Forensic Analysis of Windows User Space Applications Through Heap Allocations

Michael Cohen
Google Inc.
Zurich, Switzerland
Why Userspace analysis?

- Forensically very valuable:
  - Users interact directly with applications.
  - Applications interact with the OS kernel.
  - Therefore we can sometimes infer user activity by OS kernel evidence but not always:
    - e.g. user chats on IRC
      - Sockets, Connections, network packets
      - Strings in IRC process - no context!
Challenges for user-space analysis

- So many user space applications - manual reversing just does not scale.
  - Lots of attention on reversing malware but there are many regular apps that are interesting too.
- User space memory is often paged and address translation is more complex.
  - Current tools and techniques are unable to resolve user space memory from Prototype PTEs or the Pagefile.
Challenges for user-space analysis

● Why is page translation in userspace fairly complex?
  ○ Have to consider shared memory (Prototype PTEs).
  ○ Some memory forensic tools are extremely buggy:
    ■ Associate random data with the content of user space memory. (Very dangerous from an evidentiary perspective.).
  ○ Not a lot of tool testing or verification going on in Memory Forensics.
Experiment

● Small test program:
  ○ VirtualAlloc a large region (around 800Mb).
  ○ Mark each page with its sequence number - we can find the page in physical memory.
  ○ Sleep. Gives us plenty of time to acquire memory.

● What do we expect?
  ○ A VAD region for the allocated region.
  ○ When dumping the VAD region we expect marked pages in sequence (0, 1, 2, 3 etc).
Swapper program

Working set is trimmed as soon as acquisition starts.

Winpmem acquires an AFF4 image.

Source code - mark each page with its number
Let's inspect the VAD
Let's dump out the allocated region

Export the image for Volatility (It can not process AFF4 images)

Latest commit Jun 23 2015: Run vaddump plugin to extract VAD regions.
Volatility: Many of the pages dumped are random garbage. Therefore Volatility cannot handle user space memory reliably.
Address Translation Basics
What if PTE bit 0 is unset?

Pagefault - Hardware calls into the OS kernel to resolve the mapping. Kernel may use any of the other bits in the PTE as it wants. They are OS specific and have different meaning in different OS.
PTE resolution up on pagefault
Implementing OS specific translation

- We implemented the above algorithms in Rekall - the most advanced, open source, memory analysis framework.
- Also implemented plugins to inspect and verify state of PTE so we can double check the translation process.
Typical translation

```
[1] test.aff4 17:22:10> vtop 0x000004ff0000
  vtop(0x000004ff0000)
DEBUG:rekkall.1:Running plugin (vtop) with args ((83820544,)) kwargs ({}).

*************** 0x4ff0000 ***************
 Virtual 0x000004ff0000 Page Directory 0x23d8c000
 pml4e@ 0x23d8c000 = 0x1540000021497867
 pdpte@ 0x21497000 = 0x1000003581b867
 pde@ 0x3581b138 = 0xa30000024d7f847
 pte@ 0x24d7f80 = 0x22ce20000000

PTE Contains 0x22ce20000000
PTE Type: Pagefile

[MMJTE SOFTWARE Soft] @ 0x22cf7f80
  0x00 InStore
  0x00 PageFileHigh
  0x00 PageFileLow
  0x00 Protection
  0x00 Prototype
  0x00 Reserved
  0x00 Transition
  0x00 UsedPageTableEntries
  0x00 Valid

Physical Address 0x859c4000 @ aff4://0603f7cb-b114-4e8c-b565-f43d47ab9fe/c:/pagefile.sys (Mapped 0x62ce2000)
```

Private mapping in pagefile

Resolve the address of the page in the pagefile (stored in the AFF4 Volume)
More complex example

PDE is in Transition. First resolve PDE to find PTE.
Shared memory (e.g. DLL)

- Hardware PTE points to Prototype PTE
- Prototype PTE points to file mapping
Examine our experiment

Page resolved through prototype although it is actually still resident.
Use Rekall to dump VAD

Vad region dumped accurately. All pages are correctly resolved. We have "perfect Rekall"!
What next?

- So now we can reproduce userspace memory accurately what can we do?
  - Applications allocate memory using a heap allocator typically implemented by a library (e.g. MSVCRT).
  - Applications use higher level abstractions
    - Struct - represent similar objects (Size + Use)
    - Data structures:
      - Linked lists
      - Hash table
      - Strings (UTF 16 encoded)
Heap allocation in practice

When heap needs more space, ask the kernel to wholesale allocate a new VAD region.

```
struct my_struct ptr = malloc(sizeof(my_struct))
```

Point of view of Kernel - User allocations are large VAD regions.
Point of view of Application - User allocations are small precise allocations with their own implied purpose.
Each process has multiple heaps from CreateHeap()

VAD Regions Carved out with VirtualAlloc().
Low Fragmentation Heap

LFH_HEAP

LFH_BLOCK_ZONE

HEAP_SUBSEGMENT

HEAP_SUBSEGMENT

HEAP_SUBSEGMENT

LFH_BLOCK_ZONE

HEAP_USERDATA_HEADER

Allocation

Allocation

Allocation
Reversing through heap analysis

• If we can split user space memory into precise allocations we can more easily see relationships between internal data structures.
• Sometime this avoids the need to reverse any code.
Example: Miranda IRC client

IRC Messages

Users in Channel
Pick a message and search for it

grep("Thanks, I".encode("utf-16-le"))
DEBUG:rekkall.1:Running plugin (grep) with args ("T\x09h\x00a\x00n\x00k\x00s\x00,\x00 \x00 s\x00\x00 h\x00 T.h.a.n.k.s,\n DEBUG:rekkall.1:Opened local file /usr/local/google/home/scudette/.rekkall_cache/sessions/
Offset Data
0x32ddd0 78 00 4b 00 69 00 64 00 64 00 6f 06 3a 00 20 00 x.K.i.d.d.o:....
0x32dde0 54 00 68 00 61 00 6e 00 6b 00 73 00 2c 00 20 00 T.h.a.n.k.s,....
0x32ddf0 49 00 20 00 77 00 61 00 73 00 20 00 77 00 6f 00 I...w.a.s...w.o.
Offset Data
0x32f0f8 78 00 4b 00 69 00 64 00 64 00 6f 06 3a 00 20 00 x.K.i.d.d.o:....
0x32fd08 54 00 68 00 61 00 6e 00 6b 00 73 00 2c 00 20 00 T.h.a.n.k.s,....
0x32fd18 49 00 20 00 77 00 61 00 73 00 20 00 77 00 6f 00 I...w.a.s...w.o.
Offset Data
0x33e690 78 00 4b 00 69 00 64 00 64 00 6f 00 3a 00 20 00 x.K.i.d.d.o:....
0x33dea0 54 00 68 00 61 00 6e 00 6b 00 73 00 2c 00 20 00 T.h.a.n.k.s,....
0x33e6b0 49 00 20 00 77 00 61 00 73 00 20 00 77 00 6f 00 I...w.a.s...w.o.
Offset Data
0x42ad108 cf 94 b1 45 25 00 00 90 54 00 00 00 ba ba ba ab ...E%...T........
0x42ad118 54 00 68 00 61 00 6e 00 6b 00 73 00 2c 00 20 00 T.h.a.n.k.s,....
0x42ad128 49 00 20 00 77 00 61 00 73 00 20 00 77 00 6f 00 I...w.a.s...w.o.
View the string allocation

- Rekall shows the string is allocated from front-end allocator, with size 112 bytes.
  - Appears to have 8 bytes preamble (size + const).

```
[1] MirandaTest.aff4 21:32:05> show_allocation 0x42ad118
  show_allocation(0x42ad118)
DEBUG:recall.1:Running plugin (show_allocation) with args ((69914904,)) kwargs ({}).
Address 0x42ad118 is 8 bytes into F allocation of size 112 ( 0x42ad110 - 0x42ad180)
Offset | Data | Comment
-------|------|--------
0x42ad110 | 54 00 00 00 ba ba ba ab | Appears to be extra allocation data.
... (more data)...
```
Who refers to this string?

- References to start of string.
  - No references to start of alloc - only to string.

```
[1] MirandaTest.E01 18:32:05> show_referrer_alloc 0x42ad110
  show_referrer_alloc(0x42ad110)
DEBUG:rekkal.1:Running plugin (show_referrer_alloc) with args ((69914896,)) kwargs ({})
[1] MirandaTest.E01 18:32:08> show_referrer_alloc(0x42ad118)
  Plugin: show_referrer_alloc
DEBUG:rekkal.1:Running plugin (show_referrer_alloc) with args ((69914904,)) kwargs ({})
Address 0x42ae308 is 8 bytes into F allocation of size 112 ( 0x42ae300 - 0x42ae370)
```

<table>
<thead>
<tr>
<th>Offset</th>
<th>Data</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x42ae300</td>
<td>50 00 00 00 ba ba ba ab</td>
<td>18 d1 2a 04 00 00 00 00 P.......... 0x42ad118(112@0x42ad110)</td>
</tr>
<tr>
<td>0x42ae310</td>
<td>08 43 27 04 00 00 00 00</td>
<td>00 00 00 00 00 00 00 00 C'.......... 0x4274308(48@0x4274300)</td>
</tr>
<tr>
<td>0x42ae320</td>
<td>00 00 00 00 00 00 00 00</td>
<td>00 00 00 00 00 00 00 00 .............</td>
</tr>
<tr>
<td>0x42ae330</td>
<td>00 00 00 00 00 00 00 00</td>
<td>00 00 00 00 00 00 00 00 .............</td>
</tr>
<tr>
<td>0x42ae340</td>
<td>49 00 00 00 00 00 00 00</td>
<td>00 00 00 00 00 00 00 00 .......... ^ U</td>
</tr>
<tr>
<td>0x42ae350</td>
<td>a8 e5 2a 04 00 00 00 00 ba ba ba ab</td>
<td>20 00 41 00 @.......... A 0x42ae0d8(112@0x42ae0d0)</td>
</tr>
<tr>
<td>0x42ae360</td>
<td>4e 00 53 00 49 00 00 00 e9 97 b1 45 25 00 00 94 N.S.I...... E%...</td>
<td></td>
</tr>
</tbody>
</table>
What does this struct refer to?
MESSAGE_RECORD

0x42ae0d8
0  Message
8  Sender
48 Timestamp
64 Prev
72 Next

0x42ae308
0  Message
8  Sender
48 Timestamp
64 Prev
72 Next

0x42ae5a8
0  Message
8  Sender
48 Timestamp
64 Prev
72 Next
Can we list all the messages?

```python
[1] MirandaTest.E01 19:36:26> a=miranda.MESSAGE_RECORD(0x42ae308)
    print "%s %s: %s" % (x.Timestamp, x.Sender.deref(), x.Message.deref())

2015-06-28 23:16:46+0000 BeatrixKiddo: Thanks, I was worried for a second there.
2015-06-28 23:16:55+0000 Zereo: lol
2015-06-28 23:17:13+0000 fioco: I'll give free codes to all here with a steam that has been here before
2015-06-28 23:17:27+0000 fioco: So no to new comers that randomly join for a code :p
2015-06-28 23:17:29+0000 BeatrixKiddo: Was that even a sentence?
2015-06-28 23:17:39+0000 fioco: Yes
2015-06-28 23:17:43+0000 fioco: Yes it was
2015-06-28 23:18:08+0000 BeatrixKiddo: A sentence usually needs to be grammatically correct.
2015-06-28 23:18:28+0000 fioco: Minus maybe a comma or two, it was
2015-06-28 23:18:35+0000 Zereo: Ohh that reminds me I still got 2 castle crasher free game coupons from s
2015-06-28 23:18:37+0000 fioco: It's IRC not a business email ;p
2015-06-28 23:18:42+0000 fioco: I DONT
```
We can now write a plugin

- Extracting the internal Miranda state is as simple as understanding the data structures used by the application.
- No need to reverse engineer code in many cases.
  - It helps when we see the memory the way the application sees it:
    - Like size allocations have same functionality.
    - Can see interconnection between allocations.
Conclusions

- For the first time a FOSS memory analysis framework supports reliable user space address translation.
  - Prototype PTE, Page file, Transitioned PDEs etc.
  - High quality address translation is essential in order to reliably parse heap structures.
- Thorough heap analysis enables seeing memory through an app's own abstractions.
Future work.

- Have you ever been disappointed that `vaddump` or `dumpfiles` plugin produces files with missing pages?
Future Work

● When you think about it - why do we ever dump files out of memory? Because we forgot to acquire them in the first place!

● Full system state = physical memory + pagefile + mapped files.

● We need better acquisition! Also grab mapped files!

● Coming soon to a Rekall near you!