Present but unreachable
Reducing persistent latent secrets in HotSpot JVM

Adam Pridgen\textsuperscript{1} Simson L. Garfinkel\textsuperscript{2} Dan S. Wallach\textsuperscript{1}

\textsuperscript{1}Rice University, Houston, TX, USA
\textsuperscript{2}George Mason University, Fairfax, VA, USA

Hawaii International Conference on System Sciences, 2017
Java runtime uses automatic memory management
Developers no longer control data lifetimes
Sensitive data cannot be explicitly destroyed
Multiple copies can be created
Research Questions

- How many secrets are retained?
- Should we be concerned?
- Can we fix the problem (without vendor intervention)?
- Is our solution useful?
Talk Overview

1. Introduction
2. Background
3. Problem
4. Approach
5. Results
6. Conclusions
7. References
Related Work

- Viega explains the insecurity of managed runtimes [1]
- Chow et al. solve secure deallocation on Unix [2, 3]
- CleanOS: Objects encrypted using a shared key [4]
- Anikeev et al. focuses on Android’s collector [5]
- Li shows RSA keys are retrievable in Python [6]
Generational GC Heap Overview

- Tracing GC: Looking for *live* objects from a set of roots
- Heap engineered for expected object life-time
- Partitions managed to meet performance goals

**Figure:** Typical generational heap layout.
low- or out-of-memory events trigger collection

GC vs. Full GC

- Young generation: copy or mark-sweep-copy
- Tenure generation: mark-sweep-compact

Figure: Typical generational heap layout.
Promote objects from one heap to the next one
- **Eden Space** → **Survivor Space**
- **Survivor Space** → **Tenure Space**

**Figure:** Typical generational heap layout.
Other Factors Affecting Measurement

- GC algorithms and various collection conditions
- Internal JVM memory management system
- Interactions between JVM internals and program data
- Java Native Interface (not evaluated)
Unmanaged Data Lifetime Overview

Figure: Example data lifetime in unmanaged memory.
Figure: Example data lifetime in managed memory.
Problem

Why is data being retained?

Figure: `String[2]` on the heap.
Problem

Why is data being retained? (2)

Figure: String[0] is reassigned but the old value remains.
Measuring Latent Secrets: Methodology

- Quantify data retention using TLS Keys
  - Vary memory pressure
  - Use well-known software examples
  - Vary heap size 512MiB-16GiB
- Modify HotSpot JVM to perform sanitization
- Re-evaluate data retention
- Measure the performance impacts
## Measuring Latent Secrets: TLS Clients

<table>
<thead>
<tr>
<th>Basic TLS Client</th>
<th>Apache HTTP TLS Client</th>
<th>Apache HTTP TLS Client with BouncyCastle</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Wrap TLS socket</td>
<td>1. Library creates socket</td>
<td>1. Library creates socket</td>
</tr>
</tbody>
</table>
## Measuring Latent Secrets: Memory Pressure

<table>
<thead>
<tr>
<th>High Memory Pressure</th>
<th>Low Memory Pressure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. High Memory Contention</td>
<td></td>
</tr>
<tr>
<td>2. Consume up to 80%</td>
<td></td>
</tr>
<tr>
<td>3. 192 requests per running session (thread)</td>
<td></td>
</tr>
<tr>
<td>1. Low Memory Contention</td>
<td></td>
</tr>
<tr>
<td>2. Consume up to 20%</td>
<td></td>
</tr>
<tr>
<td>3. 48 requests per running session (thread)</td>
<td></td>
</tr>
</tbody>
</table>
Approach

Measuring Latent Secrets: Test Bench

Figure: Overview of experiment and captured data.
Approach

Measuring Latent Secrets: Data Processing

- Dump virtual machine system memory (e.g. RAM)
- Grep RAM for captured TLS key material
- Reconstruct the JVM process memory
- Grep *process memory* for TLS key material
- Reorder TLS sessions and count keys
Failed Approach

- Modify the Java Cryptography TLS Routines
- Sanitize *out-of-scope* references
- Explicit clean-up when sockets close or shutdown
Reducing Latent Secrets

Approach

Successful Implementation

- Modify the JVM and GC algorithms
- Zero unused space after each collection
- Zero internally managed memory when deallocated
Reducing Latent Secrets

Successful Implementation
- Modify the JVM and GC algorithms
- Zero unused space after each collection
- Zero internally managed memory when deallocated

Limitations
- Dangling references cannot be collected
- GC must occur on each heap space
- Sanitization may not be timely
Results - SerialGC HMP

Figure: TLS keys recovered from HMP clients.
Figure: TLS keys recovered from LMP clients.
Figure: TLS keys recovered from Socket clients using G1GC.
Benchmarking Results

Figure: Benchmarks show modifications reduced performance.

(a) tradebeans-Day Trader  (b) lusearch-Text Searching
Conclusions

- Quantified data retention in the HotSpot JVM
- Measured these secrets in a general manner
- Developed several strategies to reduce latent secrets
- **Data security** at the expense of **performance**


Conclusions

