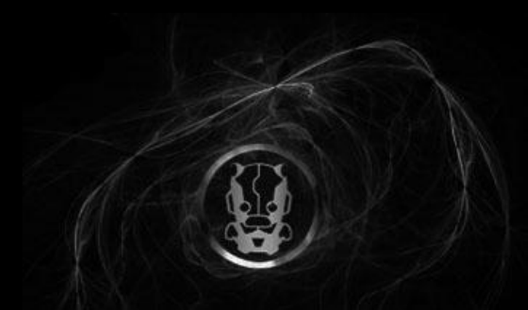
IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

Easy local Windows Kernel exploitation

Cesar Cerrudo
CTO at IOActive Labs



Introduction

- Windows kernel exploitation is still kind of a dark art
  - –Not many good and reliable kernel exploits available
- Write “what” “where” exploitation
  - –Some techniques are not reliable and/or complicated
  - –Few techniques are generic and work across different Windows versions
  - –Sometimes “what” is a fixed value, sometimes is NULL, sometimes is just one or two bytes, sometimes you can only increment or decrement the value on “where”, etc.
    - No generic technique for hard to exploit vulnerabilities
  - –Always run code on kernel mode

All started with a good paper

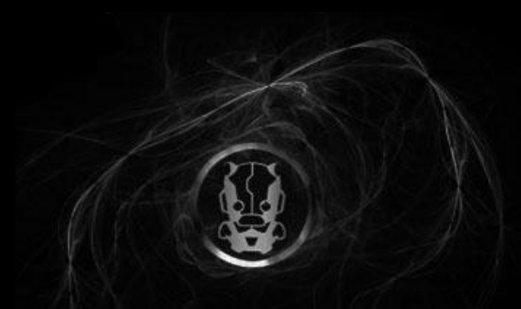
- On January 2010 Matthew “j00ru” Jurczyk and Gynvael Coldwind published “GDT and LDT in Windows kernel vulnerability exploitation” [1]
  - –NtQuerySystemInformation(SystemHandleInformation) to get kernel address of KPROCESS

```
typedef struct _SYSTEM_HANDLE_TABLE_ENTRY_INFO {
   USHORT UniqueProcessId;
   USHORT CreatorBackTraceIndex;
   UCHAR ObjectTypeIndex;
   UCHAR HandleAttributes;
   USHORT HandleValue;
   PVOID Object;
   ULONG GrantedAccess;
} SYSTEM_HANDLE_TABLE_ENTRY_INFO, *PSYSTEM_HANDLE_TABLE_ENTRY_INFO;
```

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

Making exploitation easier

- What if we can remove ACLs of almost any Windows object?
- What if we can set any privilege to a process token?
- What if we can replace a process token?
- It’s possible to do any of the above with just one write to kernel and without running code in kernel mode
- Why do you want a System shell?



Making exploitation easier

- Windows object ACL

kd> dt nt!_OBJECT_HEADER

+0x000 PointerCount : Int4B **//keeps reference counting**

+0x004 HandleCount : Int4B

+0x004 NextToFree : Ptr32 Void

+0x008 Lock : _EX_PUSH_LOCK

+0x00c TypeIndex : UChar

+0x00d TraceFlags : UChar

+0x00e InfoMask : UChar

+0x00f Flags : UChar

+0x010 ObjectCreateInfo : Ptr32 _OBJECT_CREATE_INFORMATION

+0x010 QuotaBlockCharged : Ptr32 Void

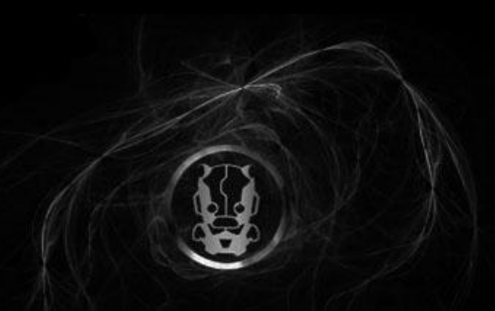
+**0x014 SecurityDescriptor** : Ptr32 Void **//Body -0x4 x86 or -0x8 x64 >=Win2K**

+0x018 Body : _QUAD **//Here starts the object structure**

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

Making exploitation easier

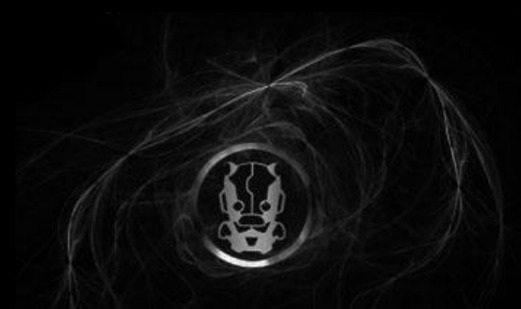
- Nulling out ACLs
  1. Get target object (process, thread, etc.) kernel address using NtQuerySystemInformation(SystemHandleInformation)
  2. Write NULL to [object address–0x4] on x86 or [object address – 0x8)]on x64
  3. Manipulate the target object (inject code, read memory, etc.) to escalate privileges from user mode.
  – Demo



Making exploitation easier

- Token privileges (Windows>=Vista)

```
typedef struct _TOKEN
   {
...  (same offset and structure on Vista, Win7, Win2008 R1 & R2 x86 x64)
/*0x040*/    typedef struct _SEP_TOKEN_PRIVILEGES
                  {
                     UINT64 Present;
/*0x048*/            UINT64 Enabled;
                     UINT64 EnabledByDefault;
                  } SEP_TOKEN_PRIVILEGES, *PSEP_TOKEN_PRIVILEGES;
...
  }TOKEN, *PTOKEN;
```



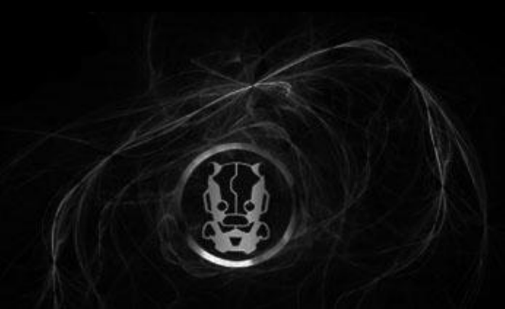
Making exploitation easier

- Token privileges (Windows XP, 2003)

```
lkd> dt _TOKEN
   ...
   +0x074 Privileges   Ptr32 _LUID_AND_ATTRIBUTES   //points to  VariablePart
   ...
   +0x0a0 VariablePart     : Uint4B

lkd> dt _LUID_AND_ATTRIBUTES
   +0x000 Luid             : _LUID
   +0x008 Attributes       : Uint4B          //0x0 disabled, 0x1 enabled by default, 0x2 enabled

lkd> dt _LUID
   +0x000 LowPart          : Uint4B          //number identifying a privilege
   +0x004 HighPart         : Int4B
```



Making exploitation easier

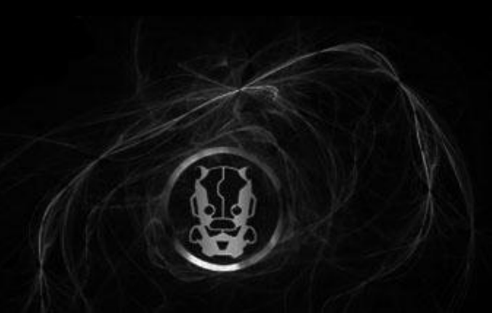
- Powerful privileges
  - –Debug programs
  - –Take ownership
  - –Restore files and directories
  - –Impersonate a client after authentication
  - –Load and unload device drivers
  - –Create a token object
  - –Act as part of the operating system, etc.

Making exploitation easier

- Enabling privileges
  1. Get process primary token and then search its kernel address using NtQuerySystemInformation(SystemHandleInformation)
  2. Write 0xffffffff or the value you can to [_TOKEN+0x48] to enable privileges in the process primary token on Win>=Vista)

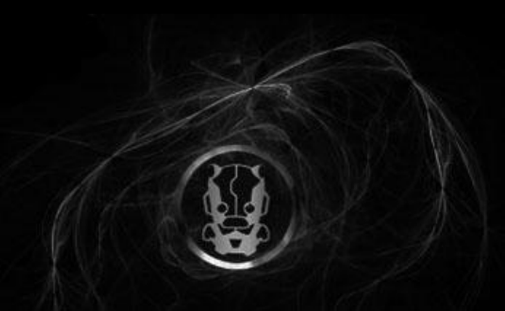
     Or

     Write some value (0x14 to enable debug privilege) to [_TOKEN+0xA0] on WinXP or 2003
  3. Perform privileged actions depending on enabled privileges



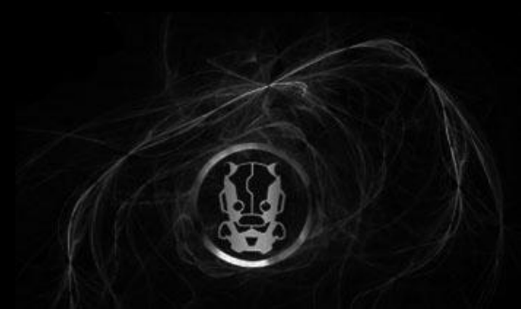
Making exploitation easier

- Exploit for Tarjei Mandt kernel vulnerability, use after free (Windows>=Vista)
  - –dec dword ptr [eax+4] // we can only control eax
  - –[_TOKEN+0x48] ==0x800000 (Enabled privs) by default on Win7
  - –0x800000==100000000000000000000000b just one priv enabled by default (SeChangeNotifyPrivilege)
  - –0x800000-0x1==0x7fffff==11111111111111111111111b lots of privs enabled
  - –Demo



Making exploitation easier

- Exploit for Tarjei Mandt kernel vulnerability, use after free (Windows XP or 2003)
  - –dec dword ptr [eax+4] // we can only control eax
  - –[_TOKEN+0xA0] ==0x15 (Audit privilege)
  - –0x15-0x1==0x14==Debug privilege
  - –Can do multiples dec to get to 0x9 (Take ownership privilege) or others



Making exploitation easier

- Process primary token

typedef struct _EPROCESS *(Win7 RTM x86)*

{

…//this offset changes in some Win versions but stable on different service pack level)

/*0x0F8*/ **struct _EX_FAST_REF Token;** ***//0x4 bytes x86 or 0x8 bytes x64)***

...

}EPROCESS, *PEPROCESS;

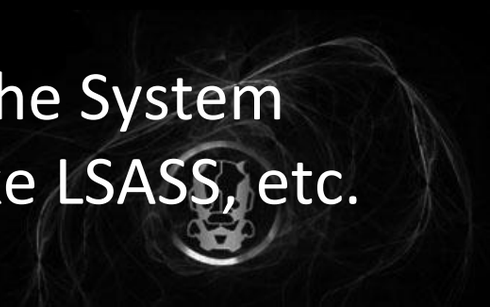
kd> dt nt!_OBJECT_HEADER

+0x000 PointerCount : Int4B

…

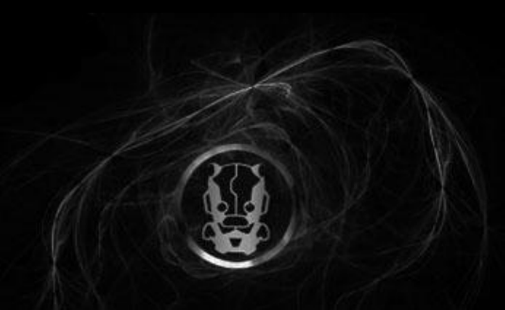
Making exploitation easier

- Replacing process token
  1. Get System Identity token by hooking NtOpenThreadToken() and calling MsiInstallProduct(), then get object kernel address using NtQuerySystemInformation(SystemHandleInformation)
  - If multiple writes
    2. Increase ref count with first write to PointerCount [_TOKEN – 0x18] on x32 or [_TOKEN – 0x30] on x64 >= Win 2000
    3. Second write to replace Token on _EPROCESS with System token
  - If one write
    2. Replace Token on _EPROCESS
    3. After elevation and before exploit finishes duplicate the System token twice in other process that never terminates like LSASS, etc.



Conclusions

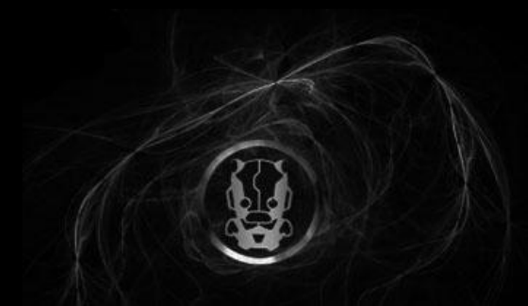
- Exploiting some kernel vulnerabilities is really easy with help of NtQuerySystemInformation(SystemHandleInformation)
  - –Allows to quickly build reliable and multi version kernel exploits even when the vulnerability is “difficult” to exploit
- These are just some ways I found and researched but there should be other ones that maybe allow even more easier exploitation
- You don’t always need a System shell



References & Thanks

- 1 - http://j00ru.vexillium.org/?p=290#more-290

- Thanks to Ruben Santamarta and Tarjei Mandt for feedback and PoCs



Fin

- Questions?

- Gracias.

- E-mail: ccerrudo>at<ioactive>dot<com
- twitter: @cesarcer