CLARION: Sound and Clear Provenance Tracking for Microservice Deployments

XUTONG CHEN¹, HASSAAN IRSHAD², YAN CHEN¹, ASHISH GEHANI², VINOD YEGNESWARAN²

NORTHWESTERN UNIVERSITY¹, SRI INTERNATIONAL²
Microservices, Containers and Provenance Tracking

**Microservices**
- Enable better resource utilization
- Increasing practical adoption

**Containers**
- Portable and lightweight software isolation technique
- New security challenges

**Provenance Tracking**
- State-of-the-art host forensic monitoring solution
- Transferred from traditional OS scenario to container scenario

A sample provenance graph generated by SPADE, Middleware’12
Motivating Example: Insider Attack

- Three major steps of a trivial insider attack is shown on the left.
- Perform all three steps in both the host and the container.
- We drill down just on the first step in the next slide for simplicity.
- Motivate our work by illustrating limitations in the provenance graphs from three contemporary solutions.
Existing Provenance Tracking Solutions

**Namespace-unaware**

**Container-aware**

*Winnower: NDSS 2018*

Fail on soundness
- Fragmentation in `bash-cat` provenance
- Ambiguities on `/etc/passwd`
Existing Provenance Tracking Solutions

Namespace-unaware Container-aware
Winnower: NDSS 2018
Fail on soundness
• Fragmentation in bash-cat provenance
• Ambiguities on /etc/passwd

Namespace-aware Container-unaware
Camflow: SoCC 2017
Fail on clarity
• Unclear insight about the container
Existing Provenance Tracking Solutions

Namespace-unaware

Container-aware

Winnower: NDSS 2018
Fail on soundness
- Fragmentation in `bash-cat` provenance
- Ambiguities on `/etc/passwd`

Namespace-aware

Container-unaware

Camflow: SoCC 2017
Fail on clarity
- Unclear insight about the container

Namespace-unaware

Container-unaware

SPADE: Middleware 2012
Fail on both soundness and clarity
CLARION Solution and Key Contributions

- The first in-depth analysis of the implications of namespaces on provenance tracking.

- CLARION: A namespace-aware and container-aware provenance tracking solution.
  - Extension of the SPADE Provenance Engine to address clarity and soundness issues.
  - Linux kernel module, netfilter hooks, and modifications to SPADE application-level handlers.

- Comprehensive evaluation of the effectiveness, efficiency, and generality.
Inconsistency in low-level events occur when low-level events report data values from varying (host / container) contexts that lead to vertex splitting in provenance graphs.

Consider the “clone” Linux audit event as an example, its “pid” field (5903) is in host context but its “exit” field (2), which is also a PID, is affected by pid namespace and in container context.

If provenance tracking system directly uses those two data fields to generate provenance, graph fragmentation will occur.

```
type=SYSCALL msg=audit(1567029444.851:431219): arch=c000003e syscall=56 success=yes exit=2 a0=3d0f00 a1=7f81aa8f8fb0 a2=7f81aa8f99d0 a3=7f81aa8f99d0 items=0 ppid=5880 pid=5903 auid=4294967295 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=none ses=4294967295 comm="runc:[2:INIT]" exe="/" key=(null)
```
Soundness Challenge: In-depth Analysis of Namespace Implication

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Virtualized System Resource</th>
<th>What low-level events are affected?</th>
<th>Impact on soundness</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID</td>
<td>Process IDs</td>
<td>Events with PIDs</td>
<td>Yes, fragmentation</td>
</tr>
<tr>
<td>Mount</td>
<td>Mount points</td>
<td>Events with file pathnames</td>
<td>Yes, ambiguities</td>
</tr>
<tr>
<td>Network</td>
<td>Network device/stack, etc.</td>
<td>Events with Local IPs/ports</td>
<td>Yes, both</td>
</tr>
<tr>
<td>IPC</td>
<td>SYSV IPC objects &amp; POSIX msg queue</td>
<td>Events with ID/names of SYSV IPC object/POSIX msg queues</td>
<td>Yes, ambiguities</td>
</tr>
<tr>
<td>User</td>
<td>User/group IDs</td>
<td>Events with GIDs and UIDs</td>
<td>No</td>
</tr>
<tr>
<td>Time, UTS, Cgroup</td>
<td>Boot/ monotonic clocks; Host/NIS domain name; Cgroup root directory</td>
<td>Does not affect provenance dataflow</td>
<td>N/A</td>
</tr>
</tbody>
</table>

PID/Mount/Network/IPC ns can impact soundness. Additional details provided in paper (Tables 1 and 2).
Soundness Challenge is not specific to Linux Audit

Question: *Does soundness challenge persist on other Linux auditing subsystems/OS platforms?*

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Sysdig</th>
<th>LTTng</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Mount</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Network</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>IPC</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Namespace</th>
<th>BSD Jail</th>
<th>Solaris Zone</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Mount</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Network</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>IPC</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

- Soundness challenge persists in other Linux auditing subsystems and OS platforms.
- LTTng provides virtualized PIDs and host PIDs together so fragmentation can be solved by correlation.
- No fragmentation is caused by PID in BSD Jail/Solaris Zone because all PIDs are host PIDs.
- Additional details provided in paper (Table 3 and 4).
Soundness Challenge: Mapping Virtualized Namespaces

- **Key design:** establish a mapping between the host view and the container view on virtualized resources.

- The mapping correlates the virtualized data with the host data.

- It helps provenance tracking system to use the **consistent** provenance data.

- The principal namespaces that impact provenance are PID, Mount, Network, and IPC.
Clarity: Essential Container Semantic Patterns

• Understanding essential container semantics helps by:
  1. accelerating the automatic analysis process.
  2. simplifying security analysis.
Effectiveness: Real-world Exploit

• Question: How effective is CLARION in dealing with real-world container microservice attacks?
• Answer: Real-world exploit

<table>
<thead>
<tr>
<th>CVE #</th>
<th>Description</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-14271</td>
<td>docker-tar related</td>
<td>High. Achieve privilege escalation.</td>
</tr>
<tr>
<td>2018-15664</td>
<td>docker-cp related</td>
<td>Normal. Achieve container host file system modification.</td>
</tr>
</tbody>
</table>

• We will focus on CVE 2019-5736 (runC) to show the effectiveness:
  • The exploit involves 2 starts of the compromised container.
  • The first start does the configuration.
  • The second start does the actual damage.
Effectiveness: CVE 2019-5736
Effectiveness: CVE 2019-5736
Generality: Container Initialization

Question: Is CLARION solution generally applicable across different container engines?

Answer: Container initialization graph generation and quantitative results for different containers

Docker Hello-world Init

Rkt Hello-world Init
## Efficiency: Runtime/Storage Overhead

### Runtime Overhead Comparison of Container Provenance Systems

<table>
<thead>
<tr>
<th>Service</th>
<th>Base (secs)</th>
<th>Linux (secs)</th>
<th>Audit (secs)</th>
<th>SPADE (secs)</th>
<th>CLARION (secs)</th>
<th>Incremental Overhead (CLARION)</th>
<th>Overall Overhead (Audit + SPADE + CLARION)</th>
<th>Overall Overhead (CamFlow)</th>
<th>Overall Overhead (SEL-Audit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>frontend</td>
<td>1503 s</td>
<td>1550 s</td>
<td>1558 s</td>
<td>1578 s</td>
<td>1.3%</td>
<td>3.7%</td>
<td>4.8%</td>
<td>32.4%</td>
<td>25.0%</td>
</tr>
<tr>
<td>productcatalog</td>
<td>668 s</td>
<td>679 s</td>
<td>681 s</td>
<td>691 s</td>
<td>1.5%</td>
<td>3.4%</td>
<td>9.1%</td>
<td>25.0%</td>
<td>8.5%</td>
</tr>
<tr>
<td>service</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>currencyservice</td>
<td>1104 s</td>
<td>1139 s</td>
<td>1153 s</td>
<td>1169 s</td>
<td>1.4%</td>
<td>5.9%</td>
<td>12.9%</td>
<td>8.5%</td>
<td>11.5%</td>
</tr>
<tr>
<td>paymentservice</td>
<td>1082 s</td>
<td>1123 s</td>
<td>1126 s</td>
<td>1143 s</td>
<td>1.5%</td>
<td>5.6%</td>
<td>11.5%</td>
<td>9.7%</td>
<td>17.6%</td>
</tr>
<tr>
<td>shippingservice</td>
<td>434 s</td>
<td>446 s</td>
<td>449 s</td>
<td>451 s</td>
<td>0.4%</td>
<td>3.9%</td>
<td>22.5%</td>
<td>25.8%</td>
<td>13.9%</td>
</tr>
<tr>
<td>emailservice</td>
<td>929 s</td>
<td>960 s</td>
<td>1028 s</td>
<td>1068 s</td>
<td>3.9%</td>
<td>15.0%</td>
<td>1.2%</td>
<td>17.6%</td>
<td>13.9%</td>
</tr>
<tr>
<td>checkoutservice</td>
<td>682 s</td>
<td>719 s</td>
<td>714 s</td>
<td>734 s</td>
<td>2.8%</td>
<td>7.6%</td>
<td>3.2%</td>
<td>13.9%</td>
<td>13.9%</td>
</tr>
<tr>
<td>recommendation</td>
<td>8726 s</td>
<td>9418 s</td>
<td>9337 s</td>
<td>9729 s</td>
<td>4.2%</td>
<td>11.5%</td>
<td>9.5%</td>
<td>19.5%</td>
<td>19.5%</td>
</tr>
<tr>
<td>service</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>adservice</td>
<td>4438 s</td>
<td>4454 s</td>
<td>4518 s</td>
<td>4571 s</td>
<td>1.2%</td>
<td>3.0%</td>
<td>5.3%</td>
<td>8.5%</td>
<td>8.5%</td>
</tr>
<tr>
<td>loadgenerator</td>
<td>200 s</td>
<td>208 s</td>
<td>212 s</td>
<td>215 s</td>
<td>1.4%</td>
<td>7.5%</td>
<td>20.4%</td>
<td>29.4%</td>
<td>29.4%</td>
</tr>
</tbody>
</table>

### Storage Overhead Comparison

<table>
<thead>
<tr>
<th>SEL-Audit</th>
<th>CamFlow</th>
<th>SPADE</th>
<th>CLARION</th>
<th>Incremental Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>168.79 GB</td>
<td>312.56 GB</td>
<td>174.68 GB</td>
<td>181.75 GB</td>
<td>4.05%</td>
</tr>
</tbody>
</table>

Runtime/storage incremental overheads < 5%
Conclusion

• Existing provenance tracking solutions are inadequate for microservice scenarios
  • Namespace unawareness causes fragmentation and ambiguities (soundness).
  • Container unawareness leads to missing essential container semantics (clarity).

• CLARION
  • The first namespace-aware and container-aware provenance tracking solution
  • Comprehensive evaluation shows effectiveness, generality and efficiency of our solution
    ■ Effectiveness: We generate sound and clear provenance graphs of 3 real-world CVEs, which outperform the graphs generated by the traditional solution.
    ■ Generality: We show that our solution is independent of container engines by providing the container initialization graphs and quantitative results for 3 container engines.
    ■ Efficiency: We use a microservice benchmark to test the runtime/storage overhead of CLARION and find that the overhead is <5% over SPADE.

Questions or feedback: Xutong Chen <xutongchen2021@u.northwestern.edu>