

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Outline
 - Forensics
 - Memory
 - Windows
 - Linux
 - Linux : Live Memory Forensics
 - Symbols
 - Process
 - Tool : Draugr

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Tool : Draugr
 - URL :
 - <http://www.esiea-recherche.eu/~desnos/draugr/>
 - source code
 - paper (coming soon)
 - slides
 - videos

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- What is the forensic ?
 - Find (hidden) information
 - Photos/videos
 - Logs
 - Binaries
 - Data

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Disk Drive
 - Tools
 - Sleuthkit
 - PhotoRec

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- And the Memory
 - How to access it ?
 - How to dump/analyze it ?

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Memory Access
 - Drivers/LKM
 - Device
 - /dev/kmem (virtual memory)
 - /dev/mem (physical memory)
 - \Device\PhysicalMemory
 - Kernel bug
 - Firewire/DMA/Cold Boot Attack

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Windows
 - PTFinder
 - Memparser
 - Volatile Systems

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Linux
 - No real tools e.g. Windows
 - Volatile Systems (using specific kernel)

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Linux
 - How to access the memory ?
 - Worst case scenerio
 - No physical access
 - No specific kernel
 - No driver

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Linux
 - How to access the memory ?
 - The worst case
 - The hackers' way
 - /dev/(k)mem
 - File dump

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Linux
 - Example :
 - We would like information about process
 - name, pid, uid
 - sections

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Symbols
 - Pattern Matching
 - EXPORT_SYMBOL

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Symbols
 - Pattern Matching
 - Find useful functions :
 - kmalloc/vmalloc
 - kfree/vfree

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Symbols
 - Pattern Matching
 - Start from a known symbol
 - Ex : syscall

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Symbols
 - Pattern Matching
 - Example :
 - access_process_vm
 - read/write virtual memory of a process
 - (1) Read the source code of the function
 - (2) Find where this function is called
 - (3) If the symbol is known :
 - Exit
 - (4) Else -> (1)

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Symbols
 - Pattern Matching
 - Example :
 - access_process_vm
 - generic_ptrace_peekdata
 - ptrace_request
 - arch_ptrace
 - sys_ptrace

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)

desnos@nospam.esiea.fr

- sys_ptrace

```
Dump of assembler code for function sys_ptrace:
0xc012d234 <sys_ptrace+0>:  push  %edi
0xc012d235 <sys_ptrace+1>:  push  %esi
0xc012d236 <sys_ptrace+2>:  push  %ebx
0xc012d237 <sys_ptrace+3>:  mov   0x10(%esp),%edi
0xc012d23b <sys_ptrace+7>:  call  0xc02f21b8 <lock_kernel>
0xc012d240 <sys_ptrace+12>:  test  %edi,%edi
0xc012d242 <sys_ptrace+14>:  jne   0xc012d24d <sys_ptrace+25>
0xc012d244 <sys_ptrace+16>:  call  0xc012cd50 <ptrace_traceme>
0xc012d249 <sys_ptrace+21>:  mov   %eax,%esi
0xc012d24b <sys_ptrace+23>:  jmp   0xc012d2a8 <sys_ptrace+116>
0xc012d24d <sys_ptrace+25>:  mov   0x14(%esp),%eax
0xc012d251 <sys_ptrace+29>:  call  0xc012cd1d <ptrace_get_task_struct>
0xc012d256 <sys_ptrace+34>:  cmp   $0xffffffff00,%eax
0xc012d25b <sys_ptrace+39>:  mov   %eax,%ebx
0xc012d25d <sys_ptrace+41>:  mov   %eax,%esi
0xc012d25f <sys_ptrace+43>:  ja    0xc012d2a8 <sys_ptrace+116>
0xc012d261 <sys_ptrace+45>:  cmp   $0x10,%edi
0xc012d264 <sys_ptrace+48>:  jne   0xc012d26f <sys_ptrace+59>
0xc012d266 <sys_ptrace+50>:  call  0xc012d04e <ptrace_attach>
0xc012d26b <sys_ptrace+55>:  mov   %eax,%esi
0xc012d26d <sys_ptrace+57>:  jmp   0xc012d296 <sys_ptrace+98>
0xc012d26f <sys_ptrace+59>:  xor   %edx,%edx
0xc012d271 <sys_ptrace+61>:  cmp   $0x8,%edi
0xc012d274 <sys_ptrace+64>:  sete  %dl
0xc012d277 <sys_ptrace+67>:  call  0xc012d199 <ptrace_check_attach>
0xc012d27c <sys_ptrace+72>:  test  %eax,%eax
0xc012d27e <sys_ptrace+74>:  mov   %eax,%esi
0xc012d280 <sys_ptrace+76>:  js    0xc012d296 <sys_ptrace+98>
0xc012d282 <sys_ptrace+78>:  pushl 0x1c(%esp)
```

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- sys_ptrace

```
0xc012d286 <sys_ptrace+82>: mov    %edi,%edx
0xc012d288 <sys_ptrace+84>: mov    0x1c(%esp),%ecx
0xc012d28c <sys_ptrace+88>: mov    %ebx,%eax
0xc012d28e <sys_ptrace+90>: call  0xc010a46e <arch_ptrace>
0xc012d293 <sys_ptrace+95>: pop    %ecx
0xc012d294 <sys_ptrace+96>: mov    %eax,%esi
0xc012d296 <sys_ptrace+98>: lock decl 0x8(%ebx)
0xc012d29a <sys_ptrace+102>: sete  %al
0xc012d29d <sys_ptrace+105>: test  %al,%al
0xc012d29f <sys_ptrace+107>: je    0xc012d2a8 <sys_ptrace+116>
0xc012d2a1 <sys_ptrace+109>: mov    %ebx,%eax
0xc012d2a3 <sys_ptrace+111>: call  0xc0125e05 <__put_task_struct>
0xc012d2a8 <sys_ptrace+116>: call  0xc02f21db <unlock_kernel>
0xc012d2ad <sys_ptrace+121>: mov    %esi,%eax
0xc012d2af <sys_ptrace+123>: pop    %ebx
0xc012d2b0 <sys_ptrace+124>: pop    %esi
0xc012d2b1 <sys_ptrace+125>: pop    %edi
0xc012d2b2 <sys_ptrace+126>: ret
```

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- arch_ptrace

```
0xc010a769 <arch_ptrace+763>: pop    %ebx
0xc010a76a <arch_ptrace+764>: pop    %esi
0xc010a76b <arch_ptrace+765>: pop    %edi
0xc010a76c <arch_ptrace+766>: jmp    0xc012d3a5 <ptrace_request>
---Type <return> to continue, or q <return> to quit---
0xc010a771 <arch_ptrace+771>: mov    $0xffffffff,%eax
0xc010a776 <arch_ptrace+776>: pop    %ebx
0xc010a777 <arch_ptrace+777>: pop    %esi
0xc010a778 <arch_ptrace+778>: pop    %edi
0xc010a779 <arch_ptrace+779>: ret
```

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- ptrace_request

```
0xc012d40e <ptrace_request+105>:   cmp     $0x4201,%edx
0xc012d414 <ptrace_request+111>:   je      0xc012d4e2 <ptrace_request+317>
0xc012d41a <ptrace_request+117>:   jg      0xc012d426 <ptrace_request+129>
0xc012d41c <ptrace_request+119>:   cmp     $0x4200,%edx
0xc012d422 <ptrace_request+125>:   jne     0xc012d43e <ptrace_request+153>
0xc012d424 <ptrace_request+127>:   jmp     0xc012d465 <ptrace_request+192>
0xc012d426 <ptrace_request+129>:   cmp     $0x4202,%edx
0xc012d42c <ptrace_request+135>:   je      0xc012d4f4 <ptrace_request+335>
0xc012d432 <ptrace_request+141>:   cmp     $0x4203,%edx
0xc012d438 <ptrace_request+147>:   je      0xc012d56b <ptrace_request+454>
0xc012d43e <ptrace_request+153>:   mov     $0xffffffff,%esi
0xc012d443 <ptrace_request+158>:   jmp     0xc012d619 <ptrace_request+628>
0xc012d448 <ptrace_request+163>:   mov     %edi,%ecx
0xc012d44a <ptrace_request+165>:   mov     %esi,%edx
0xc012d44c <ptrace_request+167>:   mov     %ebx,%eax
0xc012d44e <ptrace_request+169>:   call   0xc012cca5 <generic_ptrace_peekdata>
0xc012d453 <ptrace_request+174>:   jmp     0xc012d45e <ptrace_request+185>
0xc012d455 <ptrace_request+176>:   mov     %edi,%ecx
0xc012d457 <ptrace_request+178>:   mov     %esi,%edx
0xc012d459 <ptrace_request+180>:   call   0xc012cc8c <generic_ptrace_pokedata>
```

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)

desnos@nospam.esiea.fr

- generic_ptrace_peekdata

```
gdb> disassemble generic_ptrace_peekdata
Dump of assembler code for function generic_ptrace_peekdata:
0xc012ccae <generic_ptrace_peekdata+0>: push   %esi
0xc012ccaf <generic_ptrace_peekdata+1>: mov    %ecx,%esi
0xc012ccb1 <generic_ptrace_peekdata+3>: push   %ebx
0xc012ccb2 <generic_ptrace_peekdata+4>: sub    $0x4,%esp
0xc012ccb5 <generic_ptrace_peekdata+7>: mov    %esp,%ecx
0xc012ccb7 <generic_ptrace_peekdata+9>: push   $0x0
0xc012ccb9 <generic_ptrace_peekdata+11>:        push  $0x4
0xc012ccbb <generic_ptrace_peekdata+13>: call   0xc016b435 <access_process_vm>
0xc012ccc0 <generic_ptrace_peekdata+18>: mov    0xc0177770,%edx
0xc012ccc5 <generic_ptrace_peekdata+23>: pop    %ecx
0xc012ccc6 <generic_ptrace_peekdata+24>: pop    %ebx
0xc012ccc7 <generic_ptrace_peekdata+25>: cmp    $0x4,%eax
0xc012ccca <generic_ptrace_peekdata+28>: jne    0xc012ccd8 <generic_ptrace_peekdata+42>
0xc012cccc <generic_ptrace_peekdata+30>: mov    (%esp),%eax
0xc012ccc9 <generic_ptrace_peekdata+33>: mov    %esi,%ecx
0xc012ccd1 <generic_ptrace_peekdata+35>: call   0xc0210bf8 <__put_user_4>
0xc012ccd6 <generic_ptrace_peekdata+40>: mov    %eax,%edx
0xc012ccd8 <generic_ptrace_peekdata+42>: mov    %edx,%eax
0xc012ccda <generic_ptrace_peekdata+44>: pop    %edx
0xc012ccdb <generic_ptrace_peekdata+45>: pop    %ebx
0xc012ccdc <generic_ptrace_peekdata+46>: pop    %esi
0xc012ccdd <generic_ptrace_peekdata+47>: ret
End of assembler dump.
```

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- And in a XML file :

```
<?xml version="1.0"?>
<parent id="access_process_vm">

<start name="access_process_vm">
  <next name="syscall">
    <name>sys_ptrace</name>
    <value>26</value>
  </next>

  <next name="call">
    <name>arch_ptrace</name>
    <value>5</value>
  </next>

  <next name="jmp">
    <name>ptrace_request</name>
    <value>-1</value>
  </next>

  <next name="call">
    <name>generic_ptrace_peekdata</name>
    <value>0</value>
  </next>

  <next name="call">
    <name>access_process_vm</name>
    <value>0</value>
  </next>
</start>
```

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Symbols
 - How to find the syscall table ?
 - `sys_call_table` not exported in 2.6.X
 - Via the IDT (see phrack 58 "Linux on-the-fly kernel patching without LKM" by **sd & devik**)
 - ?! doesn't work on virtual machine (wrong value for idt base)

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Symbols

- How to find the syscall table ?

- RTFSC

- In entry_32.S :

```
ENTRY(system_call)
RINGO_INT_FRAME           # can't unwind into user space anyway
pushl %eax                # save orig_eax
CFI_ADJUST_CFA_OFFSET 4
SAVE_ALL
GET_THREAD_INFO(%ebp)

                                # system call tracing in operation / emulation
/* Note, _TIF_SECCOMP is bit number 8, and so it needs testw and not testb */
testw $_TIF_WORK_SYSCALL_ENTRY, TI_flags(%ebp)
jnz syscall_trace_entry
cmpl $(nr_syscalls), %eax
jae syscall_badsys
syscall_call:
call *sys_call_table(,%eax,4)
movl %eax,PT_EAX(%esp)     # store the return value
```

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Symbols
 - How to find the syscall table ?

```
pouik@zion:/usr/src/linux/ > objdump -D vmlinux |grep -A 30 system_call|grep -A 8 -B 1 testw
c010391a: 21 e5                and    %esp,%ebp
c010391c: 66 f7 45 08 d1 01    testw $0x1d1,0x8(%ebp)
c0103922: 0f 85 00 01 00 00    jne   c0103a28 <syscall_trace_entry>
c0103928: 3d 4d 01 00 00      cmp   $0x14d,%eax
c010392d: 0f 83 4d 01 00 00    jae   c0103a80 <syscall_badsys>

c0103933 <syscall_call>:
c0103933: ff 14 85 20 4b 2f c0 call  *-0x3fd0b4e0(,%eax,4)
c010393a: 89 44 24 18        mov   %eax,0x18(%esp)

pouik@zion:/usr/src/linux/ > objdump -D /usr/src/linux/vmlinux | grep testw|grep "66 f7 45 08"
c0103835: 66 f7 45 08 d1 01    testw $0x1d1,0x8(%ebp)
c0103881: 66 f7 45 08 51 01    testw $0x151,0x8(%ebp)
c010391c: 66 f7 45 08 d1 01    testw $0x1d1,0x8(%ebp)
```

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Symbols
 - Pattern matching is a little bit tedious
 - EXPORT_SYMBOL (~3500)

```
/* For every exported symbol, place a struct in the __ksymtab section */
#define __EXPORT_SYMBOL(sym, sec) \
    extern typeof(sym) sym; \
    __CRC_SYMBOL(sym, sec) \
    static const char __kstrtab_##sym[] \
    __attribute__((section("__ksymtab_strings"), aligned(1))) \
    = MODULE_SYMBOL_PREFIX #sym; \
    static const struct kernel_symbol __ksymtab_##sym \
    __used \
    __attribute__((section("__ksymtab" sec), unused)) \
    = { (unsigned long)&sym, __kstrtab_##sym }

#define EXPORT_SYMBOL(sym) \
    __EXPORT_SYMBOL(sym, "")
```

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Symbols
 - kstrtab_symbol
 - Find the name of the symbol + '\0'
 - Get its address
 - Example: “init_task” + '\0' in the kernel string table

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Symbols
 - ksymtab_symbol
 - Search the address of the kernel string symbol (SYMBOL_kstrtab_symbol)
 - Subtract 4 from the address

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- How can you hide a process ?
 - sys_getdents, sys_read
 - vfs
 - /proc
 - linked list/Hash table
 - specific field in task_struct

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Process
 - Represented by the structure: `task_struct` (`linux/sched.h`)
 - No kernel headers
 - Independent of the version (2.6.X)
 - Build our own `task_struct` on the fly

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Process
 - Build task_struct
 - Depends on kernel options

```
#ifndef CONFIG_KEYS
    unsigned char jit_keyring; /* default keyring to attach requested keys to */
    struct key *request_key_auth; /* assumed request_key authority */
    struct key *thread_keyring; /* keyring private to this thread */
#endif
char comm[TASK_COMM_LEN]; /* executable name excluding path
    - access with [gs]et_task_comm (which lock
    it with task_lock())
    - initialized normally by flush_old_exec */

/* file system info */
int link_count, total_link_count;
#ifdef CONFIG_SYSVIPC
/* ipc stuff */
    struct sysv_sem sysvsem;
#endif
#ifdef CONFIG_DETECT_SOFTLOCKUP
/* hung task detection */
    unsigned long last_switch_timestamp;
    unsigned long last_switch_count;
#endif
/* CPU-specific state of this task */
    struct thread_struct thread;
/* filesystem information */
    struct fs_struct *fs;
/* open file information */
    struct files_struct *files;
```

```
#ifndef CONFIG_KEYS
    unsigned char jit_keyring; /* default keyring to attach requested keys to */
    struct key *request_key_auth; /* assumed request_key authority */
    struct key *thread_keyring; /* keyring private to this thread */
#endif
char comm[TASK_COMM_LEN]; /* executable name excluding path
    - access with [gs]et_task_comm (which lock
    it with task_lock())
    - initialized normally by flush_old_exec */

/* file system info */
int link_count, total_link_count;
#ifdef CONFIG_SYSVIPC
/* ipc stuff */
    struct sysv_sem sysvsem;
#endif
#ifdef CONFIG_DETECT_SOFTLOCKUP
/* hung task detection */
    unsigned long last_switch_timestamp;
    unsigned long last_switch_count;
#endif
/* CPU-specific state of this task */
    struct thread_struct thread;
/* filesystem information */
    struct fs_struct *fs;
/* open file information */
    struct files_struct *files;
```

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Process
 - Build task_struct:
 - Find offsets of interesting fields
 - Use the real first process

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Process
 - Build task_struct
 - Look for the first process
 - structure init_task
 - symbol “init_task”
 - pid 0
 - comm (16 bytes) : “swapper”

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Process
 - Build task_struct
 - Field: comm
 - ASCII string “swapper”

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Process
 - Build task_struct
 - Field: real_parent
 - real_parent == init_task

```
/*  
 * pointers to (original) parent process, youngest child, younger sibling,  
 * older sibling, respectively. (p->father can be replaced with  
 * p->real_parent->pid)  
 */  
struct task_struct *real_parent; /* real parent process */  
struct task_struct *parent; /* recipient of SIGCHLD, wait4() reports */
```

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Process
 - Build task_struct
 - Field: tasks
 - Before this field:
 - Two list_head (fields (prev == next) && > 0xc0000000)

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Process
 - Build task_struct
 - Field: pid
 - Next after tasks (% empty struct (old kernel))

```
struct list_head tasks;

struct mm_struct *mm, *active_mm;

/* task state */
struct linux_binfmt *binfmt;
int exit_state;
int exit_code, exit_signal;
int pdeath_signal; /* The signal sent when the parent dies */
/* ??? */
unsigned int personality;
unsigned did_exec:1;
pid_t pid;
pid_t tgid;
```

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Process
 - Build task_struct
 - Fields: uid/gid
 - list of zeros

```
/* process credentials */
uid_t uid,euid,suid,fsuid;
gid_t gid,egid,sgid,fsuid;
struct group_info *group_info;
kernel_cap_t cap_effective, cap_inheritable, cap_permitted, cap_bset;
```

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Process
 - Build the list
 - Now it's easy:
 - walk through the linked list

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Process
 - Build linked list `task_struct`:
 - We will not see hidden process by “linked list attack”

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Process
 - We have the `task_struct`
 - Bruteforce the memory
 - field “comm”: find ascii character
 - field “state”: 0 or 1
 - fields “pid/tpid”: a valid integer (0 ... 65535)
 - fields “stack/mm/activemm”: 0 or $\geq 0xc0000000$

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Process
 - We have the `task_struct`
 - Bruteforce the memory
 - We can find deleted processes
 - Many forks

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Process

- All fields in `task_struct`

- And the content of the binary ?

- We would like to access to all sections

- `.text`

- `.data`

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Process

- User land paging, why not ?

- access_process_vm

- **get_user_page**

- pgd_offset_gate (pgd_offset)
 - pud_offset
 - pmd_offset
 - pte_offset_kernel
 - pmd_page_vaddr
 - pte_index
 - pte_page

- **kmap**

- __va

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Process
 - User land paging, why not ?
 - We have all the information:
 - We have the address
 - vma_area_struct
 - example : 0x08048000
 - We have the field “mm”

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Process
 - Field “mm” (struct mm_struct)
 - vma_area_struct
 - pgd

```
struct mm_struct {
    struct vm_area_struct * mmap;           /* list of VMAs */
    struct rb_root mm_rb;
    struct vm_area_struct * mmap_cache;    /* last find_vma result */
    unsigned long (*get_unmapped_area) (struct file *filp,
                                        unsigned long addr, unsigned long len,
                                        unsigned long pgoff, unsigned long flags);
    void (*unmap_area) (struct mm_struct *mm, unsigned long addr);
    unsigned long mmap_base;               /* base of mmap area */
    unsigned long task_size;               /* size of task vm space */
    unsigned long cached_hole_size;        /* if non-zero, the largest hole below free_area_cache */
    /
    unsigned long free_area_cache;         /* first hole of size cached_hole_size or larger */
    pgd_t * pgd;
    atomic_t mm_users;                     /* How many users with user space? */
    atomic_t mm_count;                     /* How many references to "struct mm_struct" (users count as 1) */
    int map_count;                          /* number of VMAs */
};
```

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Process

- PGD (Page Global Directory): we have the field `pgd` in `mm`

- `pgd_offset(mm, address) ((mm)->pgd + pgd_index((address)))`
- `pgd_index(address) (((address) >> PGDIR_SHIFT) & (PTRS_PER_PGD - 1))`
- traditional i386 two-level paging structure:

- `PGDIR_SHIFT 22`
- `PTRS_PER_PGD 1024`

- With Python:

- `pgd = mmpgd + ((tmp >> 22) & (1024 - 1)) * 4`

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Process

- PTE (Page Table Entry): We have the pgd offset

- pgd -> pud -> pmd

- **pte_offset_kernel**(dir, address):

- `((pte_t *)pmd_page_vaddr(*(dir)) + pte_index((address)))`

- **pmd_page_vaddr**(pmd):

- `((unsigned long)__va(pmd_val((pmd)) & PTE_PFN_MASK)`

- **pte_index**(address):

- `((address) >> PAGE_SHIFT) & (PTRS_PER_PTE - 1)`

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Process
 - PTE: We have the pgd offset
 - PTE_PFN_MASK 0xffff000
 - PAGE_SHIFT 12
 - PTRS_PER_PTE 1024

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Process

- PTE: We have the pgd offset

- With Python:

- `dirp = unpack("<L", self.read(pgd, 4))[0]`
 - `off1 = (dirp + 0xc0000000) & PTE_PFN_MASK`
 - `off2 = (tmp >> PAGE_SHIFT) & (PTRS_PER_PTE - 1)`
 - `pte = off1 + off2 * 4`

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Process

- Page: we have the pte

- **pte_page**(pte):

- pfn_to_page(pte_pfn(pte))

- **pte_pfn**(pte_t pte)

- pte_val(pte) & PTE_PFN_MASK) >> PAGE_SHIFT

- **pfn_to_page** __pfn_to_page

- **__pfn_to_page**(pfn)

- (mem_map + ((pfn) - ARCH_PFN_OFFSET))

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Process

- Page: we have the pte

- With Python:

- `pte = unpack("<L", self.read(pte, 4))[0]`
 - `pte = (pte & PTE_PFN_MASK) >> PAGE_SHIFT`
 - `mem_map = 0xc1000000`
 - `page_addr = mem_map + pte * 0x20 # (sizeof(struct page))`

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Process

- Page

- And to finish

- `void *kmap(struct page *page)`

- `__va(page_to_pfn(page_at_addr) << PAGE_SHIFT)`

- `page_to_pfn __page_to_pfn`

- `__page_to_pfn(page) ((unsigned long)((page) - mem_map) + ARCH_PFN_OFFSET)`

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Process

- Page

- With Python:

- $ptp = (page_addr - mem_map) / 0x20$

- $page = 0xc0000000 + ((ptp + 0) \ll PAGE_SHIFT)$

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Process

- Access page:

- Valid for 32 bits (pgd == pud == pmd)
 - Can be done on 64 bits
 - Valid only for ram \leq 896 Mo (ZONE_NORMAL)
 - Use access_process_vm for $>$ 896 (ZONE_HIGHMEM)
 - So we must write in the kernel memory :(

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- New tool : Draugr
 - Written in pure python
 - Easy access to the memory:
 - Live with /dev/(k)mem
 - Memory dump
 - Find symbols (pattern matching, export_symbol)
 - Find all tasks (linked list, bruteforce)
 - Fields

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Draugr
 - Download @
 - <http://www.esiea-recherche.eu/~desnos/draugr/>
 - Demo!

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

Many thanks for your attention!
Have you any question... ?

Live Memory Forensics

Desnos Anthony (ESIEA SI&S)
desnos@nospam.esiea.fr

- Reconstruction of a binary
 - Use `vma_area_struct`
 - Process dumper