RESIN: A Holistic Service for Dealing with Memory Leaks in Production Cloud Infrastructure

Chang Lou, Cong Chen, Peng Huang, Yingnong Dang, Si Qin, Xinsheng Yang, Xukun Li, Qingwei Lin, Murali Chintalapati
Memory leak is a notorious issue in cloud

- Service processes
- OS kernel
- Device drivers
- Host software components
Memory leak is a notorious issue in cloud systems. Changes in code, OS kernel, service processes, device drivers, and host software components can lead to memory leaks.
Memory leak is a notorious issue in cloud

code changes

OS kernel

service processes

device drivers

host software components
Memory leak is a notorious issue in cloud

- code changes
- service processes
- OS kernel
- device drivers
- host software components
- performance degradation
- host reboot
- VM allocations denied
Memory leak is a notorious issue in cloud computing. Changes in code can lead to memory leaks in OS kernels, service processes, device drivers, and host software components, causing performance degradation, host reboots, or VM allocations denied.
A production memory leak in cloud

<table>
<thead>
<tr>
<th>ID</th>
<th>PERM</th>
<th>PROT</th>
<th>SRC</th>
<th>DEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>✓</td>
<td>TCP</td>
<td>ANY</td>
<td>192.168.1.21</td>
</tr>
<tr>
<td>2</td>
<td>✓</td>
<td>TCP</td>
<td>ANY</td>
<td>192.168.1.22</td>
</tr>
<tr>
<td>3</td>
<td>✓</td>
<td>UDP</td>
<td>ANY</td>
<td>192.168.1.23</td>
</tr>
<tr>
<td>4</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.27</td>
</tr>
</tbody>
</table>

Firewall service

Host machines in Azure clusters
A production memory leak in cloud

<table>
<thead>
<tr>
<th>ID</th>
<th>PERM</th>
<th>PROT</th>
<th>SRC</th>
<th>DEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>✓</td>
<td>TCP</td>
<td>ANY</td>
<td>192.168.1.21</td>
</tr>
<tr>
<td>2</td>
<td>✓</td>
<td>TCP</td>
<td>ANY</td>
<td>192.168.1.22</td>
</tr>
<tr>
<td>3</td>
<td>✓</td>
<td>UDP</td>
<td>ANY</td>
<td>192.168.1.23</td>
</tr>
<tr>
<td>4</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.27</td>
</tr>
<tr>
<td>5</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
<tr>
<td>6</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
<tr>
<td>7</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
<tr>
<td>8</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
<tr>
<td>9</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
<tr>
<td>10</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
<tr>
<td>11</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
<tr>
<td>12</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
</tbody>
</table>

Firewall service

Host machines in Azure clusters

... rule object leaked!
A production memory leak in cloud

<table>
<thead>
<tr>
<th>ID</th>
<th>PERM</th>
<th>PROT</th>
<th>SRC</th>
<th>DEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>✓</td>
<td>TCP</td>
<td>ANY</td>
<td>192.168.1.21</td>
</tr>
<tr>
<td>2</td>
<td>✓</td>
<td>TCP</td>
<td>ANY</td>
<td>192.168.1.22</td>
</tr>
<tr>
<td>3</td>
<td>✓</td>
<td>UDP</td>
<td>ANY</td>
<td>192.168.1.23</td>
</tr>
<tr>
<td>4</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.27</td>
</tr>
<tr>
<td>5</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
<tr>
<td>6</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
<tr>
<td>7</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
<tr>
<td>8</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
<tr>
<td>...</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
</tbody>
</table>

Host machines in Azure clusters

Firewall service

testing and static checker report no bug ✓
A production memory leak in cloud

<table>
<thead>
<tr>
<th>ID</th>
<th>PERM</th>
<th>PROT</th>
<th>SRC</th>
<th>DEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>✓</td>
<td>TCP</td>
<td>ANY</td>
<td>192.168.1.21</td>
</tr>
<tr>
<td>2</td>
<td>✓</td>
<td>TCP</td>
<td>ANY</td>
<td>192.168.1.22</td>
</tr>
<tr>
<td>3</td>
<td>✓</td>
<td>UDP</td>
<td>ANY</td>
<td>192.168.1.23</td>
</tr>
<tr>
<td>4</td>
<td>✗</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.27</td>
</tr>
<tr>
<td>5</td>
<td>✗</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
<tr>
<td>6</td>
<td>✗</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
<tr>
<td>7</td>
<td>✗</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
<tr>
<td>8</td>
<td>✗</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>..</td>
<td>✗</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
</tbody>
</table>

**testing and static checker report no bug ✓**

Firewall service

Host machines in Azure clusters

only leak on some hosts 🔴
A production memory leak in cloud

<table>
<thead>
<tr>
<th>ID</th>
<th>PERM</th>
<th>PROT</th>
<th>SRC</th>
<th>DEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>✓</td>
<td>TCP</td>
<td>ANY</td>
<td>192.168.1.21</td>
</tr>
<tr>
<td>2</td>
<td>✓</td>
<td>TCP</td>
<td>ANY</td>
<td>192.168.1.22</td>
</tr>
<tr>
<td>3</td>
<td>✓</td>
<td>UDP</td>
<td>ANY</td>
<td>192.168.1.23</td>
</tr>
<tr>
<td>4</td>
<td>×</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.27</td>
</tr>
<tr>
<td>5</td>
<td>×</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
<tr>
<td>6</td>
<td>×</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
<tr>
<td>7</td>
<td>×</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
<tr>
<td>8</td>
<td>×</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
</tbody>
</table>

testing and static checker report no bug ✓

only leak on some hosts

Firewall service

Host machines in Azure clusters
The leak is cross-component

<table>
<thead>
<tr>
<th>ID</th>
<th>PERM</th>
<th>PROT</th>
<th>SRC</th>
<th>DEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>✓</td>
<td>TCP</td>
<td>ANY</td>
<td>192.168.1.21</td>
</tr>
<tr>
<td>2</td>
<td>✓</td>
<td>TCP</td>
<td>ANY</td>
<td>192.168.1.22</td>
</tr>
<tr>
<td>3</td>
<td>✓</td>
<td>UDP</td>
<td>ANY</td>
<td>192.168.1.23</td>
</tr>
<tr>
<td>4</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.27</td>
</tr>
</tbody>
</table>

Firewall service

load configured rules

Config agent
The leak is cross-component

<table>
<thead>
<tr>
<th>ID</th>
<th>PERM</th>
<th>PROT</th>
<th>SRC</th>
<th>DEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>✓</td>
<td>TCP</td>
<td>ANY</td>
<td>192.168.1.21</td>
</tr>
<tr>
<td>2</td>
<td>✓</td>
<td>TCP</td>
<td>ANY</td>
<td>192.168.1.22</td>
</tr>
<tr>
<td>3</td>
<td>✓</td>
<td>UDP</td>
<td>ANY</td>
<td>192.168.1.23</td>
</tr>
<tr>
<td>4</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.27</td>
</tr>
<tr>
<td>5</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
</tbody>
</table>

Firewall service

Config agent

load configured rules

register rules
The leak is cross-component

<table>
<thead>
<tr>
<th>ID</th>
<th>PERM</th>
<th>PROT</th>
<th>SRC</th>
<th>DEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>✓</td>
<td>TCP</td>
<td>ANY</td>
<td>192.168.1.21</td>
</tr>
<tr>
<td>2</td>
<td>✓</td>
<td>TCP</td>
<td>ANY</td>
<td>192.168.1.22</td>
</tr>
<tr>
<td>3</td>
<td>✓</td>
<td>UDP</td>
<td>ANY</td>
<td>192.168.1.23</td>
</tr>
<tr>
<td>4</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.27</td>
</tr>
<tr>
<td>5</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
</tbody>
</table>

Firewall service

Config agent

- load configured rules
- register rules
- remove rules (normal exit)
The leak is cross-component

<table>
<thead>
<tr>
<th>ID</th>
<th>PERM</th>
<th>PROT</th>
<th>SRC</th>
<th>DEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>✓</td>
<td>TCP</td>
<td>ANY</td>
<td>192.168.1.21</td>
</tr>
<tr>
<td>2</td>
<td>✓</td>
<td>TCP</td>
<td>ANY</td>
<td>192.168.1.22</td>
</tr>
<tr>
<td>3</td>
<td>✓</td>
<td>UDP</td>
<td>ANY</td>
<td>192.168.1.23</td>
</tr>
<tr>
<td>4</td>
<td>✗</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.27</td>
</tr>
<tr>
<td>5</td>
<td>✗</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
</tbody>
</table>

Firewall service

Config agent

load configured rules

register rules → crash!
The leak is cross-component

<table>
<thead>
<tr>
<th>ID</th>
<th>PERM</th>
<th>PROT</th>
<th>SRC</th>
<th>DEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>✓</td>
<td>TCP</td>
<td>ANY</td>
<td>192.168.1.21</td>
</tr>
<tr>
<td>2</td>
<td>✓</td>
<td>TCP</td>
<td>ANY</td>
<td>192.168.1.22</td>
</tr>
<tr>
<td>3</td>
<td>✓</td>
<td>UDP</td>
<td>ANY</td>
<td>192.168.1.23</td>
</tr>
<tr>
<td>4</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.27</td>
</tr>
<tr>
<td>5</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
<tr>
<td>6</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
<tr>
<td>7</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
<tr>
<td>8</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
<tr>
<td>9</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
<tr>
<td>10</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
<tr>
<td>11</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
<tr>
<td>12</td>
<td>✘</td>
<td>ANY</td>
<td>ANY</td>
<td>192.168.1.44</td>
</tr>
</tbody>
</table>

...

7,092,866 more rows
The leak is configuration-triggered

<table>
<thead>
<tr>
<th>ID</th>
<th>PERM</th>
<th>PROT</th>
<th>SRC</th>
<th>DEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>✓</td>
<td>TCP</td>
<td>192.168.1.21</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>✓</td>
<td>TCP</td>
<td>192.168.1.22</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>✓</td>
<td>UDP</td>
<td>192.168.1.23</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>✘</td>
<td>ANY</td>
<td>192.168.1.27</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>✘</td>
<td>ANY</td>
<td>192.168.1.44</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>✘</td>
<td>ANY</td>
<td>192.168.1.44</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>✘</td>
<td>ANY</td>
<td>192.168.1.44</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>✘</td>
<td>ANY</td>
<td>192.168.1.44</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>✘</td>
<td>ANY</td>
<td>192.168.1.44</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>✘</td>
<td>ANY</td>
<td>192.168.1.44</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>✘</td>
<td>ANY</td>
<td>192.168.1.44</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>✘</td>
<td>ANY</td>
<td>192.168.1.44</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td>7,092,866 more rows</td>
</tr>
</tbody>
</table>
Why detection is still a problem in cloud?
Why detection is still a problem in cloud?

Practice 1: static approach
- run static analysis on source codes
- expose bugs without running programs

Firewall service
Why detection is still a problem in cloud?

- Practice 1: static approach
  - run static analysis on source codes
  - expose bugs without running programs

- Limitations
  - no overhead, but not scalable or accurate
Why detection is still a problem in cloud?

- Practice 2: dynamic approach
  - instrument programs and track the object lifetime at runtime to find leaked objects
  - detect leaks and pinpoint leaked objects
Why detection is still a problem in cloud?

- Practice 2: dynamic approach
  - instrument programs and track the object lifetime at runtime to find leaked objects
  - detect leaks and pinpoint leaked objects

+ overhead
Why detection is still a problem in cloud?

- Practice 2: dynamic approach
  - instrument programs and track the object lifetime at runtime to find leaked objects
  - detect leaks and pinpoint leaked objects

+ overhead
Why detection is still a problem in cloud?

Practice 2: dynamic approach

- instrument programs and track the object lifetime at runtime to find leaked objects
- detect leaks and pinpoint leaked objects

+ overhead
Why detection is still a problem in cloud?

- Practice 2: dynamic approach
  - instrument programs and track the object lifetime at runtime to find leaked objects
  - detect leaks and pinpoint leaked objects

+ overhead
Why detection is still a problem in cloud?

- Practice 2: dynamic approach
  - instrument programs and track the object lifetime at runtime to find leaked objects
  - detect leaks and pinpoint leaked objects

- Limitations
  - hard tradeoff among accuracy, scalability and overhead
Detecting memory leaks with RESIN

- Our response is RESIN
  - achieve **accuracy**, **scalability** and **low overhead** all together
Detecting memory leaks with RESIN

‣ Our response is RESIN
  • achieve accuracy, scalability and low overhead all together

‣ Insight 1
  • break mixed detecting and pinpointing
Detecting memory leaks with RESIN

▪ Our response is RESIN
  • achieve accuracy, scalability and low overhead all together

▪ Insight 1
  • break mixed detecting and pinpointing
  • decompose detection to multi-stages
Detecting memory leaks with RESIN

- Our response is RESIN
  - achieve accuracy, scalability and low overhead all together

- Insight 1
  - break mixed detecting and pinpointing
  - decompose detection to multi-stages
Detecting memory leaks with RESIN

- Our response is RESIN
  - achieve accuracy, scalability and low overhead all together

- Insight 1
  - break mixed detecting and pinpointing
  - decompose detection to multi-stages
Detecting memory leaks with RESIN

- Our response is RESIN
  - achieve accuracy, scalability and low overhead all together
Detecting memory leaks with RESIN

- Our response is RESIN
  - achieve accuracy, scalability and low overhead all together

- Insight 2
  - a centralized approach for all components
  - leverage power of scale to improve accuracy
RESIN overview

Detection:
- Bucket-based pivot analysis
- Individual proc. analysis
- Leaking alert

Diagnosis:
- Pattern-based snapshot collector
- Reference builder
- Snapshot analysis

Mitigation:
- Impact-minimized decision maker
- Diagnosis report
Outline

1. Motivation
2. Two-stage leak detection
3. Trace collection and diagnosis of detected leaks
4. In-production evaluation
Outline

1. Motivation

2. Two-stage leak detection
   1. which component is leaking cluster-wide?
   2. on which hosts that component is leaking?

3. Trace collection and diagnosis of detected leaks

4. In-production evaluation
Detect leaking component

• A straightforward solution:
  • run anomaly detection on time-series data of memory usage for each host

• What are the challenges?
Challenges on detecting memory leaks in cloud

- **Challenge 1:** noisy signals from environment
  - many different workloads in the cloud with dynamic characteristics
  - false positives easily incur
Challenges on detecting memory leaks in cloud

- **Challenge 1**: noisy signals from environment
  - many different workloads in the cloud with dynamic characteristics
  - detection false positives easily incur

- **Challenge 2**: slow leaks in long-running services
  - memory leaks often last over days or weeks
  - need to capture gradual changes meanwhile alerting in time

- **Challenge 3**: large profiling data volumes
  - need to analyze >10 TB memory usage data daily
Solution: bucket-based pivot analysis

- Each bucket is a collection of hosts with memory usage in a same range
  - this bucketization is done per component
  - e.g., 50MB-bucket includes hosts running firewall services with usage 50MB-100MB

- **Insight:** monitor trend of bucket size instead of individual component usage
  - robust to tolerate noises due to workload effect (challenge 1)
  - scalable to large clusters with massive hosts (challenge 3)
Solution: bucket-based pivot analysis

- Summaries from recent time-series data
  - able to detect slow leaks for weeks (challenge 2)
Solution: bucket-based pivot analysis

- Run anomaly detection against time series of bucket size
  - build normal distribution model from baseline range (2/3 portion)

![Diagram showing bucket-based pivot analysis]

- Graph showing per-bucket normal distribution model with mean $\mu$ and standard deviation $3\sigma$.
Solution: bucket-based pivot analysis

- Run anomaly detection against time series of bucket size
  - use the remaining data points as the test (1/3 portion)

<table>
<thead>
<tr>
<th></th>
<th>50 MB</th>
<th>100 MB</th>
<th>200 MB</th>
<th>2 GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t_1)</td>
<td>![image1]</td>
<td>![image2]</td>
<td>![image3]</td>
<td>![image4]</td>
</tr>
<tr>
<td>(t_2)</td>
<td>![image5]</td>
<td>![image6]</td>
<td>![image7]</td>
<td>![image8]</td>
</tr>
<tr>
<td>(t_3)</td>
<td>![image9]</td>
<td>![image10]</td>
<td>![image11]</td>
<td>![image12]</td>
</tr>
</tbody>
</table>

![chart1] per-bucket normal distribution model
Solution: bucket-based pivot analysis

- Run anomaly detection against time series of bucket size
  - data points that exceed the $\mu + 3\sigma$ of the baseline data are anomaly

![Diagram showing anomaly detection](image)

- [1] mean and standard deviation of the distribution

outlier means the component is leaking!

per-bucket normal distribution model
Localizing leaking process

- Now we know which component is leaking
- Next question is, how to find on which host the component is leaking?
- Solution: suspicious window analysis
  - input: memory usage time-series data on each host
  - output: a list of suspected hosts with
    - leaking time windows
    - severity scores
- See algorithm details in our paper
1. Motivation

2. Two-stage leak detection

3. Trace collection and diagnosis of detected leaks
   1. what profiling traces are useful for diagnosis?
   2. what is the key challenge to collect traces?
   3. how to analyze the collected traces?

4. In-production evaluation
Profiling trace: heap snapshots

- RESIN diagnoses leaks by capturing heap snapshot traces
  - wait for leak allocation happens again to trigger completion
  - differentiate snapshots before and after memory leak allocation

```
<table>
<thead>
<tr>
<th>Alloc Addr</th>
<th>Stack Id</th>
<th>Size</th>
<th>RefCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80000</td>
<td>1</td>
<td>64</td>
<td>2</td>
</tr>
<tr>
<td>0x90000</td>
<td>1</td>
<td>128</td>
<td>1</td>
</tr>
<tr>
<td>0xf0000</td>
<td>2</td>
<td>32</td>
<td>2</td>
</tr>
</tbody>
</table>
```
Challenge: decide trace collection timing

- Snapshot differencing requires accurate triggers for leak
- Strawman solution: setting threshold on memory usage difference
  - likely complete the tracing prematurely due to a memory usage spike
  - result in failure to capture the buggy allocation
Solution: collection based on growth pattern

- RESIN collect traces with pattern-based strategy
  - leaks usually exhibits consistent patterns across time
  - we classify the pattern of leak from historical data using simple linear regression
  - RESIN trigger completion based on collection strategy pre-defined for each pattern
Analyzing trace for diagnosis

1. Differentiate allocations between snapshots before and after leak
   • returns a list of allocations containing leaky allocation

<table>
<thead>
<tr>
<th>Alloc Addr</th>
<th>Stack Id</th>
<th>Size</th>
<th>RefCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80000</td>
<td>1</td>
<td>64</td>
<td>2</td>
</tr>
<tr>
<td>0xf0000</td>
<td>3</td>
<td>224</td>
<td>2</td>
</tr>
<tr>
<td>0xf0100</td>
<td>4</td>
<td>2560</td>
<td>2</td>
</tr>
</tbody>
</table>

\[ \text{snapshot}_2 \]

<table>
<thead>
<tr>
<th>Alloc Addr</th>
<th>Stack Id</th>
<th>Size</th>
<th>RefCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80000</td>
<td>1</td>
<td>64</td>
<td>2</td>
</tr>
<tr>
<td>0x90000</td>
<td>1</td>
<td>128</td>
<td>1</td>
</tr>
<tr>
<td>0xb0000</td>
<td>2</td>
<td>256</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ \text{snapshot}_1 \]

\[ \text{outstanding allocations} \]
Analyzing trace for diagnosis

2. Sort the allocation list by size

- prioritize allocations whose memory usage is closer to estimated size
- challenge: the list still contains some noisy allocations, how to filter them?

<table>
<thead>
<tr>
<th>Alloc Addr</th>
<th>Stack Id</th>
<th>Size</th>
<th>RefCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xb0000</td>
<td>2</td>
<td>256</td>
<td>1</td>
</tr>
<tr>
<td>0xf0000</td>
<td>3</td>
<td>224</td>
<td>2</td>
</tr>
<tr>
<td>0xf0100</td>
<td>4</td>
<td>2560</td>
<td>2</td>
</tr>
</tbody>
</table>

outstanding allocations

<table>
<thead>
<tr>
<th>Alloc Addr</th>
<th>Stack Id</th>
<th>Size</th>
<th>RefCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xf0100</td>
<td>4</td>
<td>2560</td>
<td>2</td>
</tr>
<tr>
<td>0xb0000</td>
<td>2</td>
<td>256</td>
<td>1</td>
</tr>
<tr>
<td>0xf0000</td>
<td>3</td>
<td>224</td>
<td>2</td>
</tr>
</tbody>
</table>

outstanding allocations (sorted)
Solution: references from non-leaking hosts

- Collect reference snapshots to filter noises
  - fingerprint leaking processes and find its non-leaking hosts as references
    - (cluster_id, OS version, service version, date)
  - collect heap snapshots to retrieve stack traces from normal workloads

leaking hosts (high severity score)  non-leaking hosts (low severity score)
Analyzing trace for diagnosis

3. Filter likely noisy allocations
   - remove allocations larger than estimated size or from reference snapshots
   - output diagnosed stack trace as result

<table>
<thead>
<tr>
<th>Alloc Addr</th>
<th>Stack Id</th>
<th>Size</th>
<th>RefCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xf0100</td>
<td>4</td>
<td>2560</td>
<td>2</td>
</tr>
<tr>
<td>0xb0000</td>
<td>2</td>
<td>256</td>
<td>1</td>
</tr>
<tr>
<td>0xf0000</td>
<td>3</td>
<td>224</td>
<td>2</td>
</tr>
</tbody>
</table>

outstanding allocations (sorted)
Analyzing trace for diagnosis

3. Filter likely noisy allocations
   - remove allocations larger than estimated size or from reference snapshots
   - output diagnosed stack trace as result

<table>
<thead>
<tr>
<th>Alloc Addr</th>
<th>Stack Id</th>
<th>Size</th>
<th>RefCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xf0100</td>
<td>4</td>
<td>2560</td>
<td>2</td>
</tr>
<tr>
<td>0xb0000</td>
<td>2</td>
<td>256</td>
<td>1</td>
</tr>
<tr>
<td>0xf0000</td>
<td>0</td>
<td>224</td>
<td>2</td>
</tr>
</tbody>
</table>

top in reference snapshot allocation lists

outstanding allocations (sorted)
Analyzing trace for diagnosis

3. Filter likely noisy allocations
   - remove allocations larger than estimated size or from reference snapshots
   - output diagnosed stack trace as result

<table>
<thead>
<tr>
<th>Alloc Addr</th>
<th>Stack Id</th>
<th>Size</th>
<th>RefCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xf0100</td>
<td>4</td>
<td>2560</td>
<td>2</td>
</tr>
<tr>
<td>0xb0000</td>
<td>2</td>
<td>256</td>
<td>1</td>
</tr>
<tr>
<td>0xf0000</td>
<td>0</td>
<td>224</td>
<td>2</td>
</tr>
</tbody>
</table>

stack trace
- ConfManager::ApplyUnlocked
- Conf::Apply
- FirewallRuleInfo::Create
- Firewall::AddRule
Outline

1. Motivation
2. Two-stage leak detection
3. Trace collection and diagnosis of detected leaks
4. In-production evaluation
RESIN deployment status and scale

- Running in Azure production since late 2018
  - cover millions of hosts
  - detect leaks for 600+ host processes
  - detect leaks for 800+ kernel pool tags
  - the detection engine analyzes more than 10 TB memory usage data daily
  - the diagnosis module collects 56 traces on average daily
In-production evaluation

- Our evaluation aims to answer questions:
  - (1) how effective is RESIN in addressing memory leaks in Azure?
  - (2) how accurate is the detection?
  - (3) can RESIN help developers diagnose leaks?
  - (4) what is the overhead of trace collection?
  - ...
Evaluation setting

- We collected data from July 2020 to August 2021
  - the detection engine reports 564 tickets in total
  - developers explicitly resolved 291 (52%) tickets
How effective is RESIN?

- VM reboots reduced by 41x
  - average number of reboots per 100,000 hosts per day due to low memory

- VM allocation errors reduced by 10x
  - ratio of erroneous VM allocation requests due to low memory

* data is normalized
How accurate is the detection?

- 7 false positives out of 291 resolved cases
  - caused by new software features or configuration changes

- 4 false negatives not covered in RESIN’s reports among 14 months
  - the leak bugs were captured by developers before causing noticeable impact
Can RESIN help developers diagnose leaks?

- RESIN collects traces and generates reports for 157 cases
  - we followed debugging 14 issues to validate diagnosis usefulness
  - directly pinpoint for 11 out of 14 cases
  - save developers days to weeks on diagnosis workloads
Diagnosis for the earlier firewall example
Diagnosis for the earlier firewall example

1. Detection alerts on firewall service
Diagnosis for the earlier firewall example

1. Detection alerts on firewall service
2. Collect trace on firewall service
Diagnosis for the earlier firewall example

1 detection alerts on firewall service
2 collect trace on firewall service
3 diagnosis report
   - ConfManager::ApplyUnlocked
   - Conf::Apply
   - FirewallRuleInfo::Create
   - Firewall::AddRule
Diagnosis for the earlier firewall example

1. detection alerts on firewall service
2. collect trace on firewall service
3. diagnosis report
   - ConfManager::ApplyUnlocked
   - Conf::Apply
   - FirewallRuleInfo::Create
   - Firewall::AddRule
4. • bug not found in AddRule function
   • but triggers developers to check rules
Diagnosis for the earlier firewall example

1. Detection alerts on firewall service

2. Collect trace on firewall service

3. Diagnosis report
   - ConfManager::ApplyUnlocked
   - Conf::Apply
   - FirewallRuleInfo::Create
   - Firewall::AddRule

4. Bug not found in AddRule function
   - but triggers developers to check rules

5. Found not released rules from config agent crashes

<table>
<thead>
<tr>
<th>ID</th>
<th>PERM</th>
<th>PRO</th>
<th>SRC</th>
<th>DEST</th>
<th>OWNER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>✓</td>
<td>TCP</td>
<td>ANY</td>
<td>192.168.1.21</td>
<td>CA</td>
</tr>
<tr>
<td>2</td>
<td>✓</td>
<td>TCP</td>
<td>ANY</td>
<td>192.168.1.22</td>
<td>CA</td>
</tr>
</tbody>
</table>
Diagnosis for the earlier firewall example

1. Detection alerts on firewall service
2. Collect trace on firewall service
3. Diagnosis report
   - ConfManager::ApplyUnlocked
   - Conf::Apply
   - FirewallRuleInfo::Create
   - Firewall::AddRule
4. Bug not found in AddRule function
   - but triggers developers to check rules
5. Found not released rules from config agent crashes
6. Root cause pinpointed in config agent and fixed

<table>
<thead>
<tr>
<th>ID</th>
<th>PERM</th>
<th>PRO</th>
<th>SRC</th>
<th>DEST</th>
<th>OWNER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>✓</td>
<td>TCP</td>
<td>ANY</td>
<td>192.168.1.21</td>
<td>CA</td>
</tr>
<tr>
<td>2</td>
<td>✓</td>
<td>TCP</td>
<td>ANY</td>
<td>192.168.1.22</td>
<td>CA</td>
</tr>
</tbody>
</table>
What is the overhead of trace collection?

**Affected hosts:** < 0.1% of all nodes

**Affected sessions:** < 9% on affected hosts

**Memory:** + 1.93 MB

**CPU:** a spike lasting for seconds

**End-to-end latency:** +1 second (median)
Conclusion

‣ Addressing memory leaks in cloud infrastructure is challenging
‣ RESIN, an end-to-end memory leak solution in production
  • divide-and-conquer to decompose the problem
  • multi-level solution with novel algorithms
‣ Running in Azure for more than 3 years
  • low-memory-induced VM reboots reduced 41×
  • new VM allocation errors reduced 10×
Backup slides
Decision tree based mitigation

- Goal: mitigate the memory leaks while minimizing the user impact

- customized?
  - Yes → Perform customized mitigations (e.g. unload driver)
  - No → in allow-list?
- in allow-list?
  - Yes → Process restart
  - No → empty host?
- empty host?
  - Yes → not working?
  - No → live VM migrate

- Host reboot
- Kernel soft reboot
Mitigation duration

- **①** testing on small scale (0-7 day)
- **②** mitigate on large scale (7-30 day)
- **③** mitigation continues while the fix rolling out (30-130 day)
- **④** retry on few remaining hosts (130-145 day)