Present but unreachable

Reducing persistent latent secrets in HotSpot JVM

Adam Pridgen [1] Simson L. Garfinkel [2] Dan S. Wallach [1]

[1]Rice University, Houston, TX, USA

[2]George Mason University, Fairfax, VA, USA

Hawaii International Conference on System Sciences, 2017

Introduction

- 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

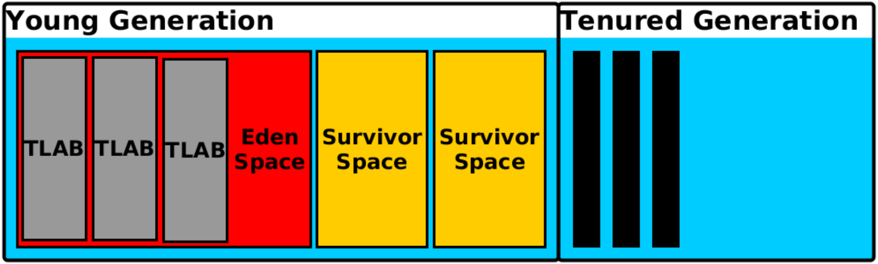


Figure: Typical generational heap layout.

Generational GC Heap Overview

- *low-* or *out-of-memory* events trigger collection
- *GC* vs. *Full GC*
    - **Young generation**: copy or mark-sweep-copy
    - **Tenure generation**: mark-sweep-compact

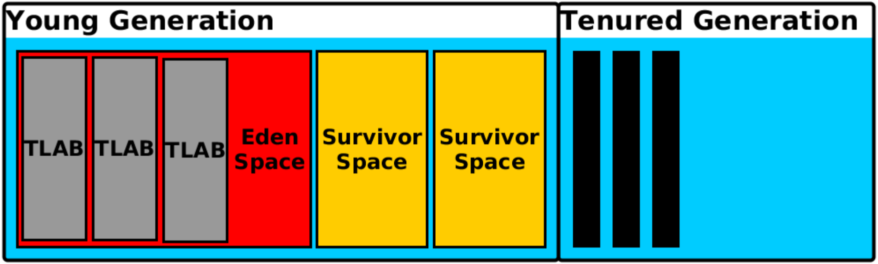


Figure: Typical generational heap layout.

Generational GC Heap Overview

- Promote objects from one heap to the next one
  - **Eden Space → Survivor Space**
  - **Survivor Space → Tenure Space**

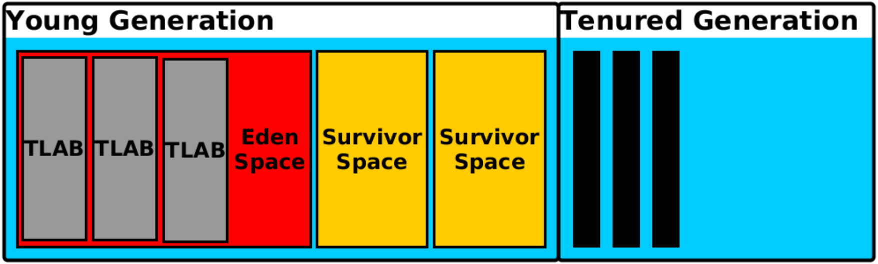


Figure: Typical generational heap layout.

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

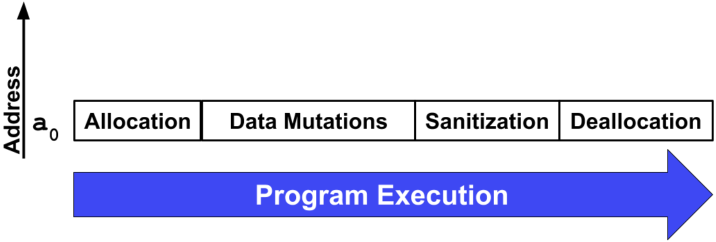


Figure: Example data lifetime in unmanaged memory.

Managed Data Lifetime Overview

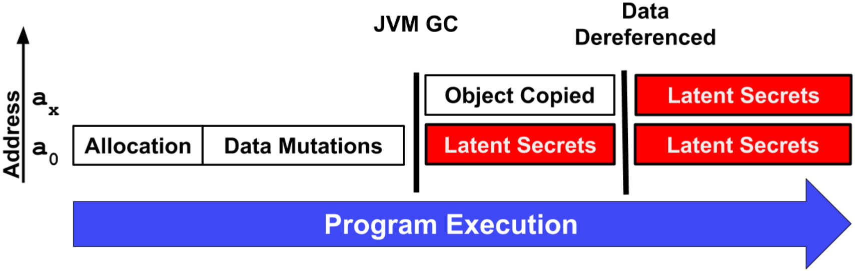
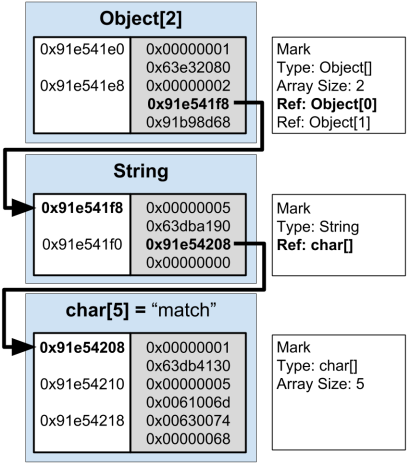


Figure: Example data lifetime in managed memory.

Why is data being retained?



Object[2]

0x91e541e0	0x00000001 0x63e32080
0x91e541e8	0x00000002 **0x91e541f8** 0x91b98d68

Mark
Type: Object[]
Array Size: 2
Ref: Object[0]
Ref: Object[1]

String

0x91e541f8	0x00000005 0x63dba190
0x91e541f0	**0x91e54208** 0x00000000

Mark
Type: String
Ref: char[]

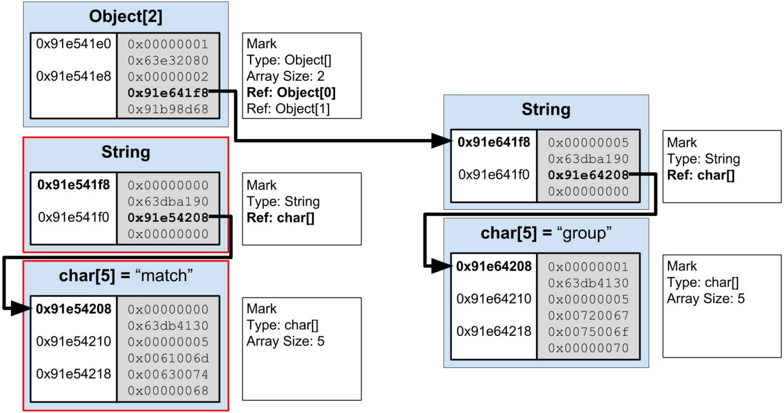
char[5] = “match”

0x91e54208	0x00000001 0x63db4130
0x91e54210	0x00000005 0x0061006d
0x91e54218	0x00630074 0x00000068

Mark
Type: char[]
Array Size: 5

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

Why is data being retained? (2)



Object[2]

0x91e541e0	0x00000001 0x63e32080
0x91e541e8	0x00000002 **0x91e641f8** 0x91b98d68

Mark
Type: Object[]
Array Size: 2
Ref: Object[0]
Ref: Object[1]

String

0x91e541f8	0x00000000 0x63dba190
0x91e541f0	**0x91e54208** 0x00000000

Mark
Type: String
Ref: char[]

char[5] = “match”

0x91e54208	0x00000000 0x63db4130
0x91e54210	0x00000005 0x0061006d
0x91e54218	0x00630074 0x00000068

Mark
Type: char[]
Array Size: 5

String

0x91e641f8	0x00000005 0x63dba190
0x91e641f0	**0x91e64208** 0x00000000

Mark
Type: String
Ref: char[]

char[5] = “group”

0x91e64208	0x00000001 0x63db4130
0x91e64210	0x00000005 0x00720067
0x91e64218	0x0075006f 0x00000070

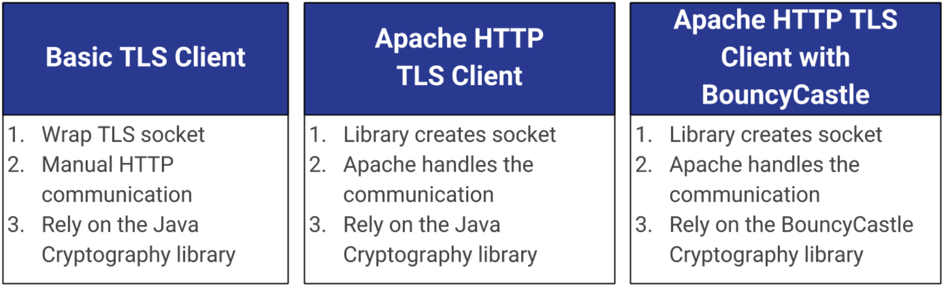
Mark
Type: char[]
Array Size: 5

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



Basic TLS Client	Apache HTTP TLS Client	Apache HTTP TLS Client with BouncyCastle
1. Wrap TLS socket 2. Manual HTTP communication 3. Rely on the Java Cryptography library	1. Library creates socket 2. Apache handles the communication 3. Rely on the Java Cryptography library	1. Library creates socket 2. Apache handles the communication 3. Rely on the BouncyCastle Cryptography library

Measuring Latent Secrets: Memory Pressure

RICE

High Memory Pressure

1. High Memory Contention
2. Consume up to 80%
3. 192 requests per running session (thread)

Low Memory Pressure

1. Low Memory Contention
2. Consume up to 20%
3. 48 requests per running session (thread)

Measuring Latent Secrets: Test Bench

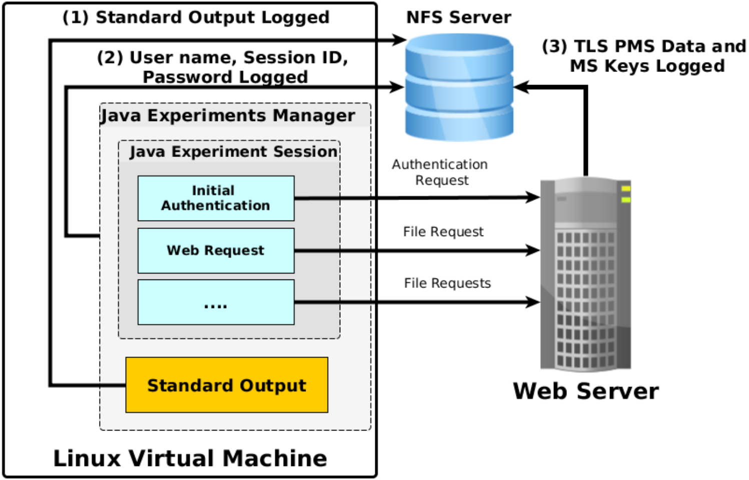


Figure: Overview of experiment and captured data.

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

Reducing Latent Secrets

Failed Approach

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

Reducing Latent Secrets

Successful Implementation

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

Reducing Latent Secrets

RICE

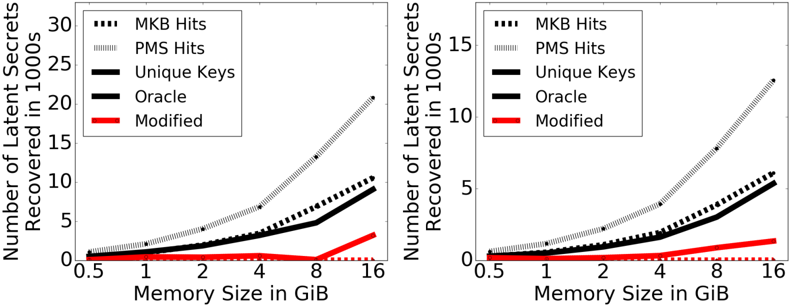
Successful Implementation

- 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



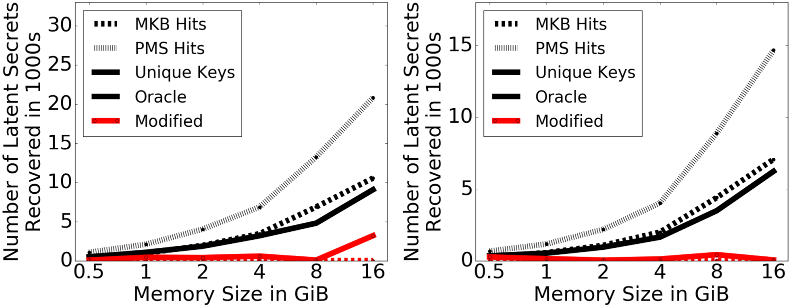
(a) Socket Results

(b) Apache Results

Figure: TLS keys recovered from HMP clients.

Results - SerialGC LMP

RICE

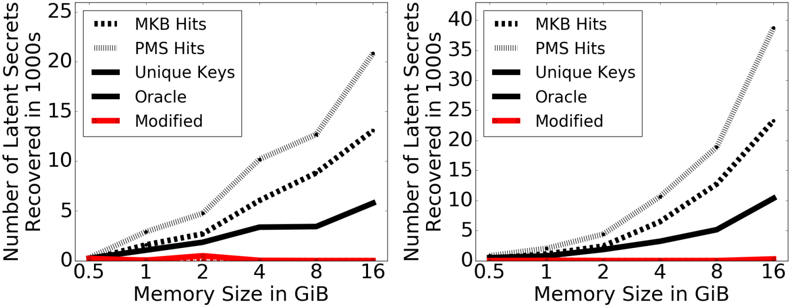


(a) Socket Results

(b) Apache Results

Figure: TLS keys recovered from LMP clients.

Results - G1GC Sockets Client

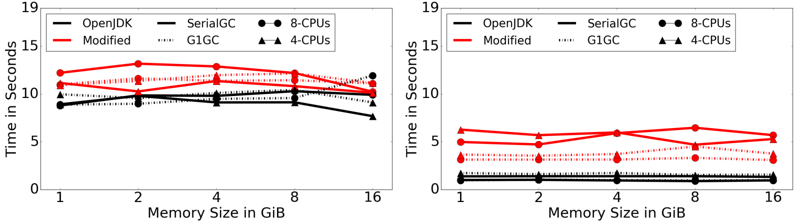


(a) HMP Results

(b) LMP Results

Figure: TLS keys recovered from Socket clients using G1GC.

Benchmarking Results



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

Figure: Benchmarks show modifications reduced performance.

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**

Questions

RICE

[1] J. Viega, “Protecting sensitive data in memory,” 2001.

[2] J. Chow, B. Pfaff, T. Garfinkel, K. Christopher, and M. Rosenblum, “Understanding data lifetime via whole system simulation,” in *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM’04, (Berkeley, CA, USA), pp. 22–22, USENIX Association, 2004.

[3] J. Chow, B. Pfaff, T. Garfinkel, and M. Rosenblum, “Shredding your garbage: Reducing data lifetime through secure deallocation,” in *Proceedings of the 14th Conference on USENIX Security Symposium - Volume 14*, SSYM’05, (Berkeley, CA, USA), pp. 22–22, USENIX Association, 2005.

[4] Y. Tang, P. Ames, S. Bhamidipati, A. Bijlani, R. Geambasu, and N. Sarda, “Cleanos: limiting mobile data exposure with idle eviction,” in *Presented as part of the 10th USENIX*

Symposium on Operating Systems Design and Implementation (OSDI 12), pp. 77–91, 2012.

[5] M. Anikeev, F. C. Freiling, J. Götzfried, and T. Müller, “Secure garbage collection: Preventing malicious data harvesting from deallocated java objects inside the dalvik vm,” *Journal of Information Security and Applications*, vol. 22, pp. 81–86, 2015.

[6] Y. Li, “Where in your ram is “*python san_diego.py*”?,” 2015.