



# VBootKit 2.0a - Attacking Windows 7 via Boot Sectors

**Nitin Kumar**  
Security Researcher

nitin@nvlabs.in

**Vipin Kumar**  
Security Researcher

vipin@nvlabs.in

Copyright © The OWASP Foundation  
Permission is granted to copy, distribute and/or modify this document under  
the terms of the OWASP License.

**OWASP**

20/11/2009

**The OWASP Foundation**

<http://www.owasp.org>

# VBootKit 2.0a - Attacking Windows 7 via Boot Sectors

Securitybyte & OWASP AppSec Asia  
Conference 2009

2009-11-20

**Nitin Kumar**

Security Researcher

[nitin@nvlabs.in](mailto:nitin@nvlabs.in)

**Vipin Kumar**

Security Researcher

[vipin@nvlabs.in](mailto:vipin@nvlabs.in)

## What we do ?

- Analysing malware
- Code Reviewing
- Network PenTests

and also, a bit of this and a bit of that.



# Presentation outline

- Introduction to Bootkits
- Windows 7 boot process (x86 )
- Vbootkit 2.0a architecture and working
- Vbootkit 2.0a Payloads aka shell-codes
  - Remote Command & Control protocol
  - Privilege escalation of programs
- Demo
  - Remote Keylogger
- Demo
  - Login without passwords
- Demo
- Vbootkit and DRM
- Question time and Thanks



# Introduction to Bootkits

- Bootkits are rootkits in which first point of control is during the boot process such MBR , VBR etc
- Bootkits are almost impossible to detect
- Bootkits can be used to avoid all protections of an OS, because OS consider that the system was in trusted stated at the moment the OS boot loader took control.
- Customized MBR/boot sectors are used for both to keep themselves in control and also to spread
- Age-old boot sector attacks are back in picture, ready to grab control of your system.



## Windows 7(x86) Boot Process

- MBR loads NT Boot Sector ( 8 KB in size, currently only 5 KB is used). NT boot sector has the ability to read FAT32 and NTFS. It finds and loads a file BOOTMGR.EXE from the system32 or system32/boot directory at 2000h:0000h or 0x20000 to be exact.
- BOOTMGR.EXE has a 16 header prepended to itself. This 16 bit header checks the checksum of embedded executable and maps it at 0x400000. This executable doesn't have any external dependencies and is self-contained



## Windows 7(x86) Boot Process

- Execution of BOOTMGR starts in 32 bits in **BmMain** function. It no longer verifies itself. Vista (32 bit) used to verify it's digital signature. However, (this change is welcome). NO software in world can detect changes to itself and declare that the results can be relied upon.
- After this, it checks for hibernation state, if it's found, it loads winresume.exe and gets done
- It then mounts BCD database and enumerates boot entries, settings etc

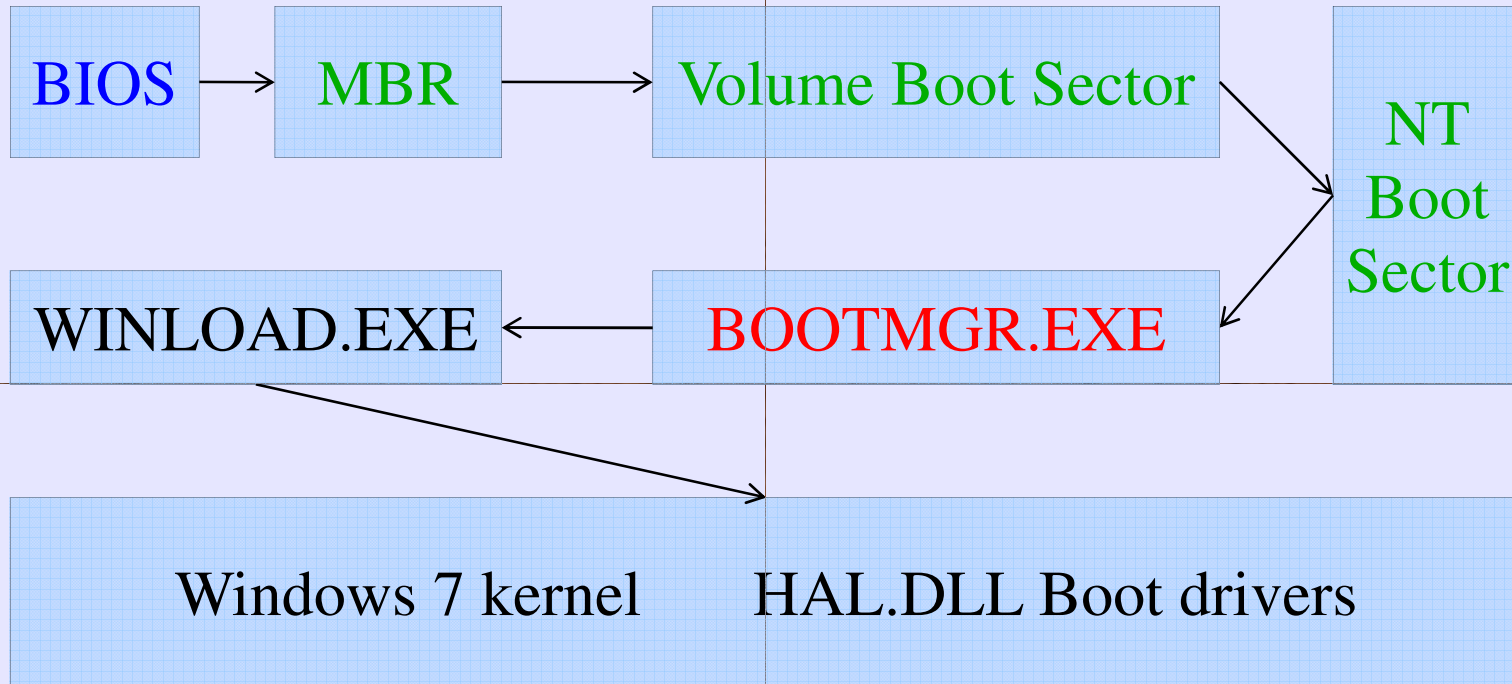


## Windows 7(x86) Boot Process

- After user selects a boot entry, it is launched using **BmLaunchBootEntry** with added switches. (However, in case of 64 bit OS, CPU is changed to 64 bit mode else no mode change occurs ) just before jumping to WINLOAD.EXE
- Now WINLOAD.EXE is loaded, it loads NTOSKRNL.EXE, HAL.DLL, dependencies, boot drivers after loading SYSTEM registry hive
- Creates a PsLoadedModuleList & LOADER\_PARAMETER\_BLOCK structure which contains memory map, options list etc
- Control is then transferred to kernel using **OslArchTransferToKernel** after stopping boot debugger



# Summary of Booting Process



Legend CPU Mode

Blue > UNKNOWN

Green > 16 bit

Red > 32 bit

Black > 64 bit in case of 64 bit OS else 32 bit



# Windows 7 Kernel Startup

NTOSKRNL uses 2 phases to initialize system

- First phase (phase 0) initializes the kernel itself

- Calls HalInitialiseBios

- Inits Display driver

- starts Debugger

- Calls KiInitializeKernel

- Second phase (phase 1) initializes the system

- Phase1 InitializationDiscard

- HalInitSystem

- ObInitSystem

- Sets boot time bias for ASLR

- PsInitialSystemProcess

- StartFirstUserProcess ( starts SMSS.EXE)



## But what is Minwin and minkernel ???

- Minwin is Microsoft's internal project kinda stuff which is how small and independent can the Windows kernel(and related core components) can be made but at the same time keep it useful and working. A severely stripped down version (almost 20 MB) of Windows 7 was able to run a http server ( but no GUI). Windows 7 takes several GIGs to even start on my PC.
- Minkernel is the step towards Minwin so as to make the kernel more streamlined and small and it actively started from Windows 7 onwards.It doesn't mean rewrite of the kernel, it's just making it more stream-lined.



# Vbootkit 2.0a



## Objective of Vbootkit 2.0a

- The objective is to get the Windows 7 (x86) running normally with some of the our changes done to the kernel.
- Also, the Vbootkit 2.0a should pass through all the security features implemented in the kernel without being detected namely Driver signing.
- No files should be patched on disk, it should run complete in memory to avoid later on detection.



Basically, we follow a very simple algorithm for vbootkit

- Hook INT 13 ( for disk reads)
- Keep on patching files as they load
- Hook onto next stage
- Repeat the above process, till we reach the kernel, then sit and watch the system carefully

Vbootkit 2.0 featured a major design change,

Instead of patching protections, we try to take control in such a place, so as the trigger never occurs but we get the control in our own hands.



## Vbootkit – Functional workout

- Our code gains execution from the CD-Rom, PXE relocates ourselves to 0x9e000.
- Hook INT 13 .
- The hook searches every read request for a signature, if the signature matches it executes its payload.
- Vbootkit 2.0 reads MBR and starts normal boot process with INT 13 hook installed
- When the NT boot sector loads BOOTMGR.EXE , our hooks finds the signature and executes the payload
- The signature is last 8 bytes from bootmgr.exe excluding zeroes
- The payload patches bootmgr.exe at single location



## Vbootkit – Functional workout (cont.)

- Now, the 16 bit header starts execution and we face the first security check. It's a simple checksum protection stored in the PE Header. However, we never modified BOOTMGR.EXE, the check passes successfully
- Now the bootmgr is mapped at 0x400000 and just before execution is transferred to BOOTMGR.EXE, Vbootkit gains control
- We apply a single patch to BOOTMGR.EXE and give control back.
- BOOTMGR.EXE which used to verify itself earlier, now no longer verifies itself, thus making our job a lil easier.



## Vbootkit – Functional workout (cont.)

- Now bootmgr loads its resources and displays boot menu.
- After the user , selects an Entry to boot, the bootmgr calls `BllmgLoadPEImageEx` to load `WINLOAD.EXE`. It also verifies the digital signature of the file.
- The location is executed just before `BOOTMGR.EXE` transfer execution to `WINLOAD`.
- Vbootkit puts a patch in `WINLOAD.EXE` at a similar place as in `BOOTMGR.EXE`. The patch location is `OslArchTransferToKernel` for both `BOOTMGR` and `WINLOAD`



## Vbootkit – Functional workout (cont.)

• Winload completely trusts BOOTMGR.EXE that it has provided a safe environment, so it validates all the options, maps SYSTEM registry hive, loads boot drivers, prepares a structure called loader block. This loader block contains entry of all drivers loaded, their base addresses also also contains the memory map of the system( which block is used).It also passes the famous option list, which is processed by kernel to set some features such as enabling of debugger,DEP ( Data Execution Policy),patchguard etc



## Vbootkit – Functional workout (cont.)

- Our Winload detour takes control just before the control is passed to kernel. This transfer of control takes place in a function called **OslArchTransferToKernel**
- This detour relocates vbootkit once again to blank space in kernel memory which has read/write access, and applies an 20 byte detour to a function called **StartFirstUserProcess**. It's in the INIT section of kernel. It allocates memory, relocates Vbootkit 2.0 to newly allocated space and jumps to new location



## Vbootkit – Functional workout (cont.)

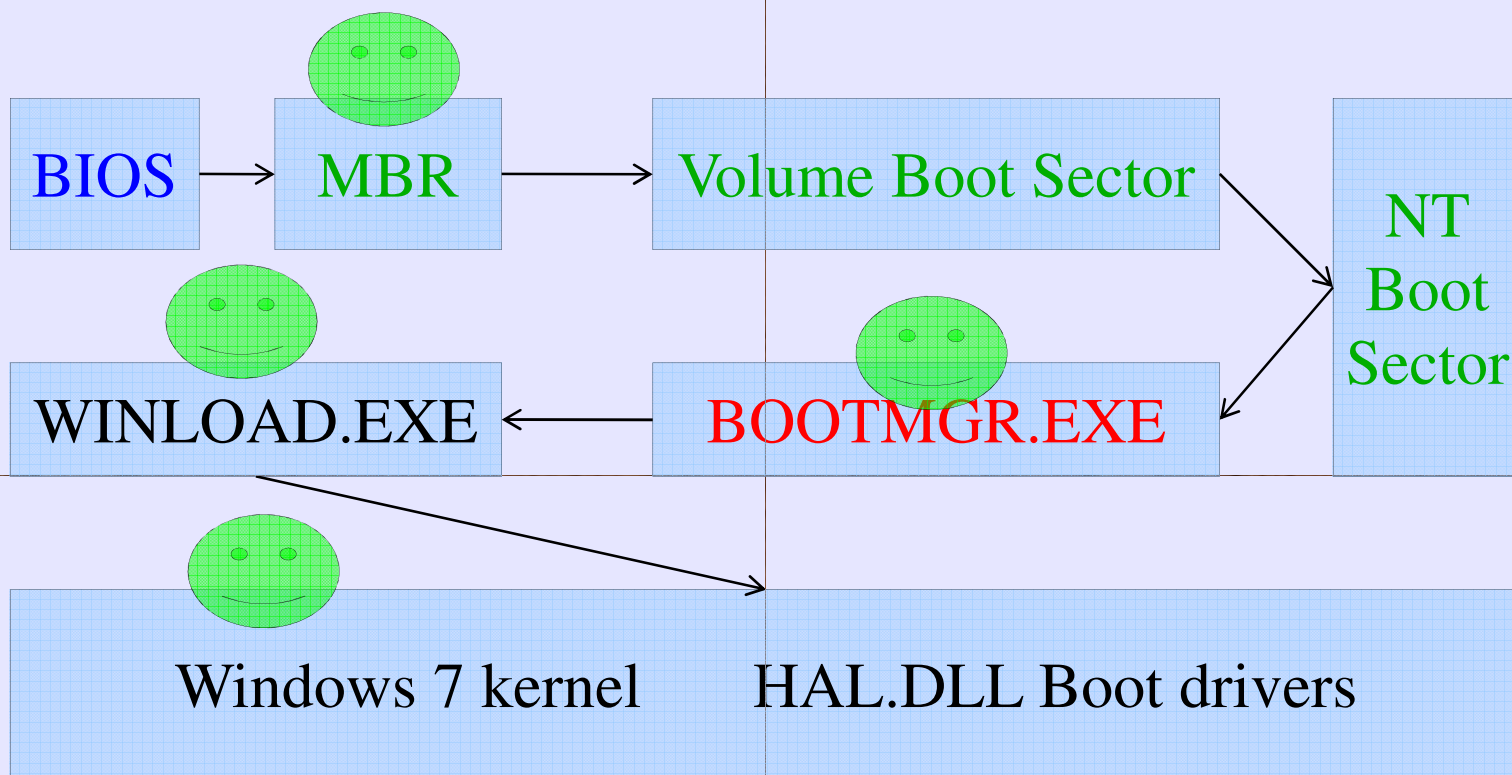
■ First job after obtaining control is to FIX the kernel, to avoid any triggers and to continue gracefully. We also create a number of kernel threads which will satisfy the payloads then return back.

The tasklist done above is

- Fix the kernel
  - Load Driver
    - Create shell code threads
    - Allocate workspace buffers
    - Hook PING requests
    - Install keylogger
- Return back



# Summary of detours applied by Vbootkit 2.0



Legend CPU Mode

Blue > UNKNOWN

Green > 16 bit

Red > 32 bit

Black > 64 bit in case of 64 bit OS else 32 bit



## Difference between vbootkit 2.0 and 2.0a

### Vbootkit 2.0

- 64 bit
- Written in x86
- Small ( just 3 KB)

### Vbootkit 2.0a

- 32 bit
- Written in x86 and companion driver
- Not very small ( 8 KB 6 KB due to driver)



Payloads



# Remote Command & Control



# Remote Command & Control protocol

Vbootkit 2.0 uses a very simple protocol to communicate with remote clients. Communication is done over a PING packets.

The protocol is based on a request response model. After a request is made, if it cannot be satisfied immediately, it is put in delayed mode, In delayed mode, the response will be sent in the next packet



# Remote Command & Control protocol

Command Byte

Response Code Byte

8 Byte signature

Data

Command Packet

PING Packet

Command Byte

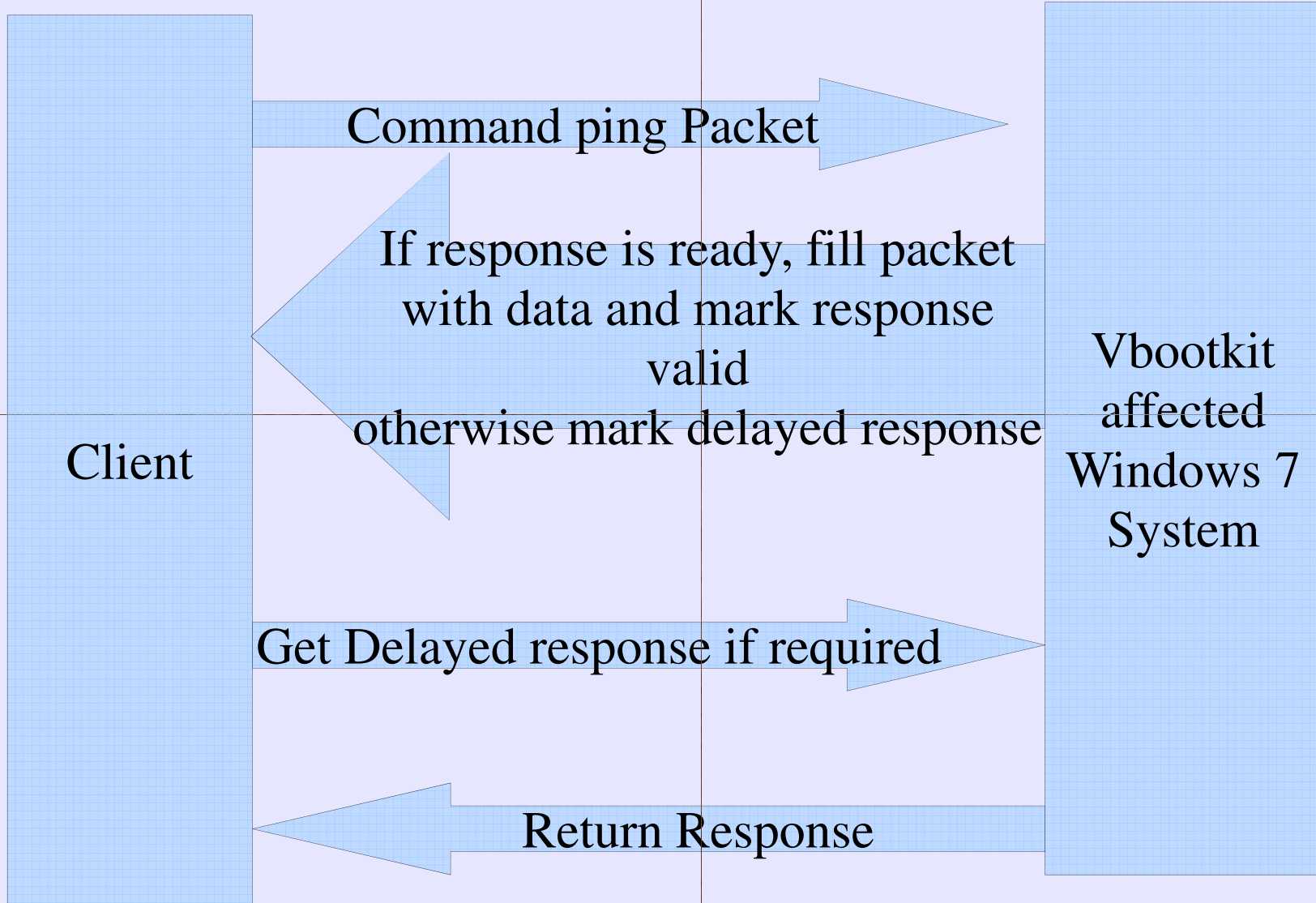
Response Code Byte

Response Data

Response Packet



# Remote Command & Control protocol



Communication FLOW

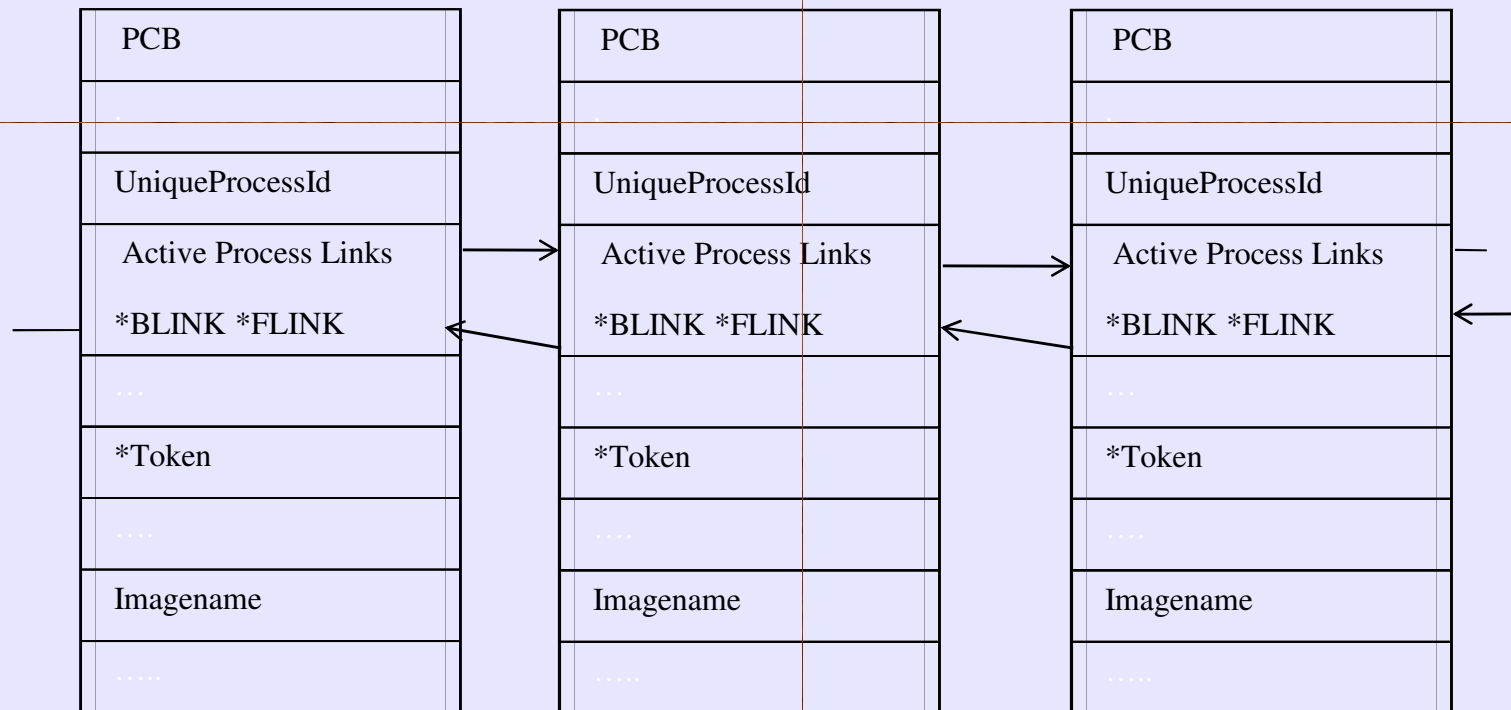


# Privilege Escalation



# Privilege Escalation

All the process running in a system are maintained by the kernel using a structure called EPROCESS. These structures are linked together using a doubly linked list structure



## Token structure

```
typedef struct _EX_FAST_REF
{ union {
    PVOID Object;
    ULONG RefCnt: 3;
    ULONG Value;
};} EX_FAST_REF, *PEX_FAST_REF;
```

It is part of the Object Manager Fast-Referencing implementation (ObFastReferenceObject, etc). It allows the kernel to encode the reference count as a pointer bias, so that the object is actually only truly "referenced" once with the Object Manager, and every other additional time inside the EX\_FAST\_REF structure itself. The bias towards the 8-byte alignment is the number of fast references an object can have. When the last fast-reference is removed, the Object Manager actually gets the real ObDereferenceObject call



## How Vbootkit 2.0 does privilege escalation?

- Vbootkit first finds SERVICES.EXE, since it is part of OS and always runs with SYSTEM level privileges and stores its token
- After receiving command, It scans the process list for running Command Prompts( CMD.EXE), All running instances are given the service token of SERVICES.EXE thus giving SYSTEM level privileges.



# Privilege Escalation Demonstration Time



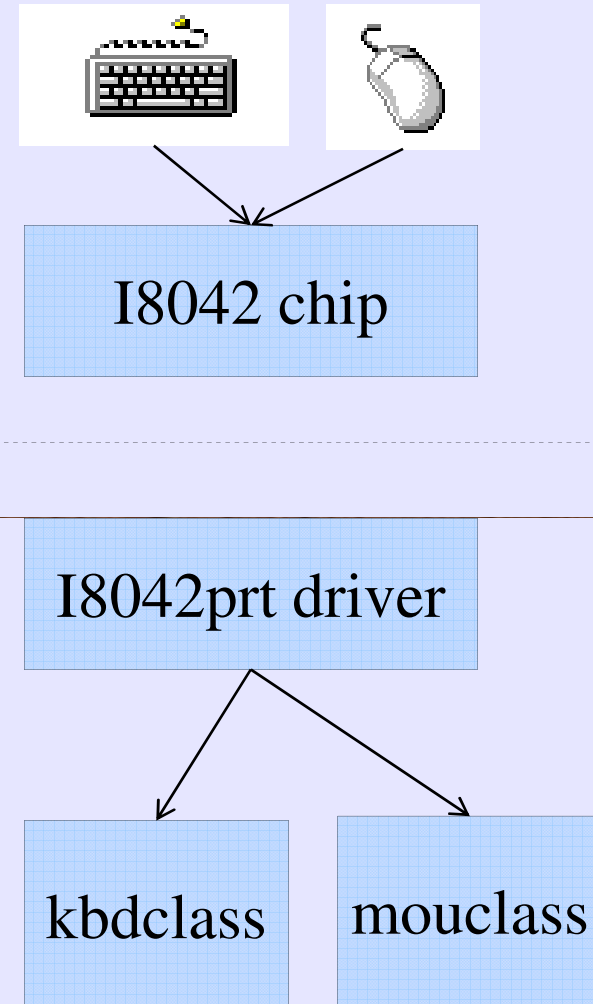
# Keyboard & mouse data flow in Windows

I8xGetByteAsynchronous function in i8042 driver is used to fetch data from the chip.

It's declaration is

**I8xGetByteAsynchronous(  
char device, char\* outputbyte)**

Keyboard device ID: 1



Hooking at this place has many advantages

- Ability to get keyboard data
- Ability to get mouse data
- Ability to inject keyboard keys
- Ability to inject mouse data
- Ability to modify data such making 'Z' to 'A' and so on
- Single hook lets you do many things

However, there are disadvantages also

- More complex
- Instead of giving you keys, it gives you scan-codes, both make and break sequence
- You have to map the keys yourself



## Implementation in Vbootkit 2.0

Implementation is basic, the Vbootkit 2.0 internally uses a 256 byte cyclic buffer to store keyboard data. Currently mouse data is ignored but it can also be captured and remote transferred easily just as keyboard

Whenever Vbootkit sees a Keyboard Buffer Request Command, it copies the buffer to the PING packet as response and continues capturing keystrokes

Returned data is converted to key by the client and is currently a very basic implementation( only done as POC)



# Remote Keylogger Demonstration



# Password Removal



# Security Accounts Manager (SAM)

SAM stores users' passwords in a hashed format (in an LM hash and an NTLM hash). It has been reverse engineered from time to time to obtain the hashes and remove passwords.

It's structure is not documented. However, some fields are used

- .



# Windows 7 Password Checking algorithm

This is real simple.

- Check if the NT Hash len field contains 4 or 0, this signifies password field is blank
- If the password hash length is not blank, fetch the hash
- Calculate hash from user supplied password
- Compare both the hash
- if equal then login the user
- Otherwise display wrong password message





## Vbootkit 2.0 Password removal

- Vbootkit on receiving command to remove password will go through all the users
- And check whether they have a null password
- If they have null password, skip the user
- Otherwise, null the password length
- Keep repeating until all the users have been processed



# Recovering Passwords

- After the user's job is done, Vbootkit 2.0 should put everything in place, so as the original user doesn't panic
- This is done by putting the original nthash length field to its original value.
- This puts the user's password back in place

The whole setup lets us login into the system without any password, get the job done and then put everything back in place.



controlling passwords demo



Is this enough ??



**of course, NOT**

Vbootkit 2.0a currently supports fetching the registry, one value at a time but right now we support fetching only strings.

Slowly but steadily this lets us build most of the registry of the remote system.

Key modification is also supported, but the feature is broken right now.( and yes, we are not fixing it, because this is just a POC )



## A quick question ?

What is the size of Vbootkit 2.0 ??  
**just 3 KB**

What is the current size of Vbootkit 2.0 ?  
**1.5KB(vbootkit+driver loader) + 6.5KB (driver) = 8 KB**



Vbootkit 2.0 ( even older Vbootkit 1.0 ) can be used to capture streams. This is because Vbootkit runs in completely undetected form( less no. of payloads definitely means even less chances of detection).

Windows Vista and Windows 7 Protected Media Path(PMP) model is completely violated by Vbootkit.

To Capture, all audio streams (in WAV format), Vbootkit has to put a hook on the CopyTo function in the right place and the DRM is no longer standing in your way !!! This is true for all known and unknown audio DRM implementations.

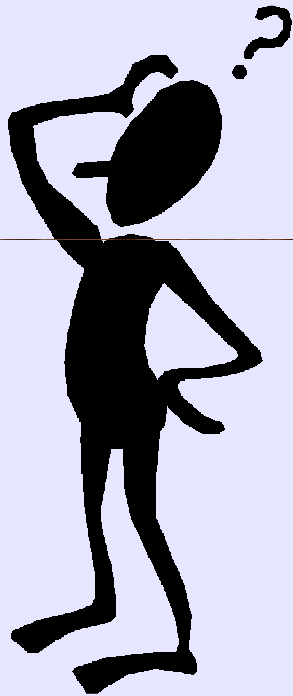


# References

- Brown, Ralf. Ralf Brown's Interrupt List. <http://www.cs.cmu.edu/~ralf/files.html>
- Russinovich, Mark. "Inside the Boot Process, Part 1."  
<http://www.windowsitpro.com/Article/ArticleID/3952/3952.html>
- Windows Vista Security  
<http://blogs.msdn.com/windowsvistasecurity/>
- Microsoft. Boot Configuration Data Editor FAQ,  
<http://www.microsoft.com/technet/windowsvista/library/85cd5efe-c349-427c-b035-c2719d4af778.mspx>
- P. N. Biddle. "Next-Generation Secure Computing Base," PDC, Seattle, 2004,  
[http://download.microsoft.com/download/1/8/f/18f8cee2-0b64-41f2-893d-a6f2295b40c8/TW04008\\_WINHEC2004.ppt](http://download.microsoft.com/download/1/8/f/18f8cee2-0b64-41f2-893d-a6f2295b40c8/TW04008_WINHEC2004.ppt)
- M. Conover (2006, March). "Analysis of the Windows Vista Security Model,"  
[http://www.symantec.com/avcenter/reference/Windows\\_Vista\\_Security\\_Model\\_Analysis.pdf](http://www.symantec.com/avcenter/reference/Windows_Vista_Security_Model_Analysis.pdf)
- Microsoft. "First Look: New Security Features in Windows Vista," TechNet,  
<http://www.microsoft.com/technet/technetmag/issues/2006/05/FirstLook/default.aspx>
- Randall Hyde, Art of assembly Language
- Bugcheck and Skape, Kernel Mode Payloads on Windows  
<http://www.uninformed.org/?v=3&a=4&t=pdf>



# Questionnaire ??



Questions ??  
Comments ??  
Ideas ??  
email us

[nitin@nvlabs.in](mailto:nitin@nvlabs.in)  
[vipin@nvlabs.in](mailto:vipin@nvlabs.in)

<http://www.nvlabs.in>



Thanks

