the same host OS kernel → faster and lighter than virtual machines
  - Microservices → the major use of containerization technologies

- The state of container security
  - Tripwire's Sate of Container Security Report (2018) [1]
    - Increasing number of container security incidents
      - 60% of organizations → security incidents
    - Vulnerable container images
      - 47% of containers they deployed → vulnerable
    - Limited visibility into container security
      - 52% of organizations → difficulty in investigating container security issues

[1] https://www.tripwire.com/state-of-security/devops/organizations-container-security-incident

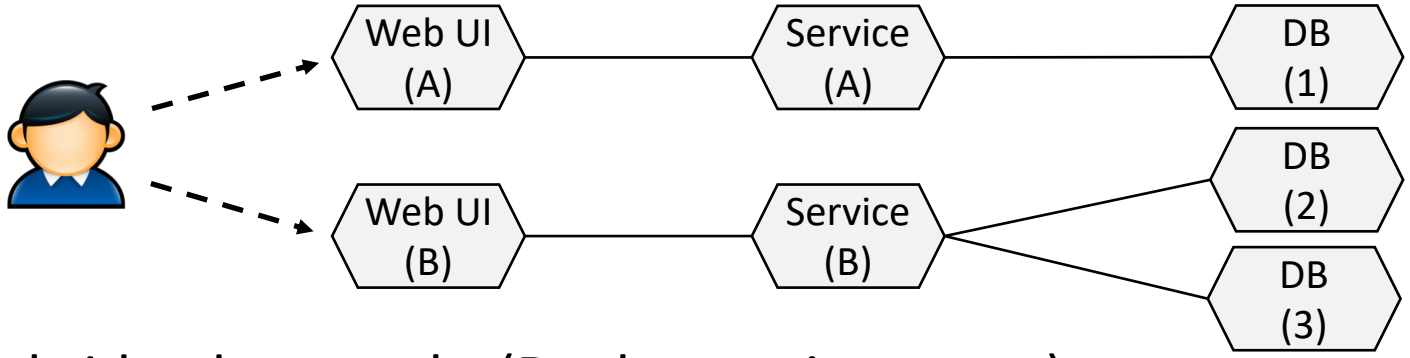
Current Container Security Solutions

- Container Image Integrity
  - Vulnerability scanning nested in container images using CVE databases
- Application Isolation
  - Container isolation → Namespaces, cgroups, capabilities
  - Further restrictions on system resources → AppArmor, Seccomp, SELinux
- Run-time threat detection
  - Monitoring container behaviors, and detecting run-time policy violations

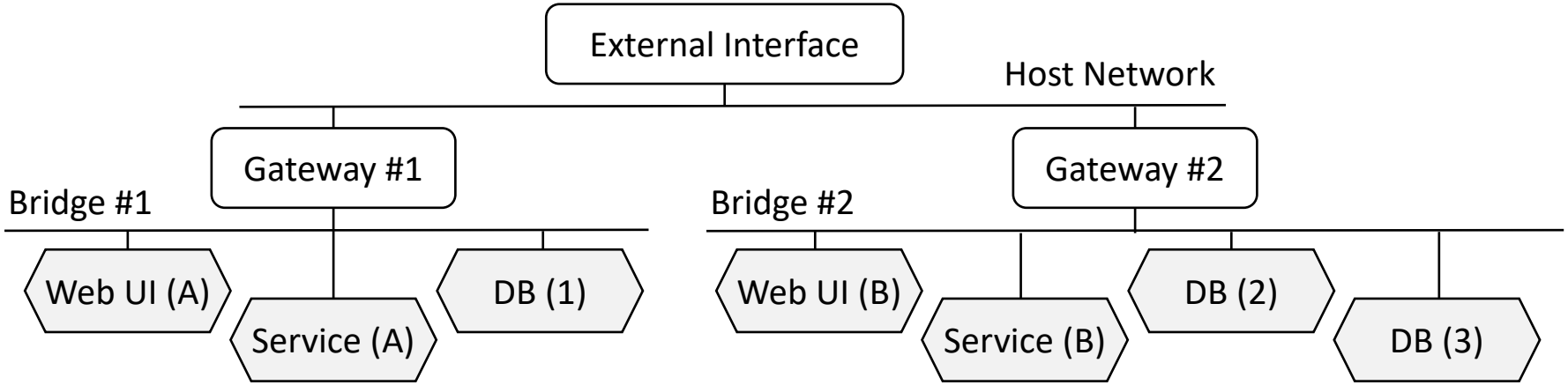
Mostly focus on the security of containers!
Less concern for the security of container networks!

Current Container Networks (1/2)

- Conceptual microservice architecture

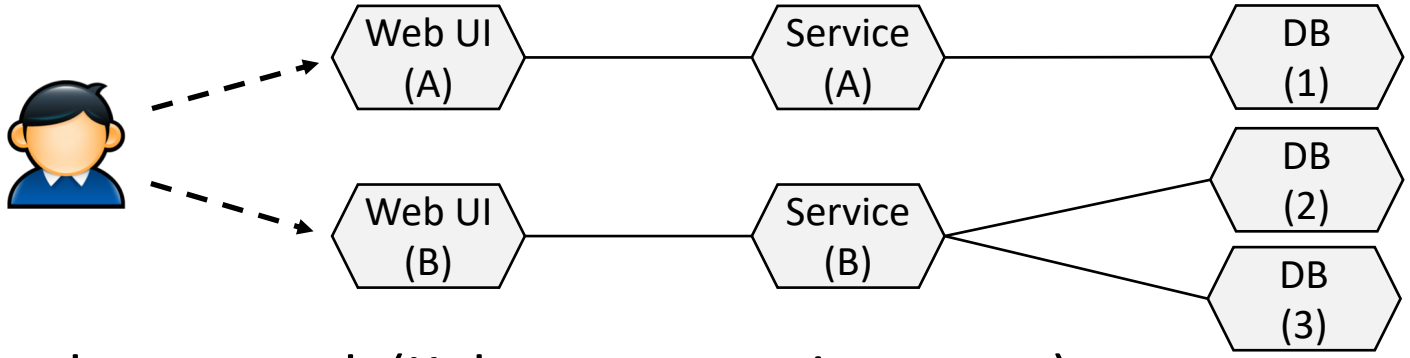


- Separate bridged networks (Docker environment)

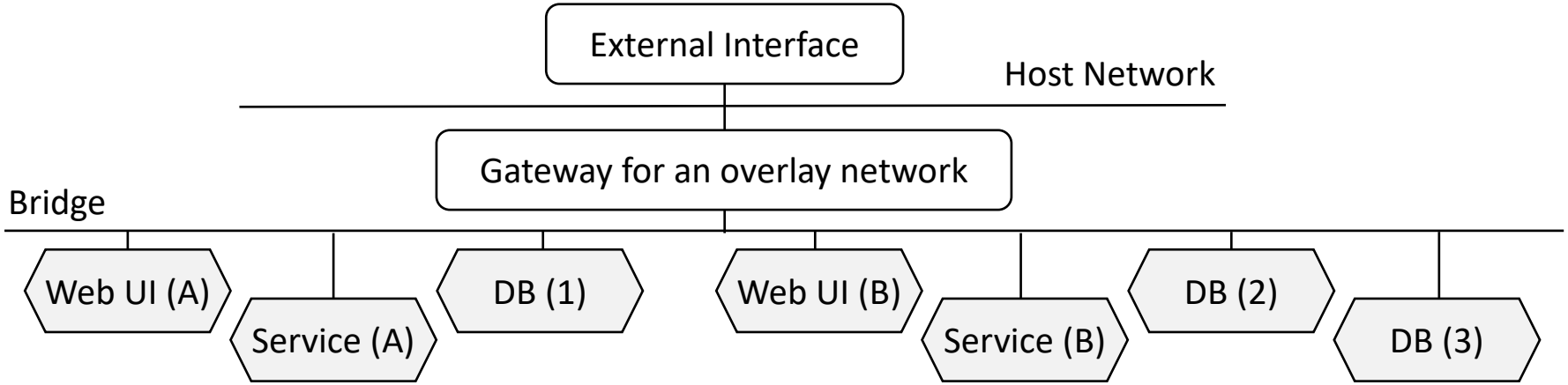


Current Container Networks (2/2)

- Conceptual microservice architecture

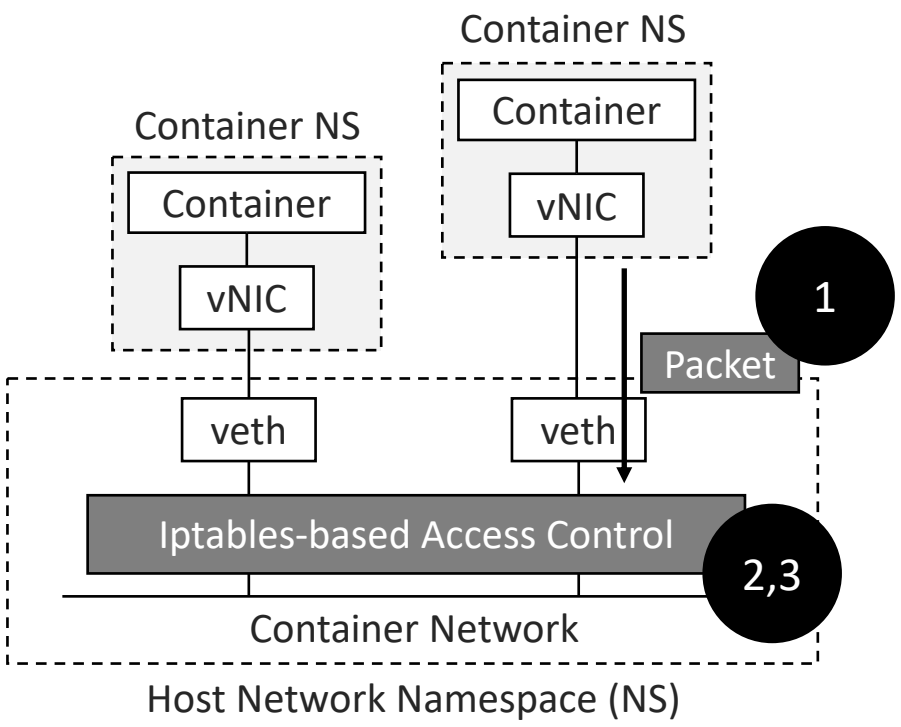


- Single overlay network (Kubernetes environment)



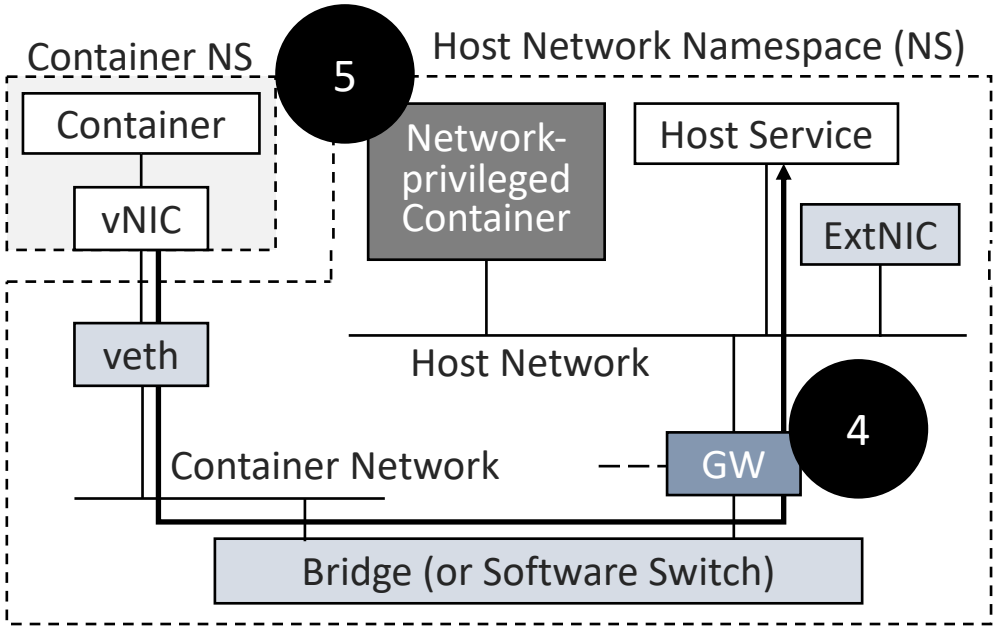
Security Challenges in Container Networks (1/2)

- Loss of container context
  - Do not know where packets actually come from in the host network namespace
  - Possible to forge packets on behalf of any other containers
- Limitations of IP-based access controls
  - Dynamically changed container IP addresses
  - Still vulnerable to L2 attacks due to limited scope
- Network policy explosion
  - Iptables: centralized mechanism for all network interfaces
  - Monolithic network rules daunting to manage



Security Challenges in Container Networks (2/2)

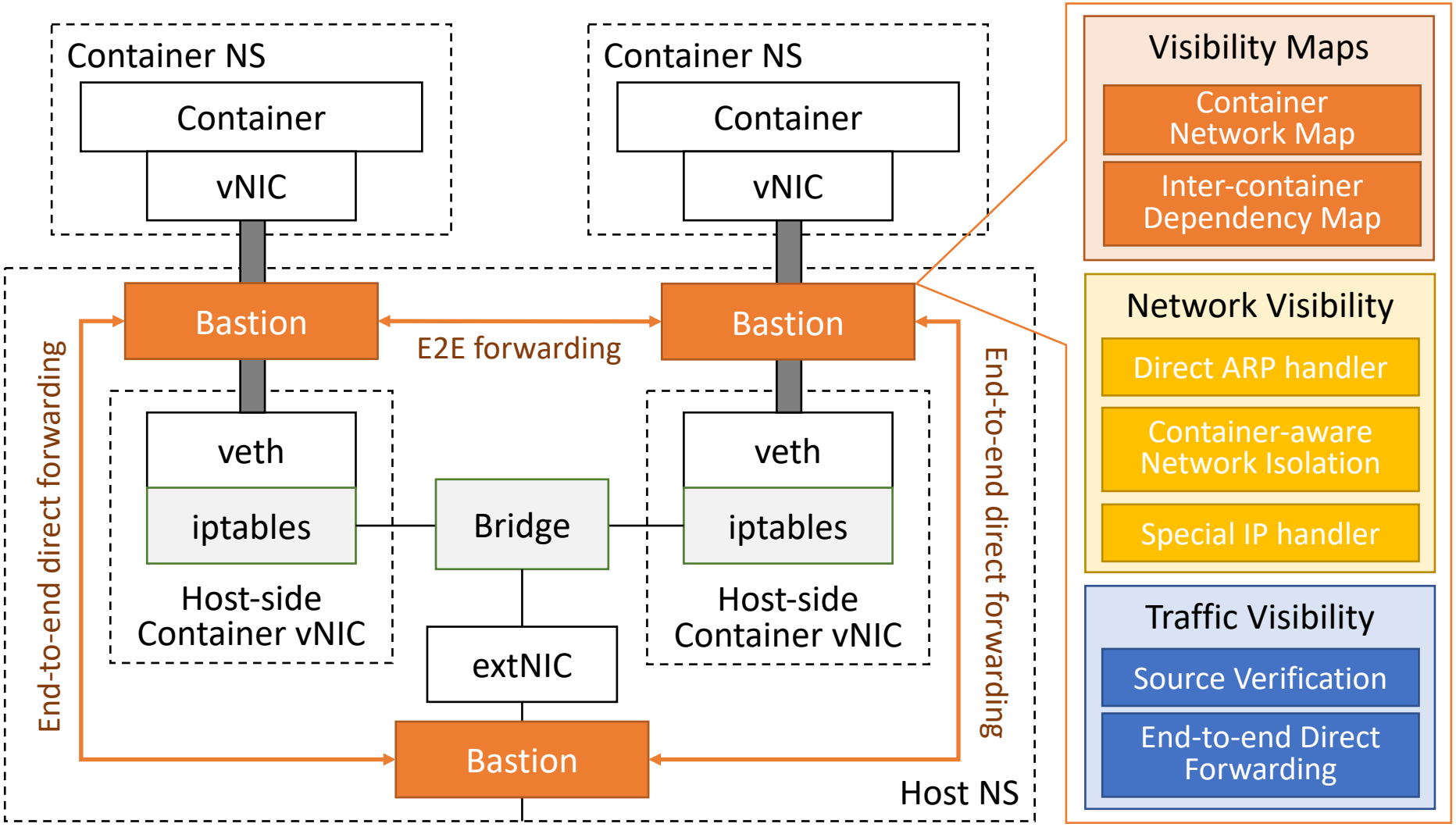
- Unrestricted host access
  - The gateway of a container network in the host network namespace
    - Necessary to access external networks
    - Allow accessing the services running at the host
- Unrestricted network-privileged containers
  - Network-privileged containers
    - Share the host network namespace
    - Used to directly expose services through a host IP
  - All network interfaces under control
  - No solutions that consider security policies for them



Bastion: Security Enforcement Network Stack

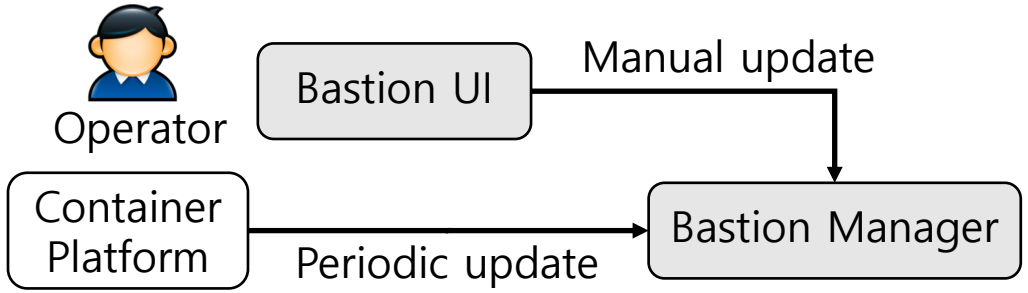
- Goal
  - Secure a container network through an intelligent container-aware communication sandbox
    - Protect network threats that abuse the security challenges of current container networks
    - Isolate inter-container communications according to their dependencies
- Key Components
  - Bastion manager
    - Collect all network information (e.g., network configurations and policies) from container platforms
  - Network visibility service
    - Provide fine-grained control over different network topology visibility per container application
  - Traffic visibility service
    - Securely isolate inter-container communications in a point-to-point manner
    - Prevent the exposure of inter-container network traffic to other peer containers

Bastion Architecture



Manager – Container Collection

- Container network and dependency maps
  - Collect the network information of deployed containers from container platforms
  - Extract the inter-container dependencies using container configurations and network policies



< Inter-container Dependency Map >

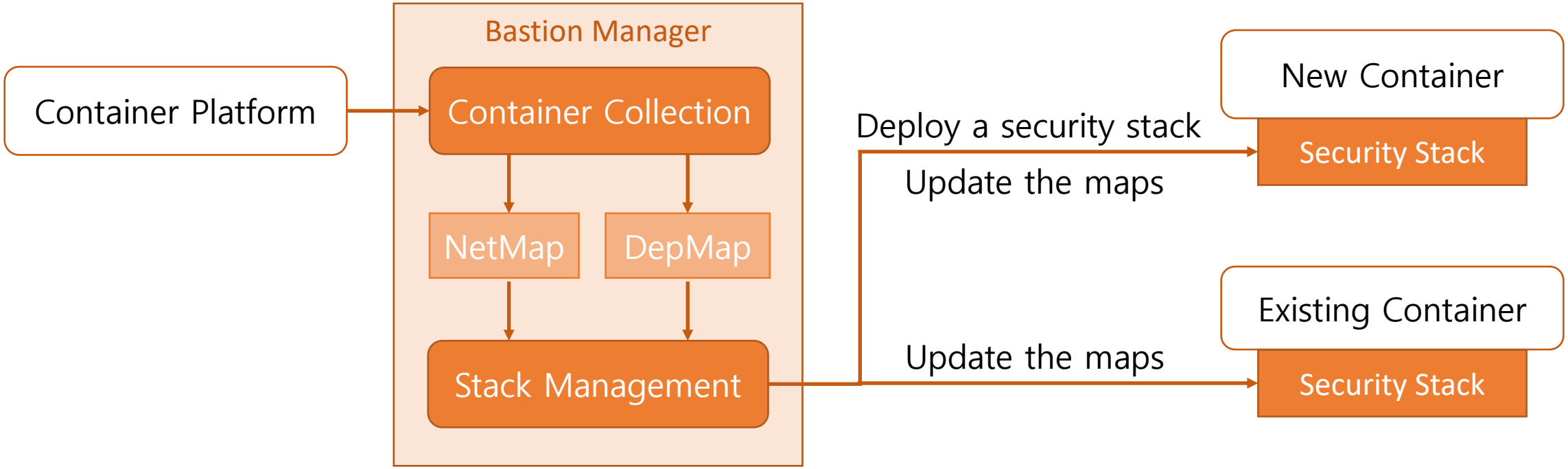
Source	Destination	Policy
WebApp	Service	Any
WebApp	Database	TCP:3306
Service	Database	TCP:3307

< Container Network Map >

Container	Network	Service	Interface	IP address	MAC address
WebApp-X1	WebService	WebApp	vethwepl6f964e8	10.32.0.2	96:0e:73:ef:86:fe
WebApp-X2	WebService	WebApp	vethweplb89dc35	10.32.0.3	6e:81:0f:a7:db:c7
Service-Y1	WebService	Service	vethweplb957e84	10.32.0.4	D6:bc:7b:20:32:c5
Database-Z1	WebService	Database	vethweplc5ee33c	10.32.0.5	42:a0:ae:b7:f5:97

Manager – Security Stack Management

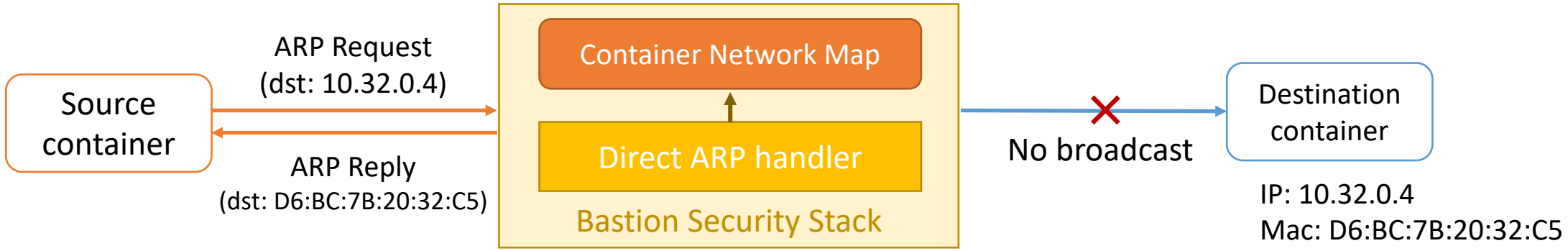
- Security enforcement network stack
  - Install a security enforcement network stack at a newly deployed container
  - Update the maps of the security stack in run time



Security Stack – Network Visibility Service (1/2)

- Container Discovery
  - First step to identify other containers (targets for communication)
  - Possible to be exploited for scanning all containers by malicious containers
    - Current solutions: No prevention against non-IP-based communications

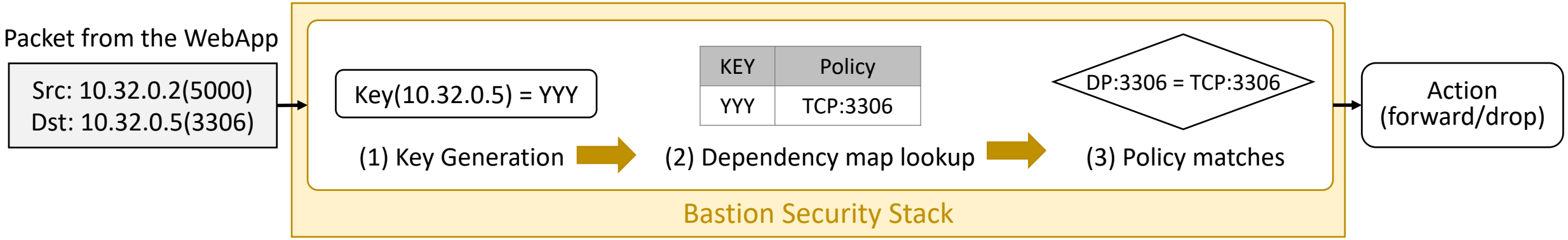
- Direct ARP handler
  - Directly response ARP requests at the security stack based on inter-container dependencies
    - Do not broadcast the requests to the network since this operation could be abused by an attacker



Security Stack – Network Visibility Service (2/2)

- Limitation of the direct ARP handler
  - Limited to container-level isolation (coverage issue)
  - Cannot address malicious network activities between inter-dependent containers

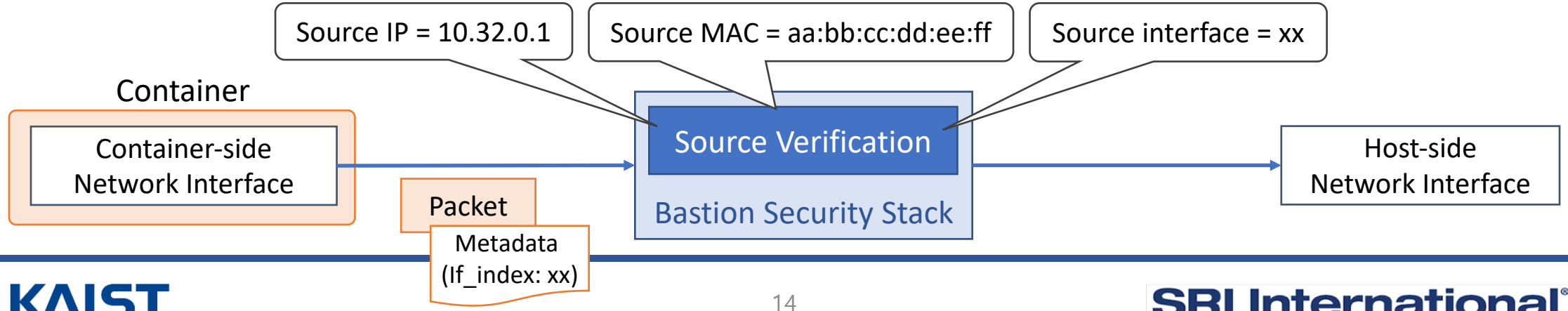
- Container-aware network isolation
  - Restrict container accesses according to finer-grained inter-container dependencies
    - In a similar way, Bastion restricts the opposite direction (10.32.0.5:3306 → 10.32.0.2:5000) as well



Security Stack – Traffic Visibility Service (1/2)

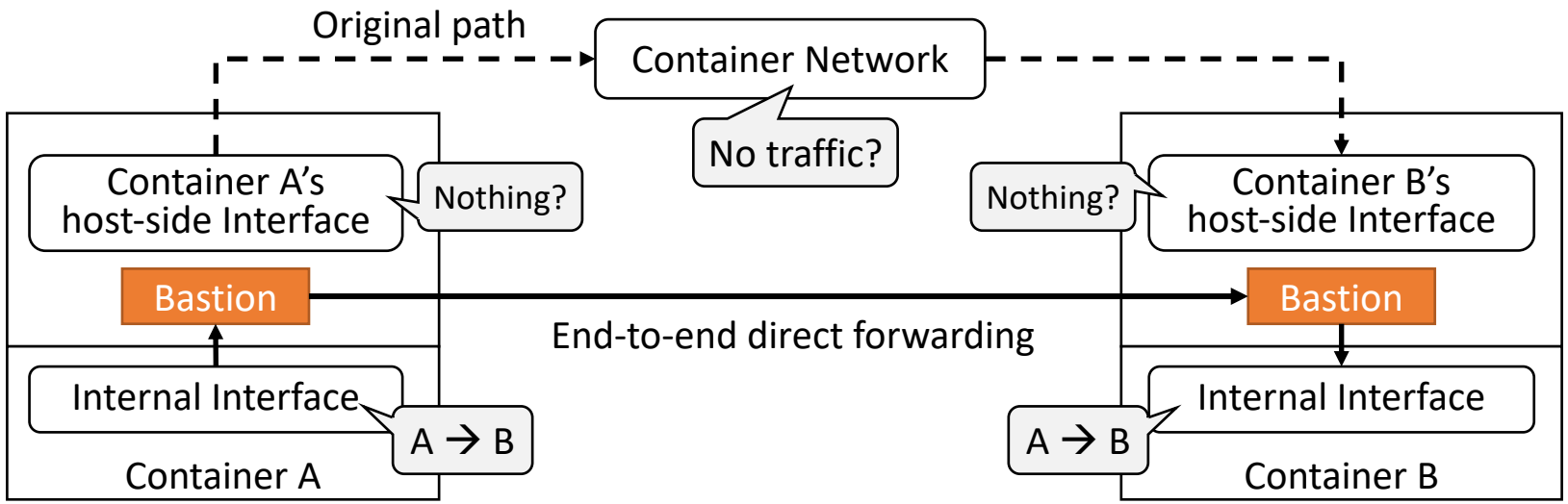
- How to verify sources in current solutions
  - Iptables: {source IP and MAC addresses} in packet headers
    - Vulnerable to Layer-2 attacks (e.g., ARP spoofing)
  - Open vSwitch: {source IP and MAC addresses} in packet headers + incoming virtual port
    - Not support a NOT operation
    - Need to define all possible flow rules with the above matching fields for source verification

- Source verification in Bastion
  - {source IP and MAC addresses} in packet headers + kernel metadata **at the container-side**



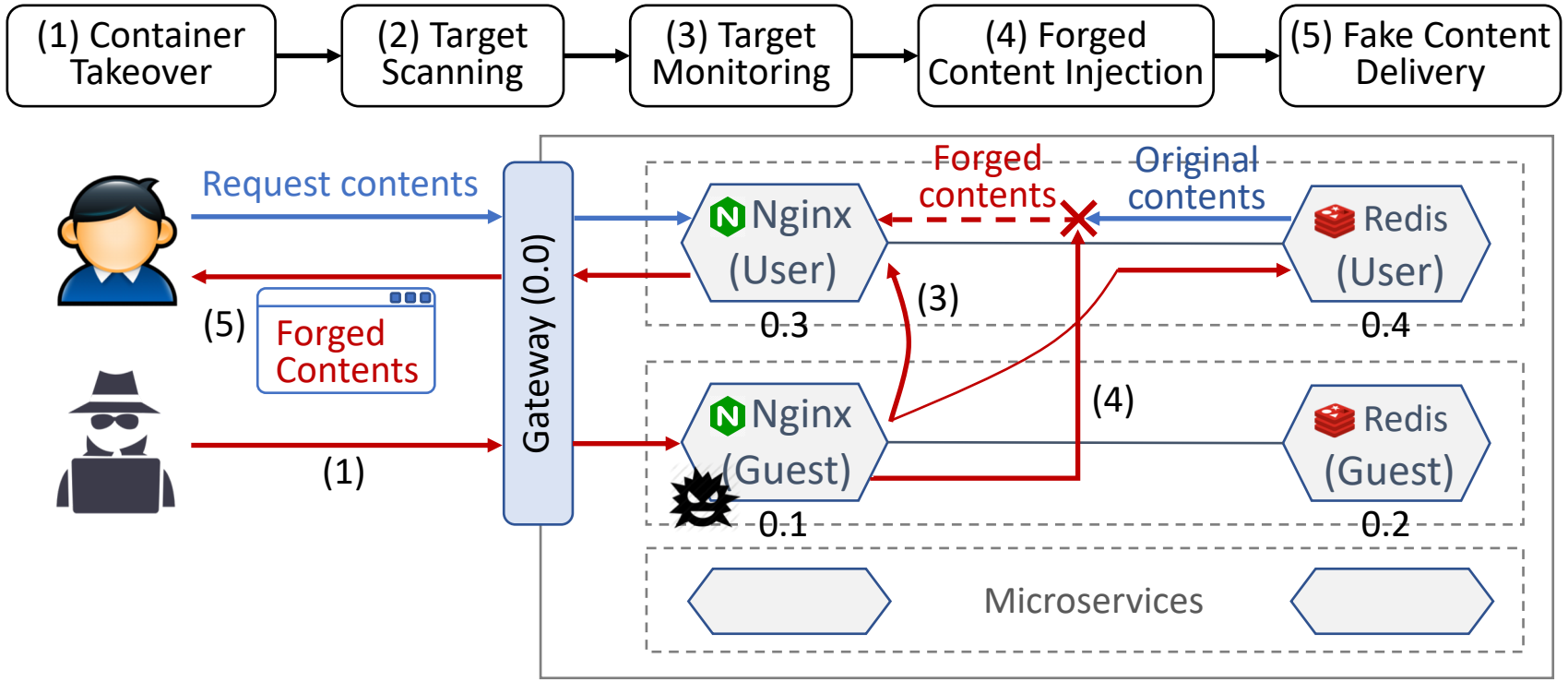
Security Stack – Traffic Visibility Service (2/2)

- End-to-end direct forwarding
  - Directly inject packets delivered from a source into a destination
  - Bypass the container network (the Linux network stack at the host-side)
  - Use the Linux networking features (XDP/eBPF)



Security Evaluation

- Goal
  - Replace the contents delivered to a user with a fake content



Security: Attack Scenario Verification

```
root@attacker-64769f9f6d-gsmds:/tmp# ./arpping 10.46.0.0/24
Number of active containers : 55 → The number of all deployed containers
10.46.0.0, 96:0e:73:ef:86:fd        10.46.0.2, a2:80:27:eb:3d:c
10.46.0.3, 72:1d:6e:15:3b:1e        10.46.0.4, 9a:0e:fa:71:24:2
10.46.0.5, ce:a1:33:bf:e1:58        10.46.0.6, 9e:b2:69:ec:5d:e
10.46.0 Nginx-User 17:0b:f0:9a:20   The original MAC address of Redis-User
```

1. Scan neighbor containers in a network (Nginx-Guest)

```
root@victim-1-fdd4f9f68-kwwtd:~# arp   The MAC address of Nginx-Guest
Address                  HWtype  HWaddress           Flags Ma
10.46.0.0                ether   96:0e:73:ef:86:fd   C
10.46.0.4 → Redis-User   ether   56:87:a7:15:17:69   C
10.46.0.1 → Nginx-Guest  ether   56:87:a7:15:17:69   C
kube-dns-86f4d74b45-zwp  ether   8e:1a:5d:bb:e8:c4   C
```

2. Spoof a target container (Nginx-User)

```
50:11.375228 IP 10.46.0.3.40133 > 10.46.0.4.8000: Flags [S],
50:11.375279 IP 10.46.0.4.8000 > 10.46.0.3.40133: Flags [S.],
```

3. Eavesdrop the network traffic of the target (Nginx-Guest)

Before injection

After injection

4. Inject a fake content (Client-side)

Security: Container Discovery

- No restriction on container reachability (without Bastion)

```
root@attacker-64769f9f6d-gsmds:/tmp# ./arpping 10.46.0.0/24
Number of active containers : 55 → The number of all deployed containers
10.46.0.0, 96:0e:73:ef:86:fd      10.46.0.2, a2:80:27:eb:3d:c
10.46.0.3, 72:1d:6e:15:3b:1e      10.46.0.4, 9a:0e:fa:71:24:2
10.46.0.5, ce:a1:33:bf:e1:58      10.46.0.6, 9e:b2:69:ec:5d:e
10.46.0.7, 52:17:0b:f0:9a:20      10.46.0.8, e2:29:5d:43:2f:6
10.46.0.9, 42:6e:ae:cd:2d:32      10.46.0.10, e6:ec:7d:5d:57
10.46.0.11, f2:2d:e0:c7:1f:7f      10.46.0.12, 26:0f:bd:29:28
10.46.0.13, 6e:a7:65:cf:05:35      10.46.0.14, 5a:02:9a:c6:95
```

- Limited container reachability (with Bastion)

```
root@attacker-64769f9f6d-gsmds:/tmp# ./arpping 10.46.0.0/24
Number of active containers : 2  → The number of dependent containers
10.46.0.0, 96:0e:73:ef:86:fd      10.46.0.2, a2:80:27:eb:3d:c
root@attacker-64769f9f6d-gsmds:/tmp#
```

Security: Passive Packet Monitoring

- Precondition: ARP spoofing attack is successfully conducted
- Network traffic for other containers → visible (without Bastion)

```
28:03.448899 IP 10.46.0.3.22167 > 10.46.0.4.8000: Flags [S],
28:03.448940 IP 10.46.0.3.22167 > 10.46.0.4.8000: Flags [S],
28:03.449027 IP 10.46.0.4.8000 > 10.46.0.3.22167: Flags [S.],
28:03.449047 IP 10.46.0.4.8000 > 10.46.0.3.22167: Flags [S.],
28:03.449155 IP 10.46.0.3.22167 > 10.46.0.4.8000: Flags [R],
28:03.449193 IP 10.46.0.3.22167 > 10.46.0.4.8000: Flags [R],
```

- Non-local traffic → invisible (with Bastion)
  - For verification, intentionally make the source-to-destination flows visible

```
29:52.941051 IP 10.46.0.3.12119 > 10.46.0.4.8000: Flags [S],
29:52.941117 IP 10.46.0.3.12119 > 10.46.0.4.8000: Flags [S],
29:52.941338 IP 10.46.0.3.12119 > 10.46.0.4.8000: Flags [R],
29:52.941384 IP 10.46.0.3.12119 > 10.46.0.4.8000: Flags [R],
```

Security: Active Packet Injection

- Precondition: ARP spoofing attack is successfully conducted
- Without Bastion

```
20:21.353453 IP 10.46.0.4.8000 > 10.46.0.3.34104: Flags [R],
20:21.353456 IP 10.46.0.4.8000 > 10.46.0.3.34104: Flags [R],
```

< Attacker side >

```
20:21.353420 IP 10.46.0.3.34104 > 10.46.0.4.8000: Flags [.],
20:21.353460 IP 10.46.0.4.8000 > 10.46.0.3.34104: Flags [R],
```

< Victim side >

- With Bastion

```
11:11.995745 IP 10.46.0.4.8000 > 10.46.0.3.12346: Flags [R],
11:11.995762 IP 10.46.0.4.8000 > 10.46.0.3.12346: Flags [R],
```

< Attacker side >

```
11:11.995614 IP 10.46.0.3.33452 > 10.46.0.4.8000: Flags [P.],
11:11.995655 IP 10.46.0.4.8000 > 10.46.0.3.33452: Flags [.],
11:11.995848 IP 10.46.0.4.8000 > 10.46.0.3.33452: Flags [.],
11:11.995866 IP 10.46.0.3.33452 > 10.46.0.4.8000: Flags [.],
```

< Victim side >

Performance: Inter-container Throughputs

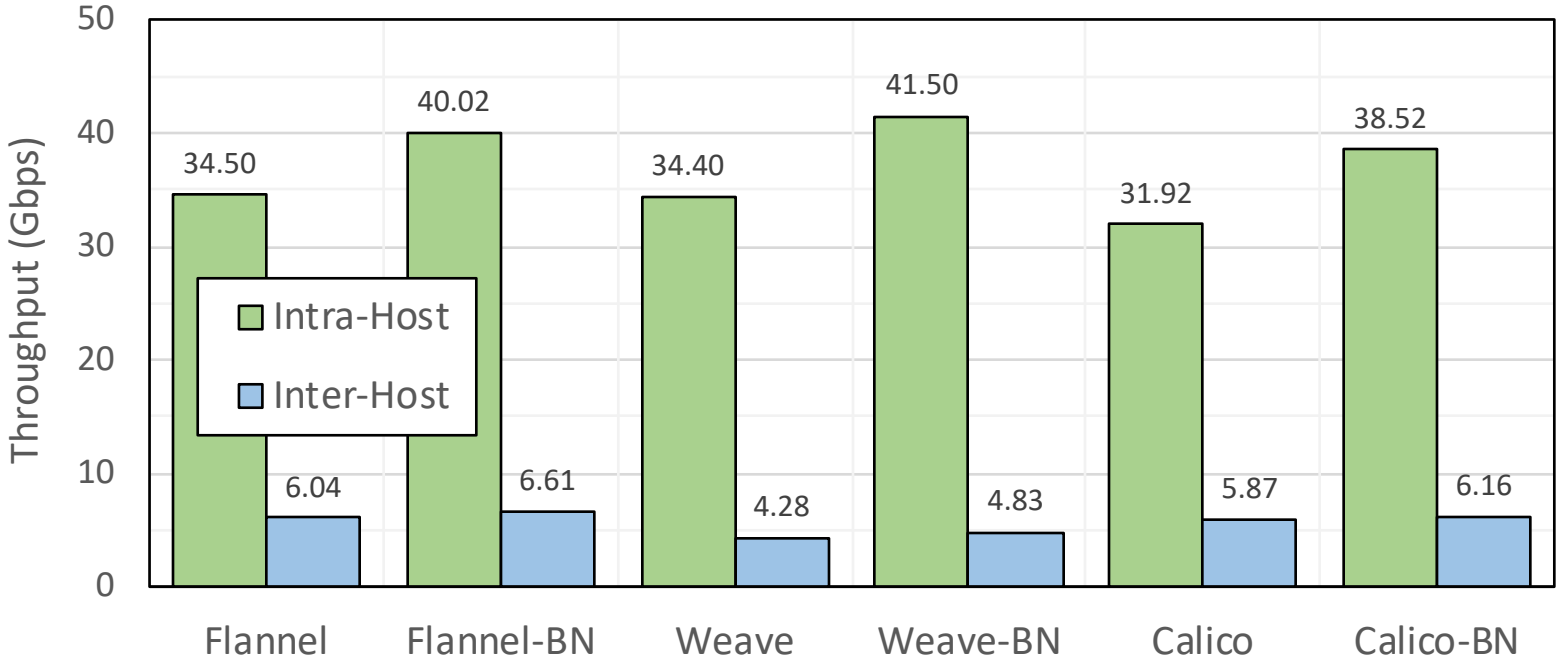
- Test environment
  - Xeon E5-2630v4 CPU with 64GB of RAM
  - Kubernetes with Weave overlay network, TCP traffic with iperf3

Throughput (Gbps)	Base (No Bastion)	Network Visibility only	Traffic Visibility only	Bastion (Fully deployed)
Within a host	34.4	33.7	41.8	41.5
Across hosts	4.28	4.23	4.91	4.83

  - Low inter-container throughputs across hosts?
    - Due to the heavy overheads in physical link traversal and tunneling between hosts

Performance: Bastion on Various Networks

- Test environment
  - Xeon E5-2630v4 CPU with 64GB of RAM
  - Kubernetes with Flannel, Weave, and Calico overlay networks, TCP traffic with iperf3



Summary

- The state of current container security
  - Mostly focus on the security of containers themselves
  - Less concern the security issues in container networks
- Security assessment of container networks
  - Identified how the security challenges in current container networks impact containers
- Bastion: security enforcement network stack for container networks
  - Intelligently isolate inter-container communications
  - Effectively mitigate lateral attacks against peer containers

Thank you for listening

namjh@kaist.ac.kr