BASTION: A Security Enforcement Network Stack for Container Networks

Jaehyun Nam\textsuperscript{1}, Seungsoo Lee\textsuperscript{1}, Hyunmin Seo\textsuperscript{1}, Phillip Porras\textsuperscript{2}, Vinod Yegneswaran\textsuperscript{2}, and Seungwon Shin\textsuperscript{1}

KAIST\textsuperscript{1} and SRI International\textsuperscript{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
    • 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

Current Container Security Solutions

- Container Image Integrity
  - Vulnerability scanning nested in container images using CVE databases

- Application Isolation
  - Container isolation \(\rightarrow\) Namespaces, cgroups, capabilities
  - Further restrictions on system resources \(\rightarrow\) 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

Container NS

Container

vNIC

Host NS

Container NS

Container

vNIC

Bastion

veth

iptables

Bridge

extNIC

Host-side Container vNIC

Host-side Container vNIC

E2E forwarding

End-to-end direct forwarding

Visibility Maps

Container Network Map

Inter-container Dependency Map

Network Visibility

Direct ARP handler

Container-aware Network Isolation

Special IP handler

Traffic Visibility

Source Verification

End-to-end Direct Forwarding

Host NS
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

<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
<th>Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebApp</td>
<td>Service</td>
<td>Any</td>
</tr>
<tr>
<td>WebApp</td>
<td>Database</td>
<td>TCP:3306</td>
</tr>
<tr>
<td>Service</td>
<td>Database</td>
<td>TCP:3307</td>
</tr>
</tbody>
</table>

< Inter-container Dependency Map >

<table>
<thead>
<tr>
<th>Container</th>
<th>Network</th>
<th>Service</th>
<th>Interface</th>
<th>IP address</th>
<th>MAC address</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebApp-X1</td>
<td>WebService</td>
<td>WebApp</td>
<td>vethwepl6f964e8</td>
<td>10.32.0.2</td>
<td>96:0e:73:ef:86:fe</td>
</tr>
<tr>
<td>WebApp-X2</td>
<td>WebService</td>
<td>WebApp</td>
<td>vethwepl89dc35</td>
<td>10.32.0.3</td>
<td>6e:81:0f:a7:db:c7</td>
</tr>
<tr>
<td>Service-Y1</td>
<td>WebService</td>
<td>Service</td>
<td>vethweplb957e84</td>
<td>10.32.0.4</td>
<td>D6:bc:7b:20:32:c5</td>
</tr>
<tr>
<td>Database-Z1</td>
<td>WebService</td>
<td>Database</td>
<td>vethweplc5ee33c</td>
<td>10.32.0.5</td>
<td>42:a0:ae:b7:f5:97</td>
</tr>
</tbody>
</table>

< Container Network Map >
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

Source container

ARP Request (dst: 10.32.0.4)

ARP Reply (dst: D6:BC:7B:20:32:C5)

Bastion Security Stack

Container Network Map

Direct ARP handler

No broadcast

Destination container

IP: 10.32.0.4
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

Packet from the WebApp
Src: 10.32.0.2(5000)
Dst: 10.32.0.5(3306)

Key(10.32.0.5) = YYY
(1) Key Generation

Key
Policy
YYY TCP:3306

DP:3306 = TCP:3306
(2) Dependency map lookup

Action
(forward/drop)
(3) Policy matches

Bastion Security Stack

Src: 10.32.0.5(3306)
Dst: 10.32.0.2(5000)
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

Container
  Container-side Network Interface

Packet
  Metadata (If_index: xx)

Bastion Security Stack
  Source Verification

Source IP = 10.32.0.1
Source MAC = aa:bb:cc:dd:ee:ff
Source interface = xx

Host-side Network Interface
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

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

2. Spoof a target container (Nginx-User)

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

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 \(\rightarrow\) 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:6
10.46.0.3, 72:1d:6e:15:3b:1e 10.46.0.4, 9a:0e:fa:71:24:3
10.46.0.7, 52:17:0b:fo:9a:20 10.46.0.8, e2:29:5d:43:2f:6
10.46.0.11, f2:2d:e0:c7:1f:7f 10.46.0.12, 26:0f:bd:29:2f:6
10.46.0.13, 6c: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 \(\rightarrow\) The number of dependent containers
10.46.0.0, 96:0e:73:ef:86:fd 10.46.0.2, a2:80:27:eb:3d:6
root@attacker-64769f9f6d-gsmds:/tmp#  
```
Security: Passive Packet Monitoring

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

<table>
<thead>
<tr>
<th>Source IP</th>
<th>Destination IP</th>
<th>Flags</th>
<th>Source Port</th>
<th>Destination Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>28:03.448899</td>
<td>10.46.0.4.8000</td>
<td>[S]</td>
<td>22167</td>
<td>22167</td>
</tr>
<tr>
<td>28:03.448940</td>
<td>10.46.0.4.8000</td>
<td>[S]</td>
<td>22167</td>
<td>22167</td>
</tr>
<tr>
<td>28:03.449027</td>
<td>10.46.0.3.22167</td>
<td>[S]</td>
<td>8000</td>
<td>10.46.0.4.8000</td>
</tr>
<tr>
<td>28:03.449047</td>
<td>10.46.0.3.22167</td>
<td>[S.]</td>
<td>8000</td>
<td>10.46.0.4.8000</td>
</tr>
<tr>
<td>28:03.449155</td>
<td>10.46.0.4.8000</td>
<td>[R]</td>
<td>22167</td>
<td>22167</td>
</tr>
<tr>
<td>28:03.449193</td>
<td>10.46.0.4.8000</td>
<td>[R]</td>
<td>22167</td>
<td>22167</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Source IP</th>
<th>Destination IP</th>
<th>Flags</th>
<th>Source Port</th>
<th>Destination Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>29:52.941051</td>
<td>10.46.0.4.8000</td>
<td>[S]</td>
<td>12119</td>
<td>22167</td>
</tr>
<tr>
<td>29:52.941117</td>
<td>10.46.0.4.8000</td>
<td>[S]</td>
<td>12119</td>
<td>22167</td>
</tr>
<tr>
<td>29:52.941338</td>
<td>10.46.0.4.8000</td>
<td>[R]</td>
<td>12119</td>
<td>22167</td>
</tr>
<tr>
<td>29:52.941384</td>
<td>10.46.0.4.8000</td>
<td>[R]</td>
<td>12119</td>
<td>22167</td>
</tr>
</tbody>
</table>
Security: Active Packet Injection

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

\[
\begin{align*}
20:21.353453 & \text{ IP } 10.46.0.4.8000 \rightarrow 10.46.0.3.34104: \text{ Flags } [R], \\
20:21.353456 & \text{ IP } 10.46.0.4.8000 \rightarrow 10.46.0.3.34104: \text{ Flags } [R],
\end{align*}
\]

\(< \text{ Attacker side }>\)

\[
\begin{align*}
20:21.353420 & \text{ IP } 10.46.0.3.34104 \rightarrow 10.46.0.4.8000: \text{ Flags } [.] , \\
20:21.353460 & \text{ IP } 10.46.0.3.34104 \rightarrow 10.46.0.4.8000: \text{ Flags } [R],
\end{align*}
\]

\(< \text{ Victim side }>\)

- With Bastion

\[
\begin{align*}
11:11.995745 & \text{ IP } 10.46.0.4.8000 \rightarrow 10.46.0.3.12346: \text{ Flags } [R], \\
11:11.995762 & \text{ IP } 10.46.0.4.8000 \rightarrow 10.46.0.3.12346: \text{ Flags } [R],
\end{align*}
\]

\(< \text{ Attacker side }>\)

\[
\begin{align*}
11:11.995614 & \text{ IP } 10.46.0.3.33452 \rightarrow 10.46.0.4.8000: \text{ Flags } [P.], \\
11:11.995655 & \text{ IP } 10.46.0.4.8000 \rightarrow 10.46.0.3.33452: \text{ Flags } [.] , \\
11:11.995848 & \text{ IP } 10.46.0.3.33452 \rightarrow 10.46.0.4.8000: \text{ Flags } [.] , \\
11:11.995866 & \text{ IP } 10.46.0.3.33452 \rightarrow 10.46.0.4.8000: \text{ Flags } [.] ,
\end{align*}
\]

\(< \text{ Victim side }>\)
Performance: Inter-container Throughputs

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

<table>
<thead>
<tr>
<th>Throughput (Gbps)</th>
<th>Base (No Bastion)</th>
<th>Network Visibility only</th>
<th>Traffic Visibility only</th>
<th>Bastion (Fully deployed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Within a host</td>
<td>34.4</td>
<td>33.7</td>
<td>41.8</td>
<td>41.5</td>
</tr>
<tr>
<td>Across hosts</td>
<td>4.28</td>
<td>4.23</td>
<td>4.91</td>
<td>4.83</td>
</tr>
</tbody>
</table>

- 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