In Lieu of Swap: Analyzing Compressed RAM in Mac OS X and Linux

Golden G. Richard III
University of New Orleans and Arcane Alloy, LLC
golden@cs.uno.edu

Andrew Case
The Volatility Project
andrew@dfir.org
Who?

Professor of Computer Science and University Research Professor, Director, GNOCIA, University of New Orleans
http://www.cs.uno.edu/~golden
Digital forensics, OS internals, reverse engineering, offensive computing, pushing students to the brink of destruction, et al.

Founder, Arcane Alloy, LLC.
http://www.arcanealloy.com
Digital forensics, reverse engineering, malware analysis, security research, tool development, training.

Co-Founder, Partner / Photographer, High ISO Music, LLC.
http://www.highisomusic.com
Rock stars. Heavy Metal. Earplugs.
Forensics, Features, Privacy

- New OS features may negatively forensics
- e.g.:
  - Native whole disk encryption
  - Encrypted swap
  - Secure Recycle Bin / Trash facilities
  - Secure erasure during disk formatting
- Today’s topic: Compressed RAM
- Claim: **positively impacts** forensics
- First some background
Where’s the Evidence?

Files and Deleted Files
Filesystem metadata
Application metadata
Windows registry
Print spool files
Hibernation files
Temp files
Log files
Slack space
Swap files
Browser caches
Network traces
RAM: OS and app data structures
Memory analysis

“Traditional” storage forensics
Memory Analysis on RAM Dump

- Processes (dead/alive)  e.g., discover unauthorized programs
- Open files  e.g., detect keystroke loggers, hidden processes
- Network connections  e.g., find backdoors, connections to contraband sites
- Volatile registry contents
- Volatile application data  e.g., plaintext for encrypted material, chat messages, email fragments
- Other OS structures  e.g., analysis of memory allocators to retrieve interesting structures, filesystem cache, etc.
- Encryption keys  e.g., keys for whole disk encryption schemes

What’s missing?
Memory Analysis: Swap Files

“Traditional” forensics: capture contents of storage devices, including the swap file

Swap file contains RAM “overflow”—if you don’t have this data, you probably don’t have a complete memory dump

Live forensics / memory analysis: capture RAM contents

If you acquire both RAM and disk contents, memory smearing is likely and the swap file may be (very) out of sync with the memory dump
Forensically, Swap Files are a Mess

Data trapped in unsanitized blocks allocated to swap file

RAM dump

Swap file

provenance??

encrypted swap

Swap file

order

void swap_crypt_ctx_initialize ( void ) {
  unsigned int i;
  if ( swap_crypt_ctx_initialized == FALSE ) {
    for ( i = 0; i < ( sizeof ( swap_crypt_key ) / 
        sizeof ( swap_crypt_key [0] ) ); i++) {
      swap_crypt_key [i] = random();
    }
    aes_encrypt_key ( (const unsigned char *) swap_crypt_key , 
                       SWAP_CRYPTO_AES_KEY_SIZE ,
                       &swap_crypt_ctx.encrypt);
    ...
    swap_crypt_ctx_initialized = TRUE ;
  }
}
Background: Memory Management

• Need to allocate RAM to:
  – Operating system
  – Individual applications
  – All applications
    • Possibly fairly, or some other policy

• OS’s need to manage memory efficiently to feed RAM-hungry apps

• Ever-increasing “need” for more memory

© 2014 by Golden G. Richard III and Andrew Case
Virtual Memory

• Provides contiguous, linear address space for processes
• Virtual address space is divided into fixed-size pages
  – Physical memory is divided into pages of same size
  – On x86 Intel, these are normally 4K
• OS keeps track of free and allocated pages
• Processes get only as many pages as they need

© 2014 by Golden G. Richard III and Andrew Case
Logical vs. Physical Addresses

- **Virtual address**: Address whose scope is a particular process or OS kernel
- **Physical address**: Location in physical RAM
Paging: (Very) Simplified View

- Page tables (per process) are used to translate logical to physical addresses
- Pages for a particular process are generally not in contiguous order in RAM

![Diagram showing paging and page tables]
64-bit Mode: 4-level Page Tables: 4K Pages

Figure 3-24. IA-32e Mode Paging Structures (4-KByte Pages)

NOTES:
1. 40 bits aligned onto a 4-KByte boundary
Stretching RAM

• Want to allocate more memory to processes than total amount of RAM
• Extend RAM by swapping pages to disk and retrieving as necessary
• Allows more (and bigger) processes to execute
• Workable, because:
  – Not all executing processes are active at once
  – Processes typically have “working sets” of pages, e.g., memory they’re actively using “now”
Virtual Memory with Swapping

For good performance, can’t overdo it--the pipe between RAM and disk is small

Otherwise: THRASHING

© 2014 by Golden G. Richard III and Andrew Case
Virtual Memory with Swapping

SSDs substantially increase size of pipe

500GB/sec or so

swap file on SSD

SSDs substantially increase size of pipe

500GB/sec or so

swap file on SSD

© 2014 by Golden G. Richard III and Andrew Case
Virtual Memory with Swapping

up to 1.2GB/sec+ (e.g., new Mac Pro, newest iMacs)

swap file on PCIe flash storage

Process P1

P1 page table

RAM

© 2014 by Golden G. Richard III and Andrew Case
Avoiding Swapping by Compressing RAM

• Optimization:
  – “Hoard” some RAM, not available to processes
  – When RAM is running low, compress pages in memory instead of swapping to disk

• Why bother?
  – SSDs are fast
  – PCIe solid state storage is “crazy” fast
  – Memory is ever cheaper

• So...why?
It’s Worth It

• Ultraportable laptops still RAM limited
  – e.g., current Macbook Air: Base 4GB, Max 8GB RAM
• Virtualization, even on laptops
• Malware analysts, VMWare Workstation, Fusion, etc.
• Swapping can cause serious performance issues
• New Mac Pro memory bandwidth: 60GB/sec
• Max swap bandwidth: 1.2GB/sec
• Modern CPUs compress and decompress very efficiently
Virtual Memory with Compressed RAM

In Mac OS X Mavericks and Linux, set aside some RAM and compress pages that would have been swapped due to memory pressure.
It’s Back

• RAM Doubler -- ~20 years ago
• Historically RAM compression not very effective
  – Processors too slow to offset compress/decompress costs
  – Poor integration with OS
• Old academic paper (2003) on compression schemes for RAM:
• Compression scheme in Mavericks (WKdm) was invented in 1999:
• Now, super fast CPUs, lots of cores, tight OS integration
Compressed RAM: Forensic Impact

Currently, “simultaneous” capture results in lots of smearing

With compressed RAM, may be little or no data swapped out—capturing RAM captures (some or all) “swap”!

No support in current tools—compressed data opaque or ignored

Our tools enable analysis of compressed RAM
Mavericks VM: Too Much Detail
Mac OS X Mavericks Implementation

• Implemented as a pager
• Fits into the standard Mach VM architecture
• Compressor pager hoards some memory
• Page in / out requests all go through the pager
• Pages compressed / decompressed on demand
• When compressor gets full, pushes compressed pages to swap file
• On page fault, page is either:
  – Compressed and in RAM: just decompress
  – On disk: read from disk, decompress
Mavericks Implementation (2)

• Implementation in C
• Like most of Mach and BSD
• ...except for optimized version of WKdm
• Pages are compressed and decompressed using WKdm
• Mavericks: Hand-optimized 64-bit assembler version of Kaplan’s original C code
Mach VM: C + asm

xnu-2422.1.72/osfmk/vm: ~73K lines of C

xnu-2422.1.72/osfmk/x86_64/WKdm*.s: ~1000 lines of 64-bit assembler for WKdm_compress_new() / WKdm_decompress_new()

Just vm_compressor / vm_compressor_pager: ~4K lines of C + the assembler, above

FIGURE 8–6 Details of the Mac OS X Mach VM architecture
Compressor / Compressor Pager Internals

virtual address to slot mapping

virtual address of page

cpgr_slots

compressor_pager (per vm_map_entry object)

c_size
c_offset

c_segments

c_slots

c_buffer

compressor

compressed pages

compressor_object (global)
Linux

• zram/ zswap (circa 3.11+ kernel release) in Linux
• zram is just a memory-based compressed swap device
• zswap uses new FRONTSWAP facility in Linux
```c
int swap_writepage(struct page *page, ... )
{
    int ret = 0;
    if (try_to_free_swap(page)) {
        unlock_page(page);
        goto out;
    }
    if (frontswap_store(page) == 0) {
        set_page_writeback(page);
        unlock_page(page);
        end_page_writeback(page);
        goto out;
    }
    ret = __swap_writepage(page, wbc, end_swap_bio_write);
out: return ret;
}
```

*Page eventually written to disk*

Can accept the page if it fits, or say no and regular swapping will occur.

e.g., *zswap*

e.g., *zram*, *way down in there*
```c
int swap_readpage(struct page *page)
{
...
...
if (frontswap_load(page) == 0) {
    SetPageUptodate(page);
    unlock_page(page);
    goto out;
}
...
// lots of pain here
// lots of pain here
out:
    return ret;
}
```

- e.g., zswap
- e.g., zram
- it’s just a stranger block device
- Page eventually read from disk

© 2014 by Golden G. Richard III and Andrew Case
Current Reality + Goal

Physical memory dump:

Process P1

Process P2

Process P1

Process P2

© 2014 by Golden G. Richard III and Andrew Case
Volatility

• Most popular memory analysis framework
• Open source
• Portable, written in Python 😞
• Supports analysis of Windows, Linux, Mac
• Plugins add new functionality
• Our contribution: New plugins to support compressed RAM analysis
New Plugins for Volatility

• `mac_compressed_swap` / `linux_compressed_swap`
  – Find, decompress, and dump all compressed pages
  – Emits compressor stats:
    • Compressor memory used : 19167408 bytes
    • Available uncompressed memory : 466462 pages
    • Available memory : 471251 pages
    • ...

• Required Python implementation of decompression algorithms

• Required detailed analysis of new compressor pager

• But not entire page fault $\rightarrow$ decompression path
New / Modified Plugins for Volatility

• `mac_dump_maps / linux_dump_maps`
  – Dump address space for all (or individual) processes
  – Previous implementation used Volatility’s standard mechanism for emitting 4K pages in address space
  – Simply skipped swapped pages
  – Now, decompressed if possible and made available

• Involves deeper analysis of OS VM systems

• Much more complex (not your problem, we’re not complaining)

• We ♥ OS internals
Representative Distribution of Compressed Pages

Mac OS X 10.9, 2GB RAM, 124 processes, moderate memory pressure, 300MB compressed
You're reading. We're hiring.
https://www.flickr.com/jobs/

The Icefjord in Ilulissat

Andalusian girls used or shall I wear a red yes and how he kissed me under the Moorish wall and I thought well as well him as another and then I asked him with my eyes to ask again yes and then he asked me would I yes to say yes my mountain flower and first I put my arms around him yes and drew him down to me so he could...
Aside: Performance

• Q: Why did Apple bother to implement WKdm compression / decompression in assembler?

• Very little kernel code in assembler, mostly C

• But obviously on a very critical path in the kernel

• Turns out, the right decision
WKdm Benchmarks

• Kaplan’s original C version
• Apple’s assembler version
• My Python implementation (needed for integrating compressed RAM analysis in Volatility)
• A version (for fun, and a reality check for Python) in go, that I wrote in a few hours
L_nonpartital:
  jl  L_ZERO_TAG
  cmpb $2, -1(%rsi)
  je  L_MISS_TAG

L_EXACT_TAG:
  movzb! (next_qpos), %eax
  // qpos = next_qpos
  incq next_qpos
  // next_qpos++
  decl  tags_counter
  // tags_counter--
  movl  (%rsp,%rax,4), %eax
  // w = dictionary[qpos]
  mov %rax, -4( dest_buf, %rax, 4)
  // read dictionary word
  dict_location = dictionary[tempTagsArray[next_tag / 4] & self.SINGLE_BYTE_MASKS[next_tag % 4]]
  // replace old value in dict
  dest_buf[next_output] = temp[dict_location]
  // echo it to output
  incq next_output
  // next_output++
  if (tag == self.ZERO_TAG):
    dest_buf[next_output] = 0
  elif (tag == self.EXACT_TAG):
    dest_buf[next_output] = temp
    dest_buf[next_output] = dictionary[dict_location]
    # strip out low bits
    dest_buf[next_output] = temp >> self.NUM_LOW_BITS
    dest_buf[next_output] = temp & self.SINGLE_BYTE_MASKS
    # add in stored low bits from temp array
    dest_buf[next_output] = dest_buf[next_output] | tempLowBitsArray[next_low_bits]
    next_low_bits += 1
  elif (tag == self.PARTIAL_TAG):
    dest_buf[next_output] = temp[next_low_bits] & self.SINGLE_BYTE_MASKS
    dest_buf[next_output] = (temp >> self.NUM_LOW_BITS) << self.NUM_LOW_BITS
    dest_buf[next_output] = dest_buf[next_output] | tempLowBitsArray[next_low_bits]
    next_low_bits += 1
  else:
    return -1  // fail, buffer is corrupted

while (next_tag < tags_area_end):
  tag = (tempTagsArray[next_tag / 4] & self.SINGLE_BYTE_MASKS[next_tag % 4])
  // calculate tag area
  if (tag == self.ZERO_TAG):
    dest_buf[next_output] = 0
  elif (tag == self.EXACT_TAG):
    dest_buf[next_output] = temp
    dest_buf[next_output] = dictionary[dict_location]
    # strip out low bits
    dest_buf[next_output] = temp >> self.NUM_LOW_BITS
    dest_buf[next_output] = temp & self.SINGLE_BYTE_MASKS
    # add in stored low bits from temp array
    dest_buf[next_output] = dest_buf[next_output] | tempLowBitsArray[next_low_bits]
    next_low_bits += 1
  elif (tag == self.PARTIAL_TAG):
    dest_buf[next_output] = temp[next_low_bits] & self.SINGLE_BYTE_MASKS
    dest_buf[next_output] = (temp >> self.NUM_LOW_BITS) << self.NUM_LOW_BITS
    dest_buf[next_output] = dest_buf[next_output] | tempLowBitsArray[next_low_bits]
    next_low_bits += 1
  else:
    return -1  // fail, buffer is corrupted
    // print "BAD TAG!!"

next_tag += 1
Benchmarks (Sorry, Python)

WKdm compression / decompression ops/sec

- asm: 391 per sec
- C: 150,000 ops/sec
- go: 50,000 ops/sec
- Python: 10,000 ops/sec

All on 3.4GHz i7 iMac w/ 32GB RAM
Compressed RAM: Conclusions

• Significantly reduces swapping to disk
• Used very aggressively in Mac OS X Mavericks
• Soothes RAM ↔ swap file consistency problems for memory analysis
• New plugins make more evidence available
• Essential for “complete picture” in memory analysis
• Our plugins integrated into Volatility
  – Check github (released in early September)
  – https://github.com/volatilityfoundation/volatility
  – Transparent decompression support in Volatility 3.0
Questions/Comments?

• Golden G. Richard III
  – http://www.cs.uno.edu/~golden/
  – golden@cs.uno.edu
  – @nolaforensix

• Andrew Case
  – http://www.dfir.org
  – andrew@dfir.org
  – @attrc