# Present but unreachable

## Reducing persistent latent secrets in HotSpot JVM

***Best Paper***, *Software Technology Track*

**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

RICE

- 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

RICE

# Related Work

RICE

- 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

RICE

- Tracing GC: Looking for *live* objects from a set of roots
- Heap engineered for expected object life-time
- GC promotes objects from one heap to the next one
  - **Eden Space** (short lived) $\rightarrow$ **Survivor Space**
  - **Survivor Space** $\rightarrow$ **Tenure Space** (long lived)

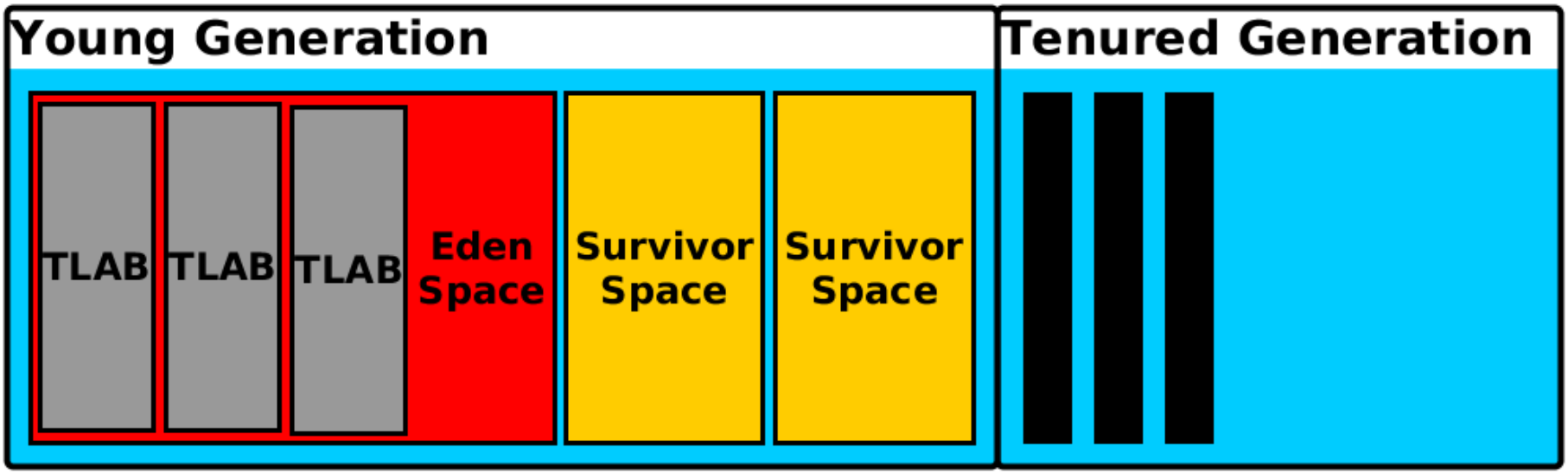



Figure: Typical generational heap layout.

# 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



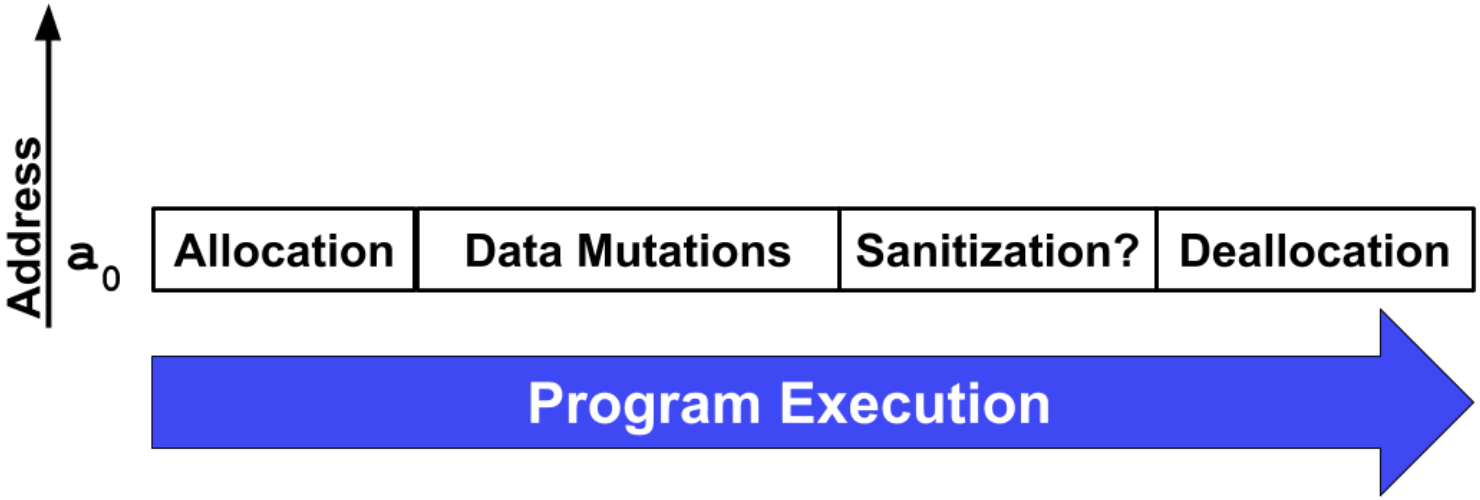



Figure: Example data lifetime in unmanaged memory.

# Managed Data Lifetime Overview

RICE

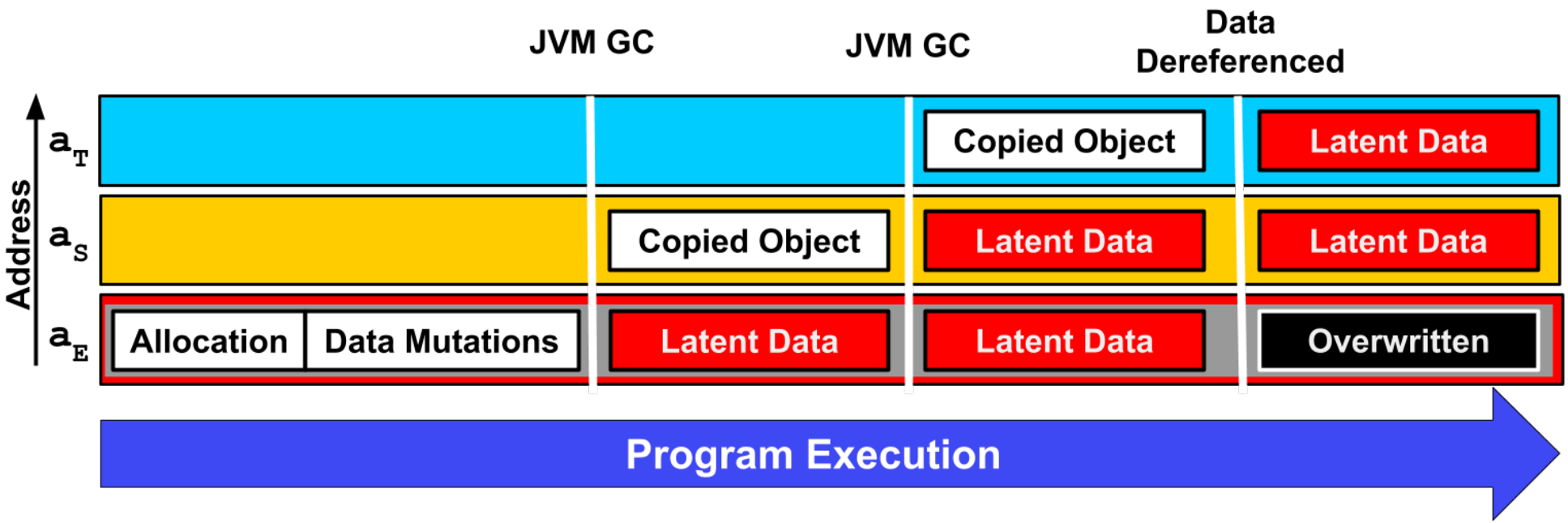



Figure: Example data lifetime in managed memory.

# Why is data being retained?

RICE

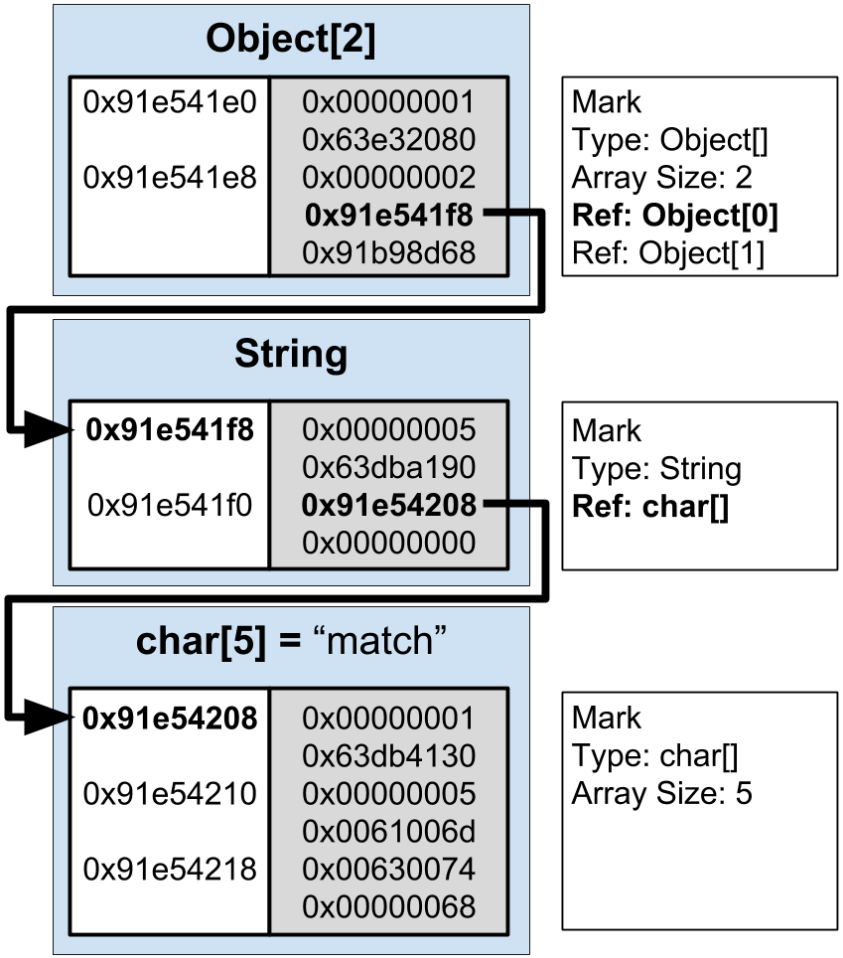



Figure: `String[2]` on the heap.

# Why is data being retained? (2)

RICE

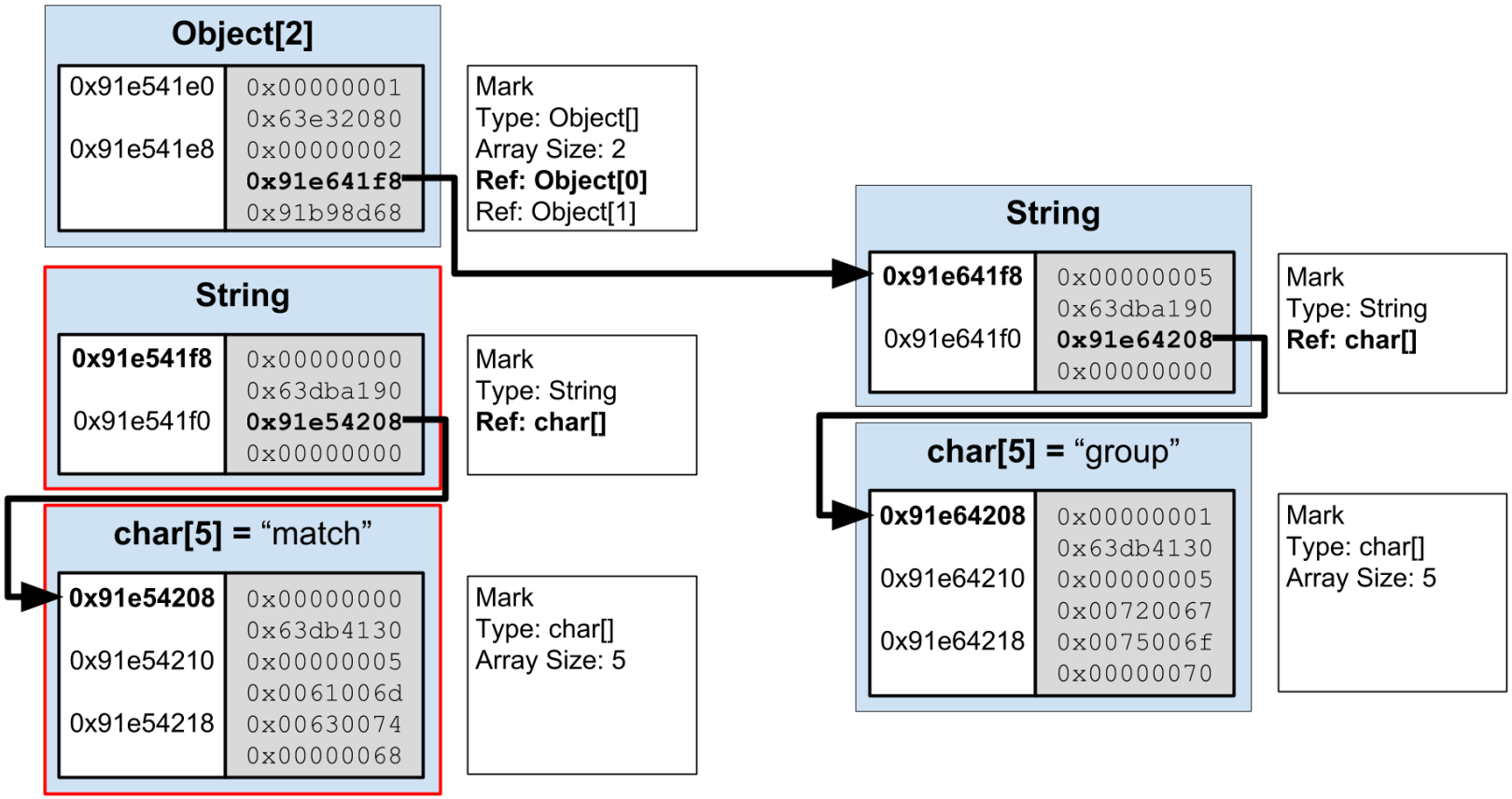



Figure: `String[0]` is reassigned but the old value remains.

# 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

RICE

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

# Measuring Latent Secrets: Memory Pressure

RICE

| High Memory Pressure |
| --- |
| 1. High Memory Contention<br>2. Consume up to 80%<br>3. 192 requests per running session (thread) |

| Low Memory Pressure |
| --- |
| 1. Low Memory Contention<br>2. Consume up to 20%<br>3. 48 requests per running session (thread) |

# Measuring Latent Secrets: Test Bench

RICE

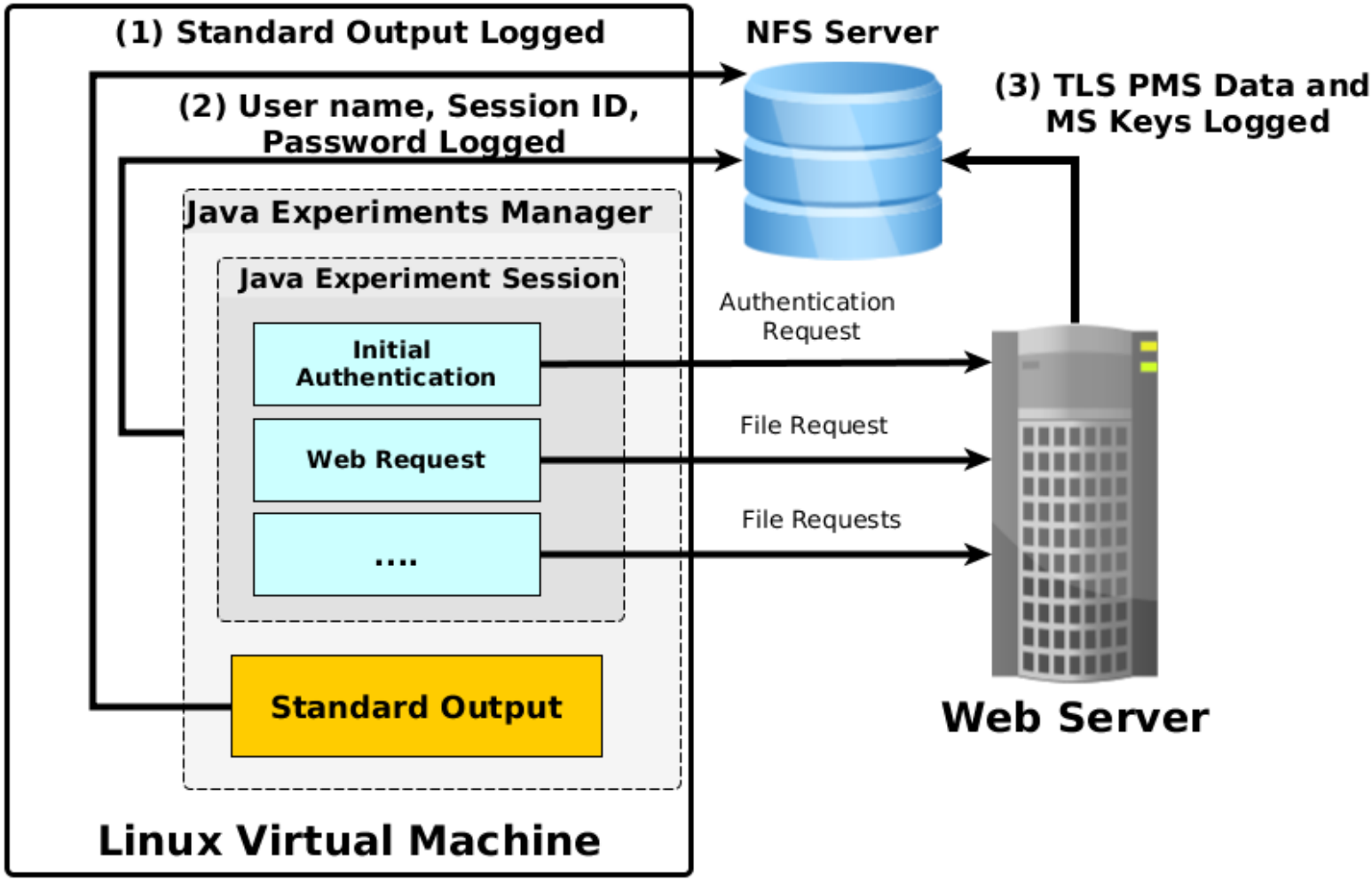



Figure: Overview of experiment and captured data.

# 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

RICE

**Failed Approach**

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

# 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 still prevent object's collection
- GC must occur on each heap space
- Sanitization may not be timely

# Reducing Latent Secrets



## Successful Implementation

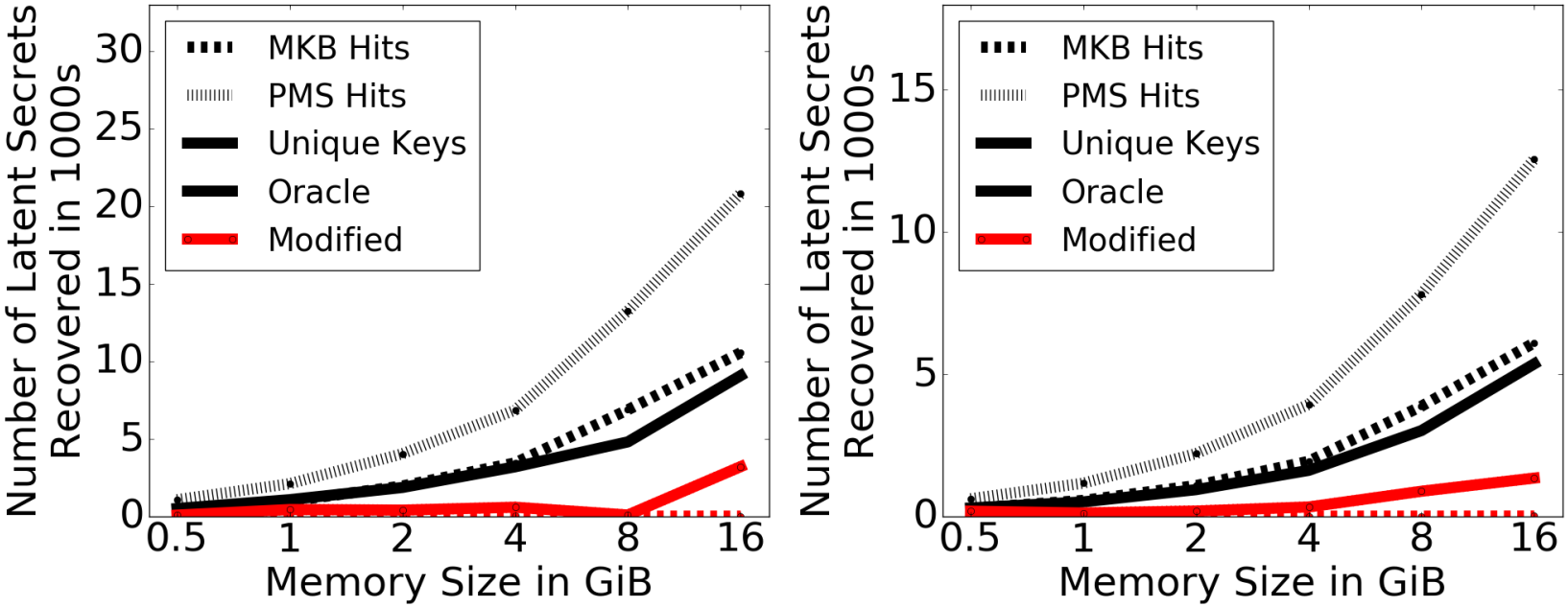
- Modify the JVM and GC algorithms
- Zero unused space after each collection
- Zero internally managed memory when deallocated

## Limitations

- Dangling references still prevent object's collection
- GC must occur on each heap space
- Sanitization may not be timely

# Results - SerialGC HMP

RICE



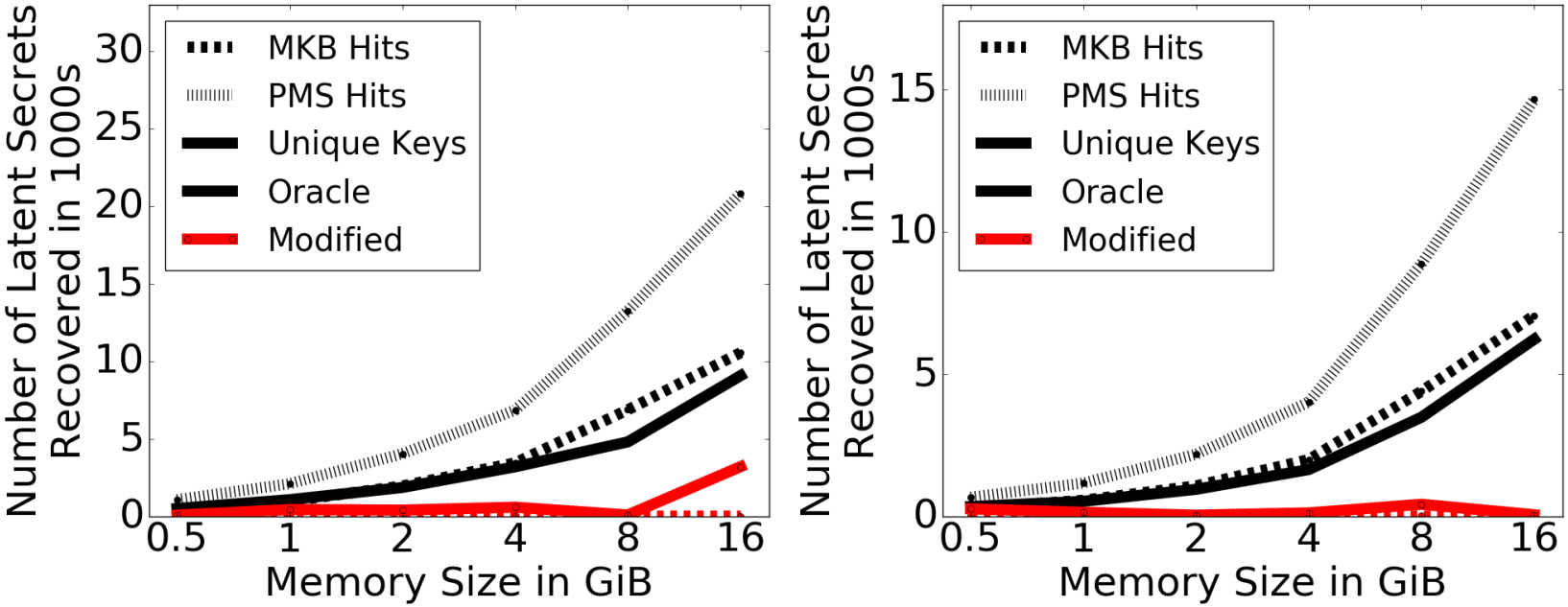

(a) Socket Results

(b) Apache Results

Figure: TLS keys recovered from HMP clients.

# Results - SerialGC LMP

RICE



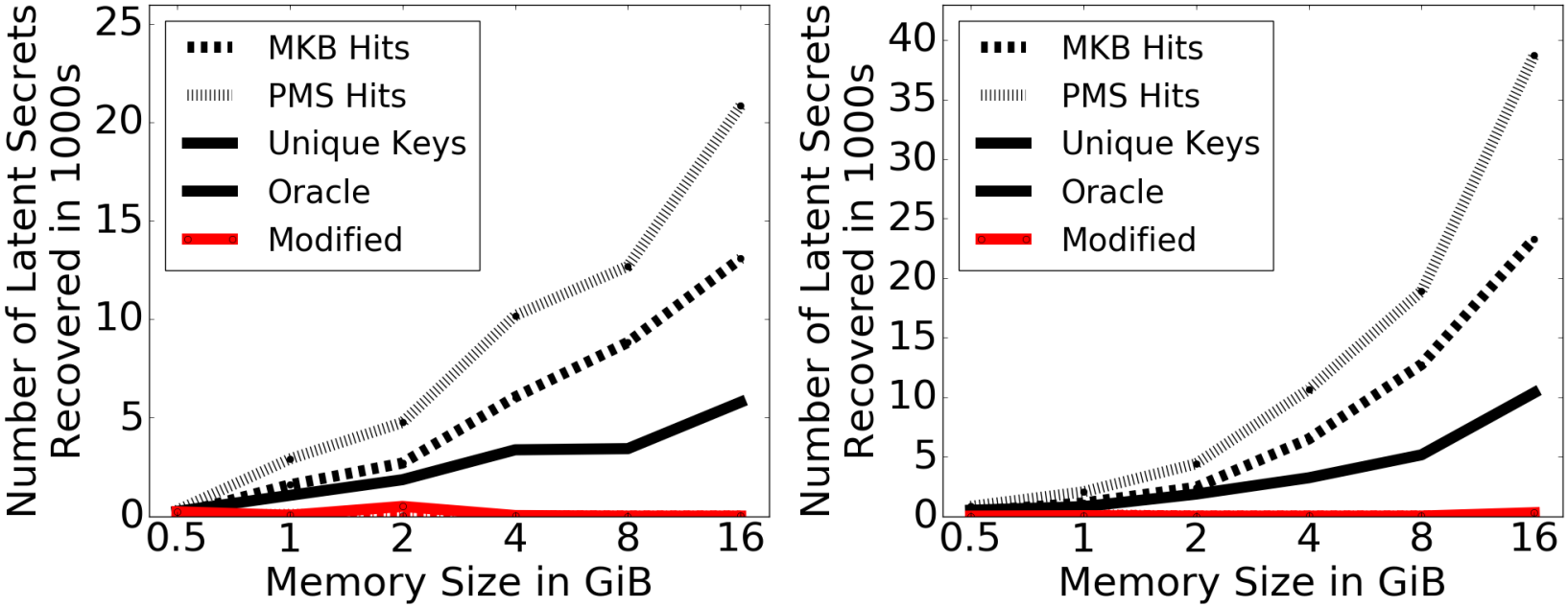

(a) Socket Results

(b) Apache Results

Figure: TLS keys recovered from LMP clients.

# Results - G1GC Sockets Client

RICE



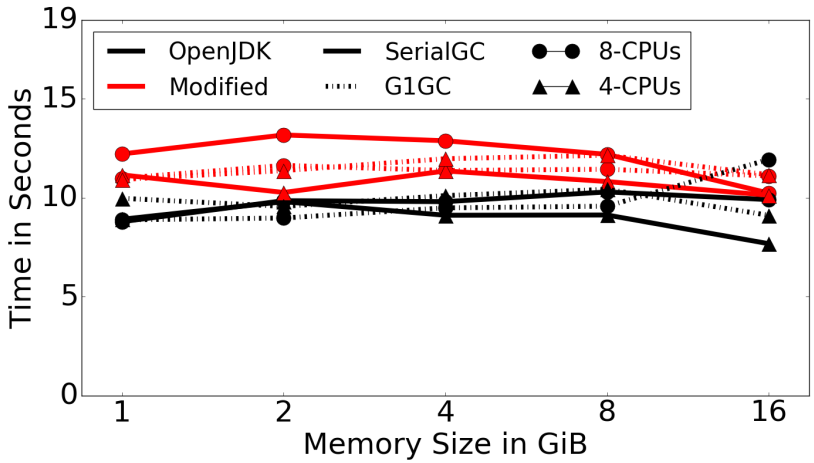

(a) HMP Results

(b) LMP Results

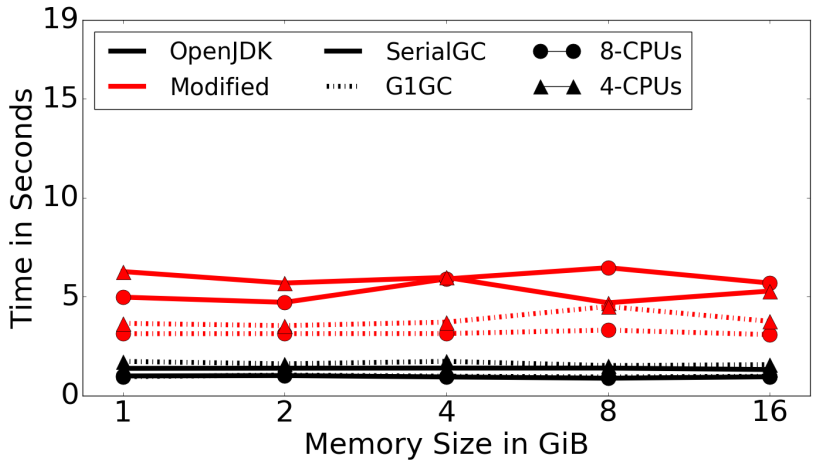
Figure: TLS keys recovered from Socket clients using G1GC.

# Benchmarking Results

RICE




(a) `tradebeans`-Day Trader




(b) `lusearch`-Text Searching

Figure: Benchmarks show modifications reduced performance.

# Conclusions

RICE

- 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, 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, 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)*, 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, 2015.

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