Profile generation in Volatility

Alpha Abdoulaye

EPITA

July 14, 2016

Volatility

A memory forensics framework

- Physical memory dumps
  - Raw dumps
  - Crash dumps
  - Hibernation files
  - VM snapshots
- Open source
- Multi-platform
  - Windows
  - Linux & Mac
- Python

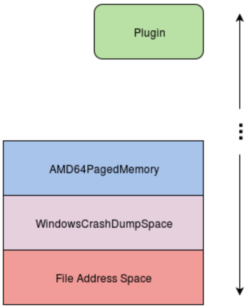
Features

- Running processes
- Memory map
- Open sockets, file descriptors, etc.
- Mounted devices
- Loaded kernel modules
- Log infos
- . . .

Plugins

- Extensive use of plugins
  - Core
  - Community
- Minimal API documentation

Address spaces stacking



Candidate AS

MachOAddressSpace

WindowsHiberFileSpace32

VMWareMetaAddressSpace

QemuCoreDumpElf

OSXPmemELF

Profiles

Content

- Metadata
- Syscalls info
- Constant values
- Native types
- System map
- VTypes
- And more

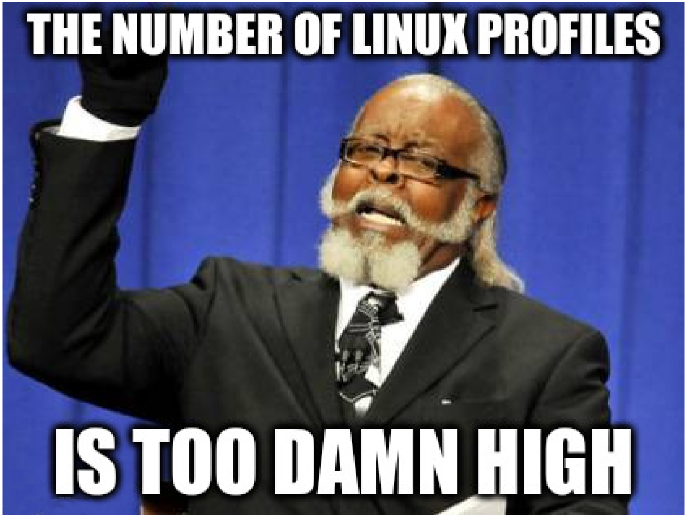
Windows

- Windows -> “reasonable” amount (∼ 30)

$ python vol.py –info

```
VistaSP0x64  - A Profile for Windows Vista SP0 x64
VistaSP1x86  - A Profile for Windows Vista SP1 x86
Win2003SP0x8 - A Profile for Windows 2003 SP0 x86
[...]
Win7SP0x86   - A Profile for Windows 7 SP0 x86
Win8SP0x64   - A Profile for Windows 8 x64
WinXPSP2x8   - A Profile for Windows XP SP2 x86
```

Linux



Symbols table

- Kernel symbols
  - Name
  - Address
  - type
- Kernel oops
- klogd
- syslogd

System.map

/boot/System.map

```
c041f144 b unix_nr_socks
c041f148 b __key.31159
c041f160 b unix_socket_table
c041f564 B unix_tot_inflight
c041f568 b unix_gc_lock
c041f56c b gc_in_progress
c041f570 b wireless_nlevent_queue
c041f580 B __bss_stop
c041f580 B _end
c0420000 B pg0
```

Kallsyms

- Procfs sequence file
- Extracts all kernel's sections and symbols

/proc/kallsyms

```
ffffffffa0001529 t cleanup_module   [serio]
ffffffffa0000900 T __serio_register_port  [serio]
ffffffffa00008c0 T serio_reconnect  [serio]
ffffffffa00000b0 T serio_open   [serio]
ffffffffa0000130 T serio_close  [serio]
ffffffffa0000d40 T __serio_register_driver [serio]
```

VTypes

- Kernel's data structure
- Dummy module
- dwarfdump output -> Get DWARF symbols

module.dwarf

```
.debug_info
<0><0x0+0xb><DW_TAG_compile_unit>[...]
<1><0x1d><DW_TAG_typedef> DW_AT_name[...]
<1><0x28><DW_TAG_base_type> DW_AT_byte[...]
```

How to. . .

VTypes

```
'process' : [ 26, {
    'pid' : [ 0, ['int']],
    'parent_pid' : [ 4, ['int']],
    'name' : [ 8, ['array', 10, ['char']]],
    'command_line' : [ 18, ['pointer', ['char']]],
    'piv' : [ 22, ['pointer', ['void']]],
}]
```

- Use close enough Kernel VTypes
- Shouldn't significally change, from one kernel to another
- Pray...

Find symbols

- Find `init_task`
- Check architecture
  - ELF's magic number statistics
  - Can be improved
- Convert to virtual addresses
  - Depends on architecture
  - Ex: x86 virtual address base -> 0xc0000000
- Handle missing symbols

Kallsyms

- Constructs a list of sections and symbols
- Creates `__kallsyms` section
- No relocs, only offset into section
  - -> Can be stored anywhere
- Contains:
  - Kallsyms header
  - Sections entries
  - Symbol entries
  - Strings

```
struct kallsyms_header {
  int           size;         /* Size of this header */
  ElfW(Word)    total_size;   /* Total size of kallsyms data */
  int           sections;     /* Number of section entries */
  ElfW(Off)     section_off;  /* Offset to first section entry */
  int           section_size; /* Size of one section entry */
  int           symbols;      /* Number of symbol entries */
  ElfW(Off)     symbol_off;   /* Offset to first symbol entry */
  int           symbol_size;  /* Size of one symbol entry */
  ElfW(Off)     string_off;   /* Offset to first string */
  ElfW(Addr)    start;        /* Start address of first section */
  ElfW(Addr)    end;          /* End address of last section */
};
```

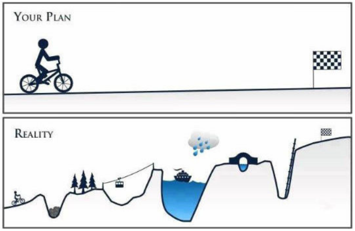
```
struct kallsyms_section {
  ElfW(Addr)  start;     /* Start address of section */
  ElfW(Word)  size;      /* Size of this section */
  ElfW(Off)   name_off; /* Offset to section name */
  ElfW(Word)  flags;     /* Flags from section */
};

struct kallsyms_symbol {
  ElfW(Off)  section_off;
  ElfW(Addr) symbol_addr;
  ElfW(Off)  name_off;
};
```

```
KallsymsHeader = Struct("KallsymsHeader",
           SLInt32("size"),
           ElfWWord("total_size"),
           SLInt32("sections"),
           ElfWOff("section_off"),
           SLInt32("section_size"),
           SLInt32("symbols"),
           ElfWOff("symbol_off"),
           SLInt32("symbol_size"),
           ElfWOff("string_off"),
           ElfWAddr("start"),
           ElfWAddr("end"))
```

Obstacles

- Some symbols might be missing
  - Tamper addresses validity
  - Volatility might need specific symbols
  - Need to "hardcode" some symbols



- Kallsyms not necessarily activated!

Questions?

Thanks!

- Contact: alpha@lse.epita.fr

References

1. The Volatility Foundation
2. The Art of Memory Forensics
3. Community contributions