Pool Tag Quick Scanning for Windows Memory Analysis

Greater New Orleans ISACA Meeting

March 10, 2016

Joe T. Sylve, M.S.

Senior Research Developer
Blackbag Technologies

Ph.D. Candidate
Department of Computer Science
University of New Orleans

Grab a Copy of These Slides

https://github.com/jtsylve/slides

#whoami

- NOLA Native
- UNO Alumnus
  - B.S., Computer Science – 2010
  - M.S., Computer Science – 2011
  - Ph.D., Computer Science (Expected) – 2016
- Digital Forensic & Computer Security Researcher and Practitioner
  - Blackbag Technologies
- Co-Organizer
  - NOLASec
  - BSidesNOLA

Commercial Break

- Monthly Security Meetup
- Next meeting tentatively March 30th
- 6 PM
- Lucy's Retired Surfer Bar
- www.nolasec.com

Commercial Break

- 4th Annual Day-Long Conference
- April 16, 2016
- 16 speakers
- $15
  - T-Shirt
  - Food
  - Drinks
- www.bsidesnola.com

What We'll Cover

Pool Tag Quick Scanning for Windows Memory Analysis

Joe T. Sylve[a,b], Vico Marziale[a], Golden G. Richard III[b]

[a] *Blackbag Technologies, Inc, San Jose, CA, USA*
[b] *Department of Computer Science, University of New Orleans, New Orleans, LA, USA*

Abstract

Pool tag scanning is a process commonly used in memory analysis in order to locate kernel object allocations, enabling investigators to discover evidence of artifacts that may have been freed or otherwise maliciously hidden from the operating system. The fastest current scanning techniques require an exhaustive search of physical memory, a process that has a linear time complexity over physical memory size. We propose a novel technique that we are calling "pool tag quick scanning" that is able to reduce the scanning space by 1-2 orders of magnitude, resulting in much faster discovery of targeted kernel data structures, while maintaining a high degree of accuracy.

Keywords: Microsoft Windows, Memory Analysis, Memory Forensics, Live Forensics, Pool Tag Scanning, Pool Scanning, Incident Response

---

Memory Forensics

A BRIEF INTRODUCTION

Why Memory Forensics?

- Traditional Analysis Deals w/ Non-Volatile Data
  - Hard Drives
  - Removable Media
  - Etc.
- Live Forensics Deals with Volatile Data
  - RAM Mostly
  - Must be Collected From a Running Machine
  - Not as Much Control Over Environment

Why Memory Forensics?

- RAM Dumps Contain Both Structured and Unstructured Information
  - Unstructured
    - Strings
    - Application Data
    - Fragments of Communication
    - Encryption Keys
  - Structured
    - Kernel and Application Structures

Why Memory Forensics?

- With This Data We Can Gather Information About
  - Processes
  - Open Files
  - Network Connections
  - "In-Memory-Only" Application Data
    - Private Browsing Mode
    - Unencrypted Data
    - Webmail
    - Etc.

Why Memory Forensics?

- Advanced Malware
- Encrypted or Temporary File Systems
- “Live” Computing Environments
- Analysis
  - Volatility
  - Rekall
  - Redline
  - HPMAF

How It Works

- Two Approaches
  - Signature-Based Data Carving
  - Structured Analysis
- We'll Discuss Both

Signature-Based Data Carving

- Simply Scan the Raw Memory Dump for Patterns
  - Ex. Firefox ~~Porn~~ Private Browsing History
    - Search for “HTTP-memory-only-PB”
    - Strings afterwards will be URLs
  - Can use Regular Expressions
    - Ex. Email Addresses: .*@.*\..*
- Bulk Extractor is a Good Example of This Technique

Signature-Based Data Carving

- Pros
  - Reasonably Fast
  - Easy to Implement
    - Strings, Grep, & Awk
  - Sometimes All You Need

Signature-Based Carving

- Cons
  - Not Context-Aware
    - Which Process or File?
  - Fragmented Data is Hard to Recover
    - Physical Memory is Essentially a “Random” Collection of 4KB Pages
  - Fails on Compressed Images
    - Pagefile.sys
    - OS X
    - Windows 10

Structured Analysis

- Attempts to Recreate the OS Runtime State
  - Locate Kernel
  - Parse Structured Kernel Information
  - Perform Address Translation
  - Parse All the Things!
- Tool Examples
  - Volatility
  - Rekall
  - HPMAF

Structured Analysis

- Why the Kernel?
  - Managing Codebase of the OS
  - References Information About Everything
    - Processes
    - Files
    - Sockets
    - Drivers

Structured Analysis

- Locate the Kernel
  - Memory Dump Format
    - Crash Dump
    - Hiberfile.sys
  - Register State
    - Some Registers Point to Key Kernel Structures
  - Data Carving
    - Look for “Known Signatures” inside the kernel

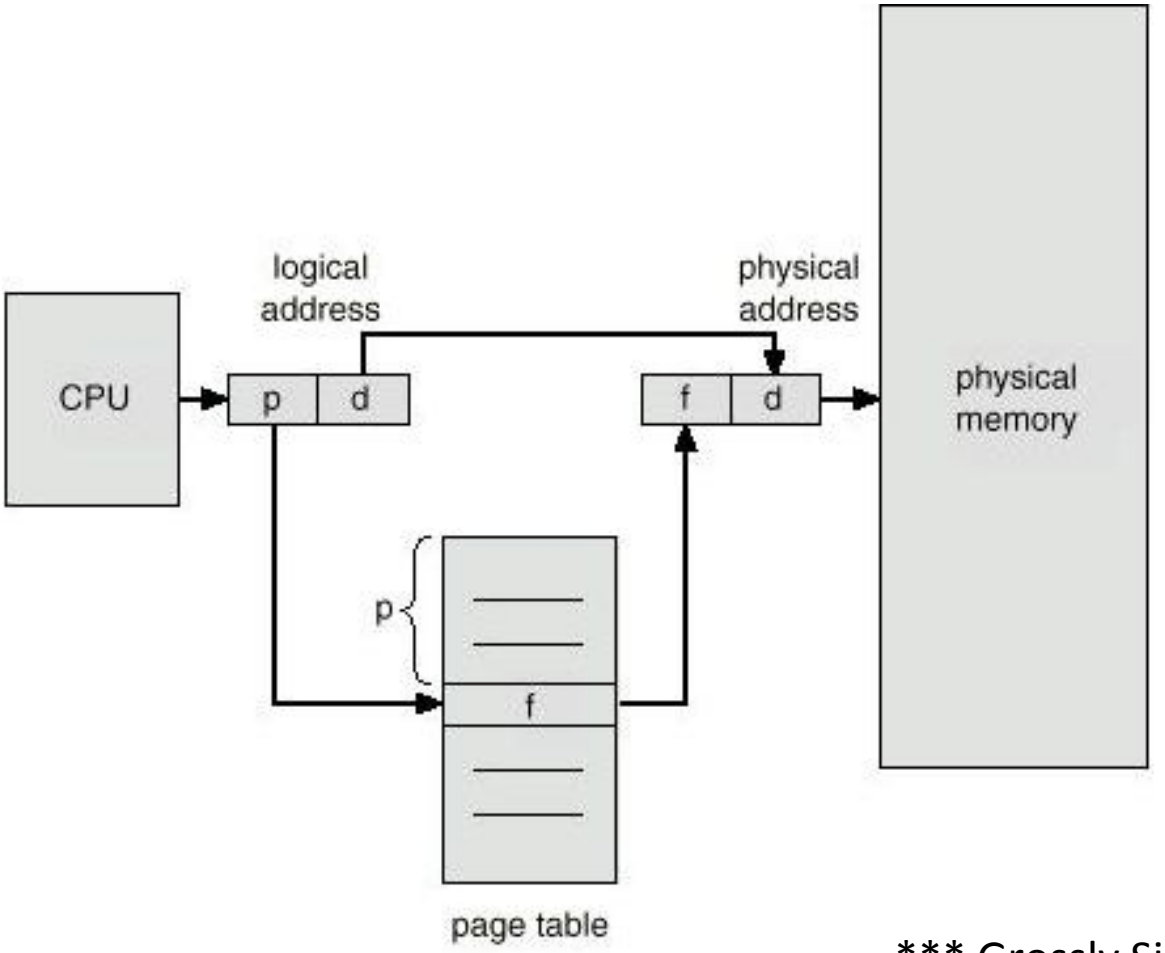
Structured Analysis

- Parse Structured Kernel Information
  - Look for Structured Data That Should Live at Known Positions in the Kernel
    - Ex. Linked List of Processes
      - nt!PsActiveProcessHead
  - Parsing This Data Will Give Us Virtual Addresses of Artifacts

Structured Analysis

- Perform Address Translation
  - Physical Memory is a “Random” Collection of Pages
    - Usually 4KiB
  - Each Process Has It’s Own “Virtual Memory”
    - Linear Addresses
    - Ordered
  - Each Process Has a Table That Maps Virtual to Physical Addresses

Structured Analysis



*** Grossly Simplified

Structured Analysis

- Parse All the Things!
  - Now We Know Where Things "Live" in Physical Memory
  - We Can Look at Physical Pages "in Order"
  - Decompress (if Needed)
  - Parse Artifacts
    - Data Carving
    - Structured Application Analysis

Structured Analysis

- Pros
  - An Incredible Amount of Information Can be Found
  - Handles “Fragmented” Data Well
  - Context Aware
    - Results Can be Associated With Specific Processes, Files, and Possibly Users
  - Very Hard to Hide From

Structured Analysis

- Cons
  - Requires More Sophisticated Tools
  - Currently Available Tools Are Slow
    - That's where my research comes in

Overview

POOL TAG QUICK SCANNING

Pool Tag Quick Scanning

- The Windows Kernel contains several “Pools” of memory to handle object allocation
  - Sort of the kernel version of a process heap
- Each allocated object is prepended with a header, which has a 4 byte identifier of type and some other info
- Scanning memory for Pool Headers allows finding unallocated or unlinked structures

Pool Tag Quick Scanning

Purpose	Pool Tag
Driver Object	Driv
File Object	File
Kernel Module	MmLd
Logon Session	SeLs
Process	Proc
Registry Hive	CM10
TCP Endpoint	TcpE
TCP Listener	TcpL
Thread	Thre
UDP Endpoint	UdpA

Table 1: Selected *non-paged pool* allocations

Pool Tag Quick Scanning

- Volatility and Rekall will do an exhaustive search of the entire memory image looking for pool headers
  - Slow
  - Many False Positives
    - Pool tags are only 4 bytes (usually ASCII)
    - Scanning Userland memory
    - Some sanity checking can cut down on these

Pool Tag Quick Scanning

- In reality the pool will only take up a very small percentage of physical memory
  - Depends on usage, not physical memory size
  - Ex. 16 GiB physical memory
    - ~38MiB of allocated Non-Paged Pool pages
- By limiting our pool tag scanning to only those pages that the kernel uses to store the pool we can vastly speed up the process and cut down false positives

Pool Tag Quick Scanning

- The kernel maintains symbols for variables that tell us where each pool starts as well as the size of the pool's virtual memory space
  - However the virtual space reserved for pools are huge!
  - The vast majority of the pages are unallocated
- The kernel also maintains an allocation bitmap
  - One bitmap per pool
  - Each bit represents 2MiB of allocations

Pool Tag Quick Scanning

Windows Version	Static Ranges	Start of Dynamic Allocation	Allocation Bitmap
Vista SP0	`MmNonPagedPoolStart - MmNonPagedPoolEnd0`	`MmNonPagedPoolExpansionStart`	`MiNonPagedPoolVaBitMap`
Vista SP1 - 8	N/A	`MiNonPagedPoolStartAligned`	`MiNonPagedPoolVaBitMap`
8.1	N/A	`MiNonPagedPoolStartAligned`	`MiDynamicBitMapNonPagedPool`

Table 2: Relevant kernel symbols for identifying *non-paged pool* ranges

Pool Tag Quick Scanning

- By enumerating the bitmaps we can learn which pages in the pool's virtual address space are allocated
- We limit our pool tag scanning to only the allocated pages
- This is several orders of magnitude faster than scanning all of physical memory
- Does not grow linearly with physical memory size

Pool Tag Quick Scanning

- Some hits may be missed (false negatives)
  - Allocations from a previous boot
    - Some systems do not wipe memory on reboot
  - Pool pages reclaimed by the OS
    - In rare circumstances the pool can shrink
  - Hits from memory mapped files
    - For example a memory image from a VM
- These limitations are common to all current techniques starting with Windows 10 anyway

RESULTS

Plugin	Type	Avg. Time	Running	Terminated	Prior Boot	Duplicate[4]
psquickscan	Virtual	0.129s	128	21	0	0
psscan	Physical	15.584s	128	22	15	43
psscan (Rekall)	Physical	35.967s	128	22	15	43
psscan (Volatility)	Physical	25.448s	128	21	15	43

Table 3: A comparison of *psquickscan* and *psscan* results

Plugin	Type	Avg. Time	Running	Terminated	Prior Boot	Duplicate[4]
psquickscan	Virtual	0.129s	128	21	0	0
psscan (Rekall)	Virtual	71.513s	128	21	0	1
psscan (Volatility)	Virtual	60.526s	128	19	0	1

Table 4: A comparison of virtual scanning

OS Version	Plugin	Data Scanned	RAM Size	Avg. Time	Running	Terminated	Duplicate
Vista SP0	psquickscan	38 MiB	1 GiB	0.083s	46	2	15
Vista SP0	psscan	1 GiB	1 GiB	0.356s	46	2	15
Vista SP1	psquickscan	60 MiB	1 GiB	0.073s	48	0	0
Vista SP1	psscan	1 GiB	1 GiB	0.400s	48	0	0
Vista SP2	psquickscan	76 MiB	1 GiB	0.236s	50	1	0
Vista SP2	psscan	1 GiB	1 GiB	0.547s	50	1	11
7 SP0	psquickscan	64 MiB	2 GiB	0.075s	43	4	0
7 SP0	psscan	2 GiB	2 GiB	0.712s	43	6	4
7 SP1	psquickscan	64 MiB	2 GiB	0.075s	50	5	0
7 SP1	psscan	2 GiB	2 GiB	0.691s	50	5	0
8	psquickscan	44 MiB	4 GiB	0.054s	36	3	0
8	psscan	4 GiB	4 GiB	1.433s	36	3	0
8.1	psquickscan	244 MiB	8 GiB	0.170s	45	0	0
8.1	psscan	8 GiB	8 GiB	2.977s	45	0	0

Table 5: A sample of *psquickscan* and *psscan* results among different OS versions

Plugin	Data Scanned	Avg. Time
psquickscan	5.76 GiB	5.797s
psscan	192 GiB	3m8.421s
psscan (Rekall)	192 GiB	6m7.207s
psscan (Volatility)	192 GiB	4m42.412s

Table 6: Comparison of scanning speed on a 192 GB Memory Image

RAM Size	Plugin	Scanned	Time	Transferred
2 GiB	psquickscan	102 MiB	9.489s	116.115 MiB
2 GiB	psscan	2 GiB	28.132s	2.014 GiB
4 GiB	psquickscan	122 MiB	9.640s	177.367 MiB
4 GiB	psscan	4 GiB	56.971s	4.027 GiB
8 GiB	psquickscan	246 MiB	15.360s	299.648 MiB
8 GiB	psscan	8 GiB	3m26.449s	8.132 GiB

Table 7: Bandwidth comparison using F-Response to scan a Windows target