Exploiting Generational Garbage Collection

Using Data Remnants to Improve Memory Analysis and Digital Forensics

Adam Pridgen [1]

[1] Rice University, Houston, TX, USA

January 18, 2017

Outline

Malware Compromises Happen

Flood of Questions

RICE

How did it happen?

Who did it?

Was it targeted?

Why did this happen?

What was lost?

Motivation (1)

RICE

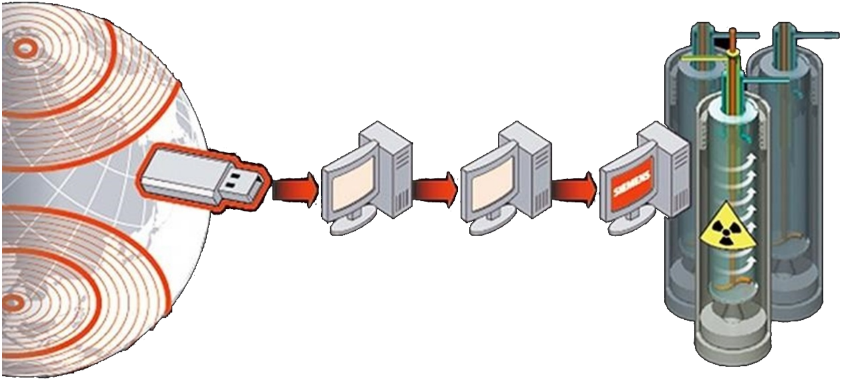


Figure: Stuxnet riding over an airgap into an ICS network [1].

Motivation (2)

RICE

- Threats actors looking to penetrate hard-targets
  - Must research and innovate on existing methods
  - Exploit technology blind spots and implicit trust
  - Looking for *knowledge-gaps*
  - Targetting technologies that are widely deployed
  - Exploring exploitation in all dimensions

Motivation (3)

RICE

- **Managed Runtimes**
  - Widely used but not well understood
  - Runtimes are complex and evolve over time
  - Backwards compatibility retained
  - Widely deployed runs on multiple platforms
  - Updates may not be feasible

Contributions

RICE

- Developed tools for Java class and archive analysis
- Established the feasibility of recovering artifacts
- Created an approach for recovering managed objects
- Developed a prototype targeting the HotSpot JVM

Attacker Techniques

RICE

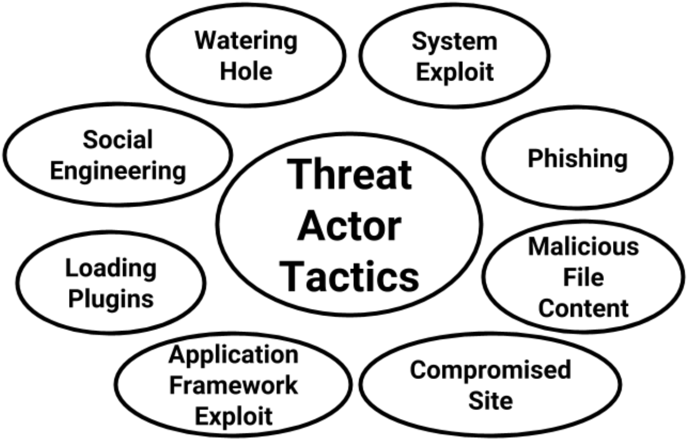


Figure: Overview of attacker tactics.

Documented Java Malware and Attacks

RICE

- Malware and Backdoors
  - **Criminal**: Adwind, JBot, etc. [2, 3, 4]
  - **Espionage**: PackRat and JavaFog [5, 6]
- Threat actors employing Java
  - **Phishing** [7, 4]
  - **Waterholing** [7, 4]
- Common Vulnerabilities and Exposure
  - **Hotspot JVM**: $\approx$ 34 since 2010 [8]
  - **Java and frameworks**: $\approx$ 1510 since 1999

Digital Forensics Overview

RICE

Acquisition	Analysis	Report
Capture Artifacts	**Identify Relevant Details**	**Enable actionable recovery and mitigation steps**
1. System Logs	1. Assemble log events	1. How?
2. Memory Dumps	2. Disk drive modification	2. Why?
3. Network Traffic	3. Memory analysis	3. When?
4. Network Access Logs	4. Network activity	4. Duration?
5. Application Logs	5. Determine movements	5. Mitigation?
6. Disk drive images	6. Enumerate access	6. Litigation or Prosecution?
7. ...	7. ...	7. ...

Related Work: Managed Runtime Analysis

RICE

- Viega explains the insecurity of managed runtimes [9]
- Chow et al. solve secure deallocation on Unix [10, 11]
- CleanOS: Objects encrypted using a shared key [12]
- Anikeev et al. focuses on Android's collector [13]
- Li shows RSA keys are retrievable in Python [14]

Related Work: Memory Analysis

RICE

- Rekall and Volatility analysis frameworks [15, 16]
- Identifying datastructures
    - Lin et al. perform automatic RE [17]
    - Lin et al. use graph-based signatures [18]
    - Dolan et al. focus on kernel structures [19]
- Android memory forensics [20, 21, 22, 23, 24, 25]
- Data carving
    - Richard developed Scalpel File Carver [26]
    - Beverly et al. focus on network packets [27]
    - Hand et al. extract binaries from memory [28]

Contents

RICE

STAAF

STAAF: Scaling Android Application Analysis with a Modular Framework

Ryan W. Smith [1] Adam Pridgen [1]

[1]Praetorian, Austin, TX, USA

[2]Rice University, Houston, TX, USA

Hawaii International Conference on System Sciences, 2012

Problem

- Engineering scalable program analysis
- Off-market Android stores were contained malware
- Android analysis tools fail to scale alone
- Developed an approach pipelining analysis

Results

RICE

- **Similar approach used to measure latent secrets**
- Emphasizes scaling analysis horizontally
- Creates a pipeline for pre- and post- analysis
- Efficiently localizes analysis results in a database

Reversing Java (Malware) with Radare

STAAF: Scaling Android Application Analysis with a Modular Framework

Adam Pridgen [1]

[1]Rice University, Houston, TX, USA

InfoSec Southwest, 2014

Problem

- Malware obfuscation would throw-off analysis
- Tools were built on Java
- Tools overlooked corner cases
- None of the tools allowed low-level manipulation

Java Malware Analysis Overview

- **Eclipse IDE**: development environment for debugging
- **IDA Pro**: marked-up analysis with no low-level access
- **JD GUI**: decompiles code but cannot be corrected
- **Jython**: run Java in Python environment

Results: Radare Extensions

RICE

- Low-level JAR and class file manipulation
- Analysis of class file artifacts
- Inject byte code for runtime analysis
- Rewrite symbolic links for hooking

Contents

RICE

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?

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) → **Survivor Space**
  - **Survivor Space** → **Tenure Space** (long lived)

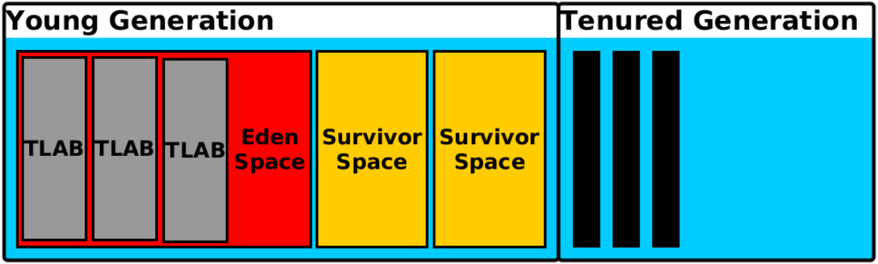


Figure: Typical generational heap layout.

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

RICE

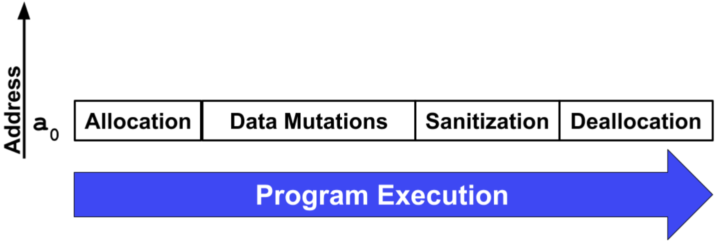


Figure: Example data lifetime in unmanaged memory.

Managed Data Lifetime Overview

RICE

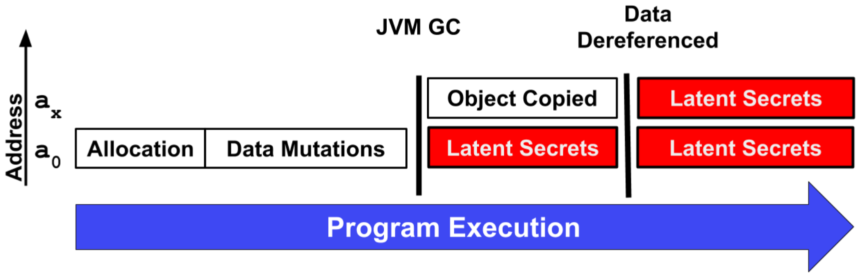


Figure: Example data lifetime in managed memory.

Why is data being retained?

RICE

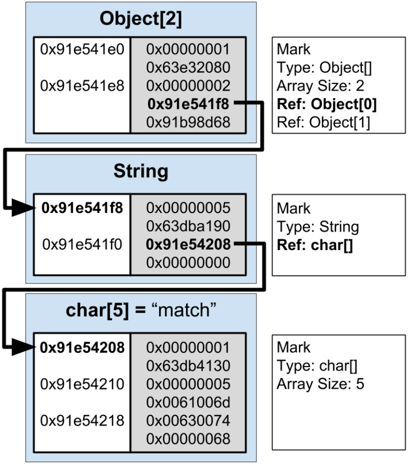


Figure: `String[2]` on the heap.

Why is data being retained? (2)

RICE

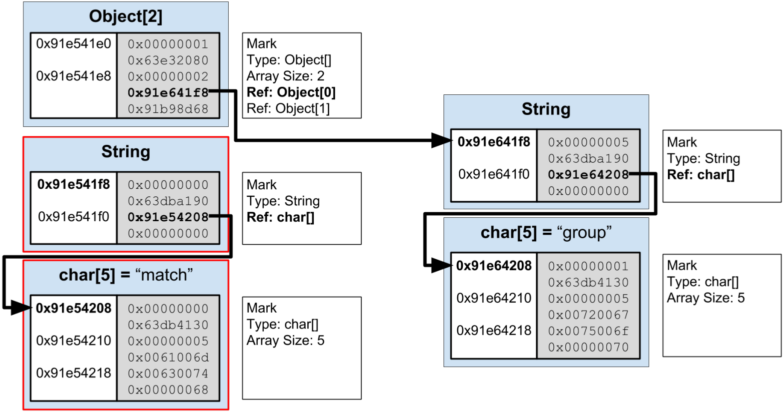


Figure: `String[0]` is reassigned but the old value remains.

Measuring Latent Secrets: Methodology

RICE

- 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 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	Low Memory Pressure
1. High Memory Contention 2. Consume up to 80% 3. 192 requests per running session (thread)	1. Low Memory Contention 2. Consume up to 20% 3. 48 requests per running session (thread)

Measuring Latent Secrets: Test Bench

RICE

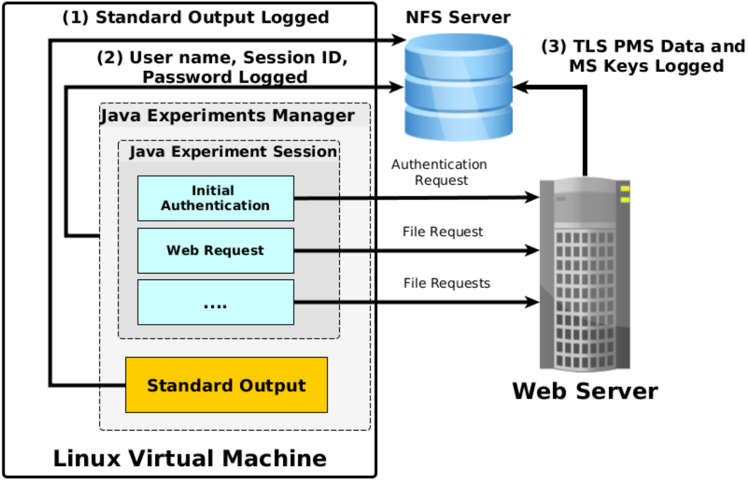


Figure: Overview of experiment and captured data.

Measuring Latent Secrets: Data Processing

RICE

- 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

RICE

Successful Implementation

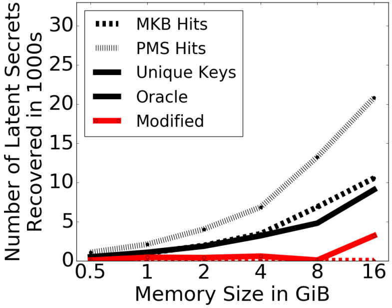
- Modify the JVM and GC algorithms
- Zero unused space after each collection
- Zero internally managed memory when deallocated

Limitations

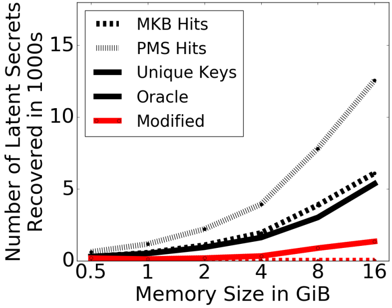
- Dangling references still prevent object's collection
- GC must occur on each heap space
- Sanitization may not be timely

Results - SerialGC HMP

RICE



(a) Socket Results

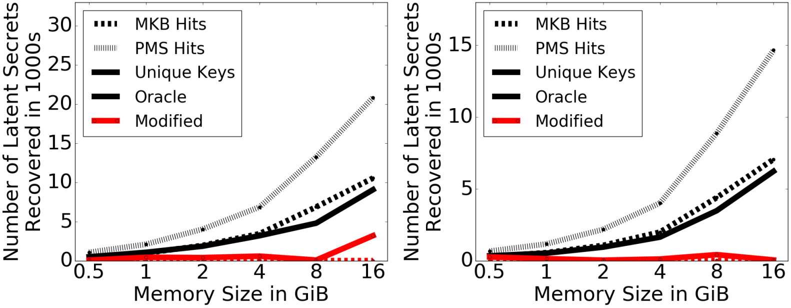


(b) Apache Results

Figure: TLS keys recovered from HMP clients.

Results - SerialGC LMP

RICE

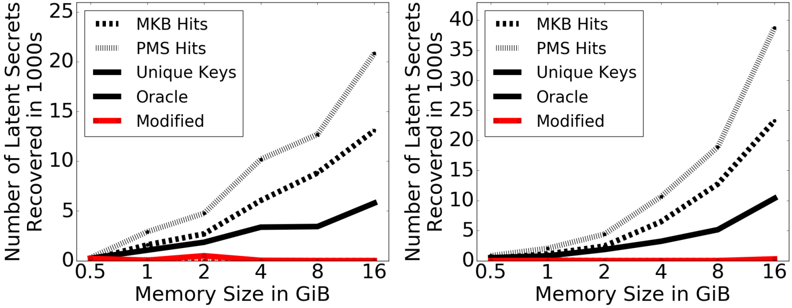


(a) Socket Results

(b) Apache Results

Figure: TLS keys recovered from LMP clients.

Results - G1GC Sockets Client



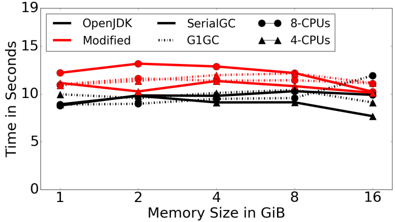
(a) HMP Results

(b) LMP Results

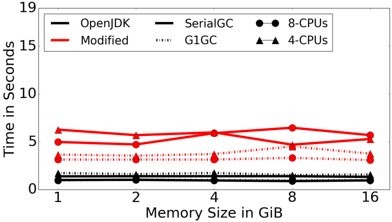
Figure: TLS keys recovered from Socket clients using G1GC.

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**

Contents

RICE

Picking up the trash

Exploiting generational GC for memory analysis

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

[1]Rice University, Houston, TX, USA

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

Digital Forensics Research Workshop, 2017

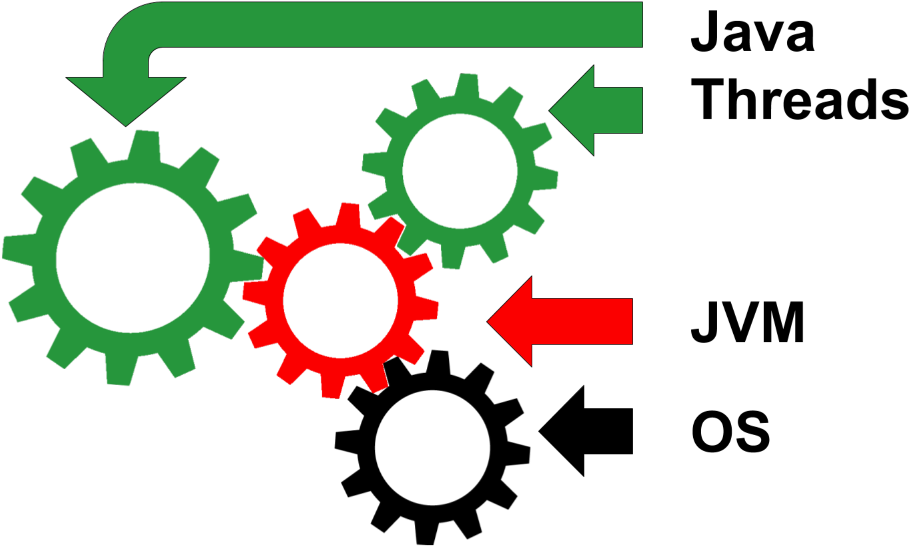
Motivation

RICE

- Authors develop malware for managed runtimes
- Threat actors exploit vulnerable internet applications
- Managed runtimes retain artifacts [29]
- Digital forensics exploit this for evidence recovery

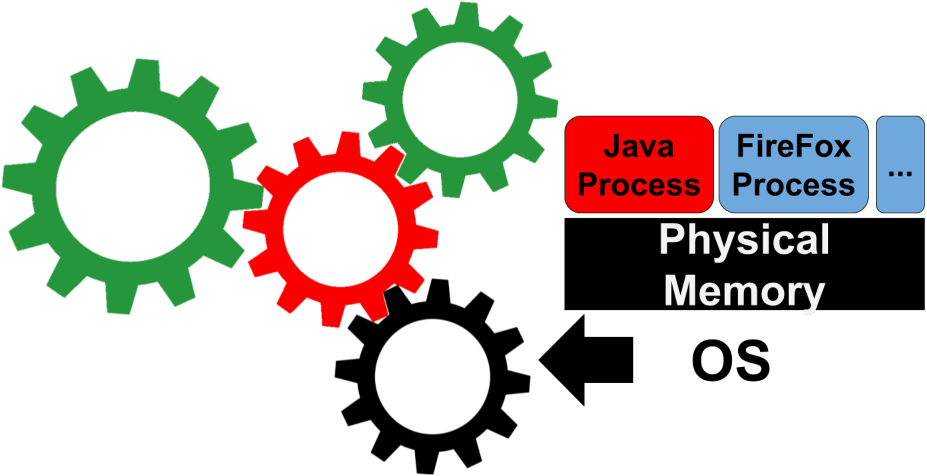
Minding the Gap: Semantic Gaps

RICE



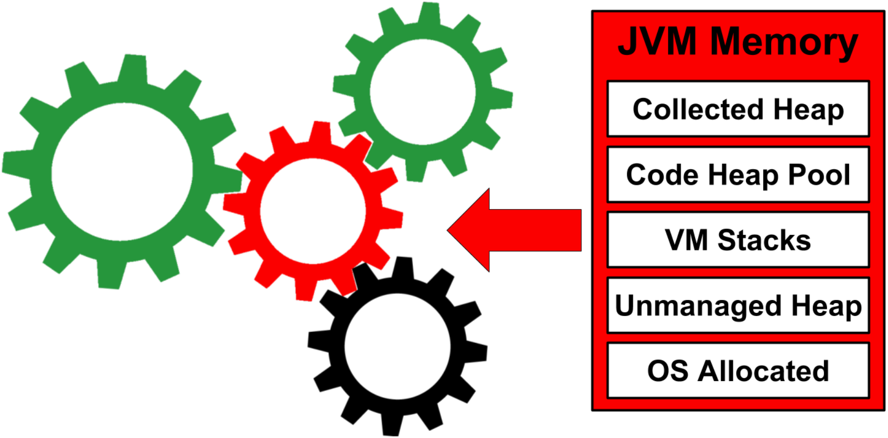
Minding the Gap: OS View

RICE



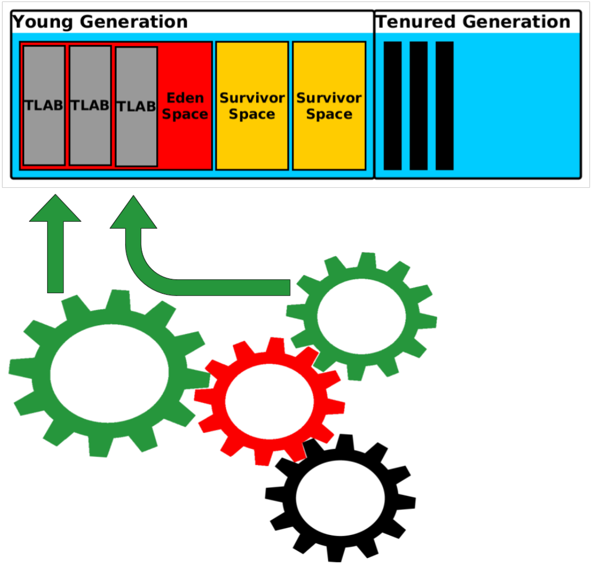
Minding the Gap: JVM Process View

RICE



Minding the Gap: Java Thread View

RICE



Attackers Advantages

RICE

- Tools and endpoint detection lack introspection
- Vulnerabilities exist in the *entire* software stack
- VMs are porous once DMA is achieved
- Complex machines limit *direct* analysis

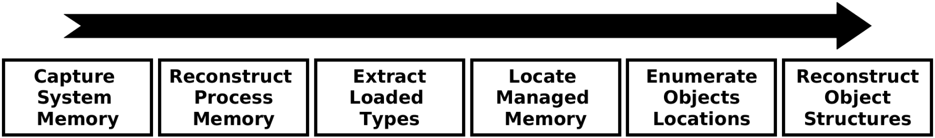
Analyst Advantages

RICE

- Separation of *code* and *data* limits attacker tricks
- Many artifacts for event reconstruction
- *Timelining* (e.g. artifacts ordered by creation)

Recovering OOP Framework

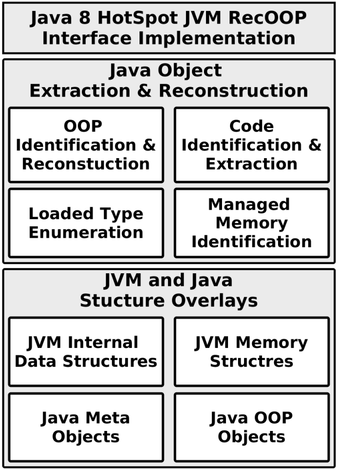
RICE



Recovering OOP Framework (2)

RICE

- Focuses recovery from x86 architecture
- Uses a minimal set of structure overlays
- Compatible with Linux and Windows OS



Memory Capture and Reconstruction

- **Capture:** system memory (e.g. RAM) is dumped
- **Reconstruction:** Target process is found and page frames are reordered

Extract Loaded Types

RICE

- Identify structures revealing loaded types
  - `SystemDictionary`: loaded classes
  - `SymbolTable`: loaded symbols)
  - `StringTable`: constants or long-lived strings
- Mine structures for the loaded data structures

Extract Loaded Types (2)

RICE

- Look for invariant values
- Walk the hash tables
- Use constraints to control recovery
- Event ordering and timelining using addresses

Locate Managed Memory with Logs

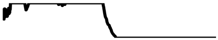
RICE

Table: The regular expression "`space.*used`" used in conjunction with `ffastrings` to determine the eden, survivor, and tenure generation spaces. Note *[...]* signifies omitted message content.

GC Log Message		
Generational Space		Start and End of the Space
eden space	*[...]* used	`[0xa4800000,` *[...]* `0xa4c50000)`
from space	*[...]* used	`[0xa4c50000,` *[...]* `0xa4cd0000)`
to space	*[...]* used	`[0xa4cd0000,` *[...]* `0xa4d50000)`
the space	*[...]* used	`[0xa9d50000,` *[...]* `0xaa800000)`

Locating Managed Memory with Pointers

Table: Java object distribution in managed process memory (e.g. eden, survivor, and tenure spaces).

Address Range	Type Pointers	Unique Pointers	Pointer Occurrences Per Page (Y-axis: 0-64)
`0xa47ff000−0xa4c0f000`	13261	266	
`0xa4c50000−0xa4c92000`	129	28	
`0xa4cd0000−0xa4d50000`	1121	79	
`0xa9d50000−0xaa000000`	28810	661	

Locating Managed Memory with Pointers (2)

Table: Java address distribution in unmanaged process memory.

Address Range	Type Pointers	Unique Pointers	Pointer Occurrences Per Page (Y-axis: 0-64)
0xa32de000-0xa3355000	1353	265	
0xa33ce000-0xa349d000	2735	331	
0xa349e000-0xa34f5000	609	122	
0xa3600000-0xa3692000	362	360	
0xc0001000-0xf7bfe000	11085	1211	

Enumerate and Extract Objects

RICE

- Scan managed heap for known-types
- Parse the object based on the report type
- Lift values for the object's fields

Objects of Immediate Interest

RICE

Java Threads	Sockets	Process Builders
Native Buffers	Streams	Child Processes
JAR Files	Buffers	JAR Entries

Malware Overview

RICE

- Adwind Malware Functionality
  - Obfuscation loads platform independent libraries
  - Modular design with plugins
  - Password protected with hard-coded IP address
  - Tested sample: beaconing, screen capture, etc.

Malware Overview

RICE

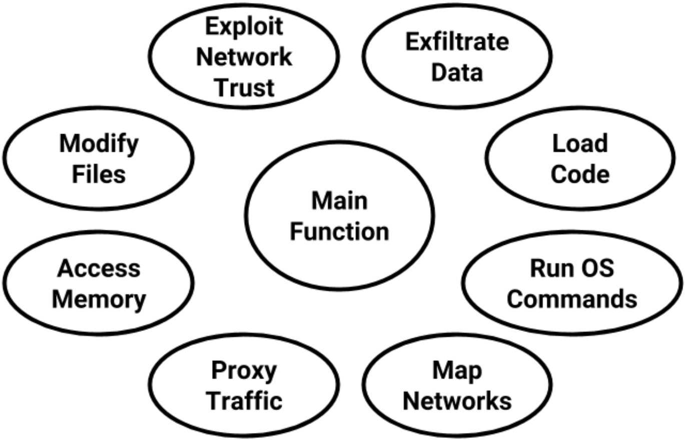


Figure: Overview of the malware functionality for experiment.

Reconstructing Socket Connections

RICE

Object Address	Remote Connection		In/Out	Data (Up to 30 Bytes)
0x91c779b8	10.18.120.18	48002	⇒	Do something evil-48002!
0x91c7ead0	10.18.120.18	48003	⇒	Do something evil-48003!
0x91c85b70	10.18.120.18	48002	⇐	s3cr3t_d4t3_48002-00000000s3cr
0x91c938d8	172.16.124.15	58860	⇒	czNjcjN0X2Q0dDNfNDgwMDItMDAw
0x91c980d0	10.18.120.18	48003	⇐	s3cr3t_d4t3_48003-00000000s3cr
0x91ca5cb8	172.16.124.15	58860	⇒	czNjcjN0X2Q0dDNfNDgwMDMtMDAw
0x91cbfef0	10.18.120.18	48004	⇒	Do something evil-48004!
0x91cc7008	10.18.120.18	48005	⇒	Do something evil-48005!
0x91ccdee8	10.18.120.18	48004	⇐	s3cr3t_d4t3_48004-00000000s3cr
0x91cdbad0	172.16.124.15	58860	⇒	czNjcjN0X2Q0dDNfNDgwMDQtMDAw
0x91ce02c8	10.18.120.18	48005	⇐	s3cr3t_d4t3_48005-00000000s3cr
0x91cedeb0	172.16.124.15	58860	⇒	czNjcjN0X2Q0dDNfNDgwMDUtMDAw

Reconstructing Events

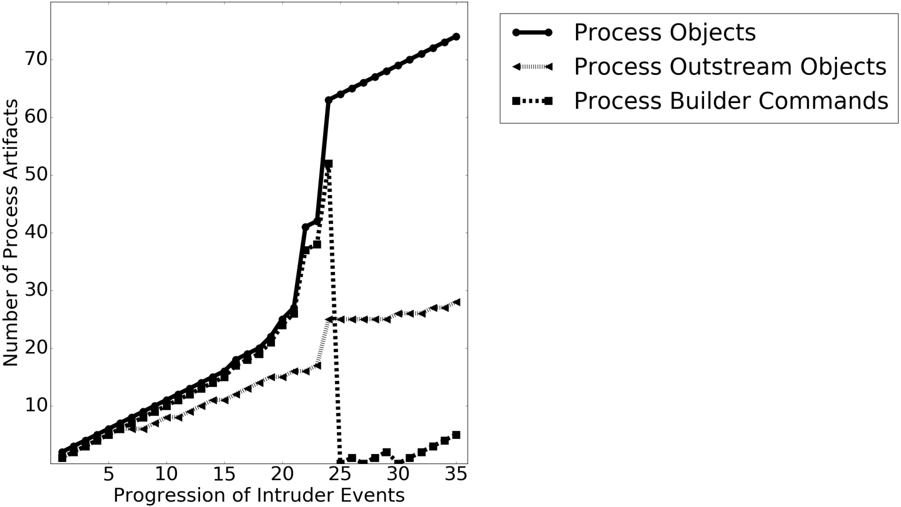
RICE

Table: This table shows a sampling of the processes started by the Java program and the `stdout` buffer at t=21.

Address	PID	Buffered Data
`0x91dff7e0`	1242	#\n# This file MUST be edited w
`0x91e1c7e8`	1245	Linux java-workx32-00 3.19.0-1
`0x91e3b0e0`	1248	java adm cdrom sudo dip plugde
`0x91e4a6e8`	1250	root:x:0:0:root:/root:/bin/bas
`0x91eb1390`	1252	root:!:16678:0:99999:7:::\ndaem
`0x91f66708`	1275	\nStarting Nmap 6.47 (http://n
`0x91ff7ed0`	1301	history | grep pg\n history | gr
`0x92014f30`	1307	ifconfig\nsudo add-apt-reposito
`0x920626d8`	1322	adding: home/java/.ssh/ (sto

Evaluating Event Reconstruction: Process Objects

RICE



Conclusions

RICE

- Improves on existing memory analysis techniques
- Developed a methodology for recovering Java Objects
- Minimal solution achieves cross platform recovery
- Able to perform timelining and event reconstruction

Conclusions

RICE

- Quantified these effects using TLS keys
- Developed a solution reducing latent secrets
- Developed a framework for recovering latent object
- Showed that latent objects can be used for forensics

Questions

RICE

[1] N. Byte, “What the Heck was Stuxnet!?,” January 2016.

[2] A. Drozhzhin, “Adwind malware-as-a-service hits more than 400,000 users globally,” February 2016.

[3] V. Kamluk and A. Gostev, “ADWIND: A CROSS-PLATFORM RAT,” February 2016.

[4] J. Scott-Railton, M. Marquis-Boire, C. Guarnieri, and M. Marschalek, “Packrat: Seven years of a south american threat actor,” December 2015.

[5] V. Kamluk, C. Raiu, and I. Soumenkov, “THE âĂŸICEFOGâĂŹ APT: A TALE OF CLOAK AND THREE DAGGERS,” September 2013.

[6] V. Kamluk, C. Raiu, and I. Soumenkov, “The icefog apt hits us targets with java backdoor,” January 2014.

[7] E. Galperin, C. Quintin, M. Marquis-Boire, and C. Guarnieri, “I Got a Letter From the Government the

Other Day...:Unveiling a Campaign of Intimidation, Kidnapping, and Malware in Kazakhstan," 2016.

[8] J. Drake, "Exploiting memory corruption vulnerabilities in the java runtime," 2011.

[9] J. Viega, "Protecting sensitive data in memory," 2001.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] A. Walters, “The Volatility Framework: Volatile memory artifact extraction utility framework,” 2007.

[16] M. Cohen, “Rekall memory forensics framework,” in *DFIR Prague 2014*, SANS DFIR, 2014.

[17] Z. Lin, X. Zhang, and D. Xu, “Automatic reverse engineering of data structures from binary execution,” 2010.

[18] Z. Lin, J. Rhee, X. Zhang, D. Xu, and X. Jiang, “Siggraph: Brute force scanning of kernel data structure instances using graph-based signatures.,” in *NDSS*, 2011.

[19] B. Dolan-Gavitt, A. Srivastava, P. Traynor, and J. Giffin, “Robust signatures for kernel data structures,” in *Proceedings of the 16th ACM conference on Computer and communications security*, ACM, 2009.

[20] H. Macht, “Live memory forensics on Android with volatility,” *Friedrich-Alexander University Erlangen-Nuremberg*, 2013.

[21] A. P. Heriyanto, “Procedures and tools for acquisition and analysis of volatile memory on Android smartphones,” 2013.

[22] J. Park, H. Chung, and S. Lee, “Forensic analysis techniques for fragmented flash memory pages in smartphones,” *Digital Investigation*, vol. 9, no. 2, 2012.

[23] A. Case, “Memory analysis of the Dalvik (Android) virtual machine,” Dec. 2011.

[24] J. Okolica and G. Peterson, “Extracting the windows clipboard from memory,” in *Proceedings of the 2011 DFRWS Conference*, 2011.

[25] A. Schuster, “Searching for processes and threads in microsoft windows memory dumps,” in *Proceedings of the 2006 DFRWS Conference*, 2006.

[26] G. G. Richard III and V. Roussev, “Scalpel: A frugal, high performance file carver.,” in *DFRWS*, 2005.

[27] R. Beverly, S. Garfinkel, and G. Cardwell, “Forensic carving of network packets and associated data structures,” *digital investigation*, vol. 8, 2011.

[28] S. Hand, Z. Lin, G. Gu, and B. Thuraisingham, “Bin-carver: Automatic recovery of binary executable files,” *Digital Investigation*, vol. 9, 2012.

[29] A. Pridgen, S. Garfinkel, and D. S. Wallach, “Present but unreachable: Reducing persistent latent secrets in HotSpot JVM,” in *System Science (HICSS), 2017 50th Hawaii International Conference on*, IEEE, 2017.