Black Hat Europe 2007

Vboot Kit:
Compromising Windows Vista Security

Nitin Kumar , Security Researcher and Consultant

nitin.kumar@nvlabs.in

Vipin Kumar, Security Researcher and Consultant

vipin.kumar@nvlabs.in

http://www.nvlabs.in

Introduction

- Overview
  - Transfer of execution from BIOS to boot-sector
  - Vista Boot Process
  - Vbootkit (how it works)
  - Capabilities
- Demonstration Time
  - Privilege escalation shell code in action

Transfer of execution from BIOS to boot sector

- CD-ROM : 2KB sector loaded at 0000h:7C00h
- HDD: 512 bytes from MBR loaded at 0000h:7C00h .MBR finds a valid boot partition and loads partition boot sector
- PXE (Preboot Execution Environment): can download and load up to 500KB code at 0000h:7C00h

NOTE: After loading, all code is executed in real mode

Vista Boot Process

- MBR load NT BootSector (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
- BOOTMGR.EXE has 16 header prepended to itself.This 16 bit header checks the checksum of embedded PE EXE and maps it at 0x400000

NOTE:-First security check is simple checksum protection.

Vista Boot Process(continued)

- Execution of BOOTMGR starts in 32 bits in BmMain function.It verifiies itself 2 times using the functions ImgpValidateImageHash & BmFwVerifySelfIntegrity

- After this, it checks for hibernation state,if it’s found, it loads winresume.exe and gets done

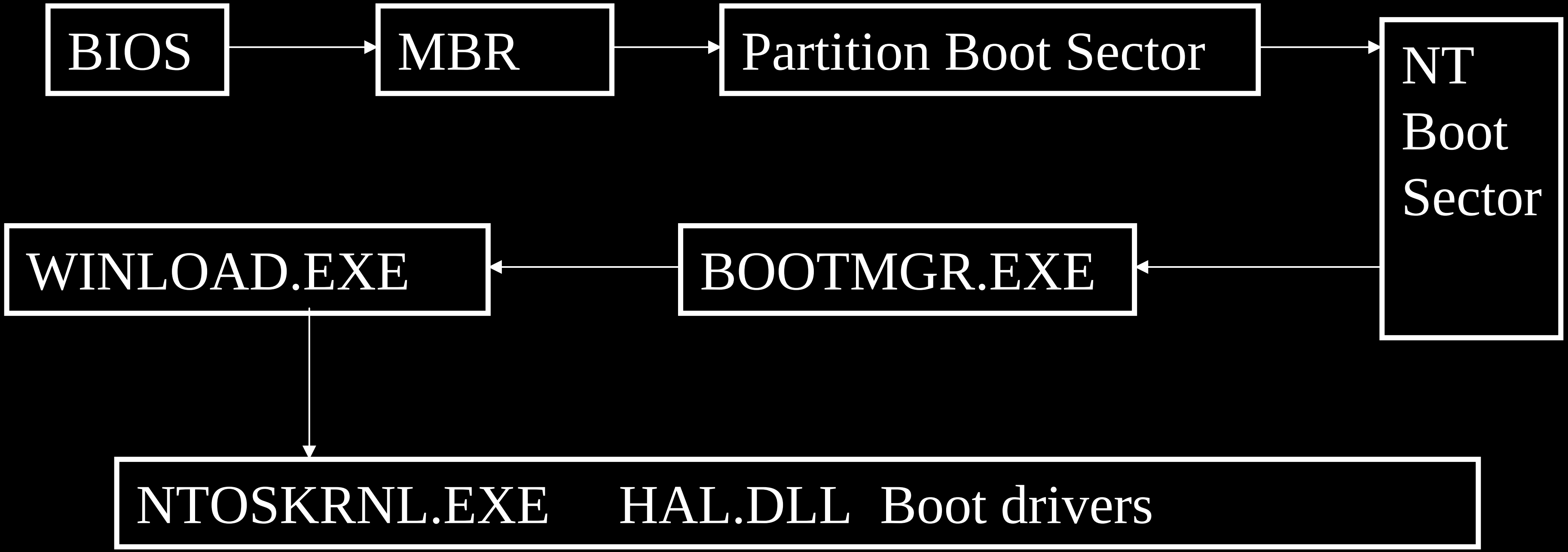
- It then mounts BCD database and enumerates boot entries,settings etc

NOTE:- 2 protections mentioned should be patched

Vista Boot Process(continued)

- After user selects a boot entry,It is launched using BmLaunchBootEntry with added switches
- 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



BIOS

MBR

Partition Boot Sector

NT Boot Sector

BOOTMGR.EXE

WINLOAD.EXE

NTOSKRNL.EXE HAL.DLL Boot drivers

Vista Kernel Start-up

- 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
    - Phase1InitializationDiscard
      - HalInitSystem
      - ObInitSystem
      - Sets boot time bias for ASLR
      - PsInitialSystemProcess
      - StartFirstUserProcess (starts SMSS.EXE)

Vboot Kit

Mission Status: Completed successfully

Vboot Kit- The Objective

- The objective is to get the Windows Vista running normally with some of the our changes done to the kernel.
- Also, the Vboot kit should pass through all the security features implemented in the kernel without being detected.
- No files should be patched on disk,it should run complete in memory to avoid later on detection.

Weak Points

- Windows Vista loader assumes that the system has not been compromised till it gains execution
- Windows Vista assumes that the memory image of an executable file is intact between the loading of file(system checks its validity just after loading a file) and execution of the file

These are the two main weaknesses Vbootkit exploits to get the job done.

Another Weak point

Every security protection implemented is of the following type

```
If (good) //security not compromised
{ // continue action
}
Else  //security has been compromised
{ //do something special
}
```

The above code when compiled by any compiler or assembler takes the following form

```
cmp, eax,1  //assume eax contains security status
Je good
               //control arrives here if security compromised
;do somethin special
Skip goog
Good:
```

Vboot Kit Features

- Proof of Concept code
  - Supports booting from CD-ROM and PXE
  - Fully demonstrates patching every protection implemented by Microsoft
  - Displays our signature at OS selection menu
  - Is just 1340 lines of code (nearly 1749 bytes after assembling)
  - Demonstrates a kernel mode shell code which peroidicaly escalates all cmd.exe to SYSTEM privileges
  - Supports pluggable shellcodes at compilation time

Vboot Kit overview

- Hook INT 13 (for disk reads)
- Keep on patching patching files as they load
  - Gain control after bootmgr has been loaded in memory
- The above would give us control so as we can patch the 16 bit header and the bootmgr itself.

Vboot kit – Functional workout

- Our code gains execution from the CD-Rom, 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 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 5 bytes from bootmgr.exe excluding zeroes

  for RC1 signature is 9d cd f5 d4 13 (in hex)

  for RC2 signature is 43 a0 48 a6 23 (in hex)
- The payload patches bootmgr.exe at 3 different places
  - Since the resources are read from MUI file,we implemented a detour style patch so as the MUI resources are patched
  - To gain control after winload has been loaded, but haven’t started executing
  - To disable FVE (full volume encryption)

Vboot kit – Functional workout(continued)

- Now, the 16 bit header starts execution and we face the first security check.It's a simple checksum protection stored the PE Header.
- The checksum algorithm is very simple

Do a add with carry on the buffer excluding the bytes where checksum is stored

Then,extract high 16 bits and low 16 bits and add them,neglecting any carry , then add the file size to the 16 bit value to get the final checksum

```
computenextword :
sub     edx,2  ;assume edx contains size to checksum
mov      cx,[esi]              ; load 2-byte block
add     eax,ecx                ; compute 2-byte checksum
adc     eax,0                  ;add carry
skip:   add     esi,2          ; update source address
cmp     edx,0                  ;buffer fully checksummed
jne   computenextword ;  more 2-bytes blocks
```

```
mov     edx,eax            ; copy checksum value
shr     edx,16             ; isolate high order bits
and     eax,0ffffh         ; isolate low order bits
add     eax,edx            ; sum high and low order bits
mov     edx,eax            ; isolate possible carry
shr     edx,16             ;
add     eax,edx            ; add carry
and     eax,0ffffh         ; clear possible carry bit
 add eax,filesize   //final checksum is now in eax
```

NOTE:- this protection is defeated by computing and fixing checksum after patching bootmgr

Vboot kit – Functional workout(continued)

- Now the bootmgr is mapped at 0x400000 and gains execution in 32-bit mode
- The first job bootmgr performs is to verify it’s own digital signature.This is done 2 times using 2 different functions **ImgpValidateImageHash** and **BmFwVerifySelfIntegrity**
- Both the patches are single byte patches , reversing the condition JE (jump if equal) to JNE (jump if not equal)
- Now after bootmgr loads its resources,detour takes control , relocates the vboot kit a second time, to protect itself to 0x45b000, patches the display message and passes control back to bootmgr
- Now bootmgr displays boot menu together with our signature
- After the user , selects an Entry to boot, the bootmgr calls **BlImgLoadPEImageEx** to load Winload.exe.It also verifies the digital signature of the file

Vboot kit – Functional workout(continued)

- After winload.exe has been mapped to memory and it's digital signature has been verified, our detour takes control and applies 2 detours
  - First detour to relocate ourselves (once again)
  - Second detour so as we can patch NTOSKRNL.exe and other drivers
- 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 al drivers loaded, their base adresses.It 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) and so on.

Vboot kit – Functional workout(continued)

- Structure of loader block Winload passes to NTOSKRNL

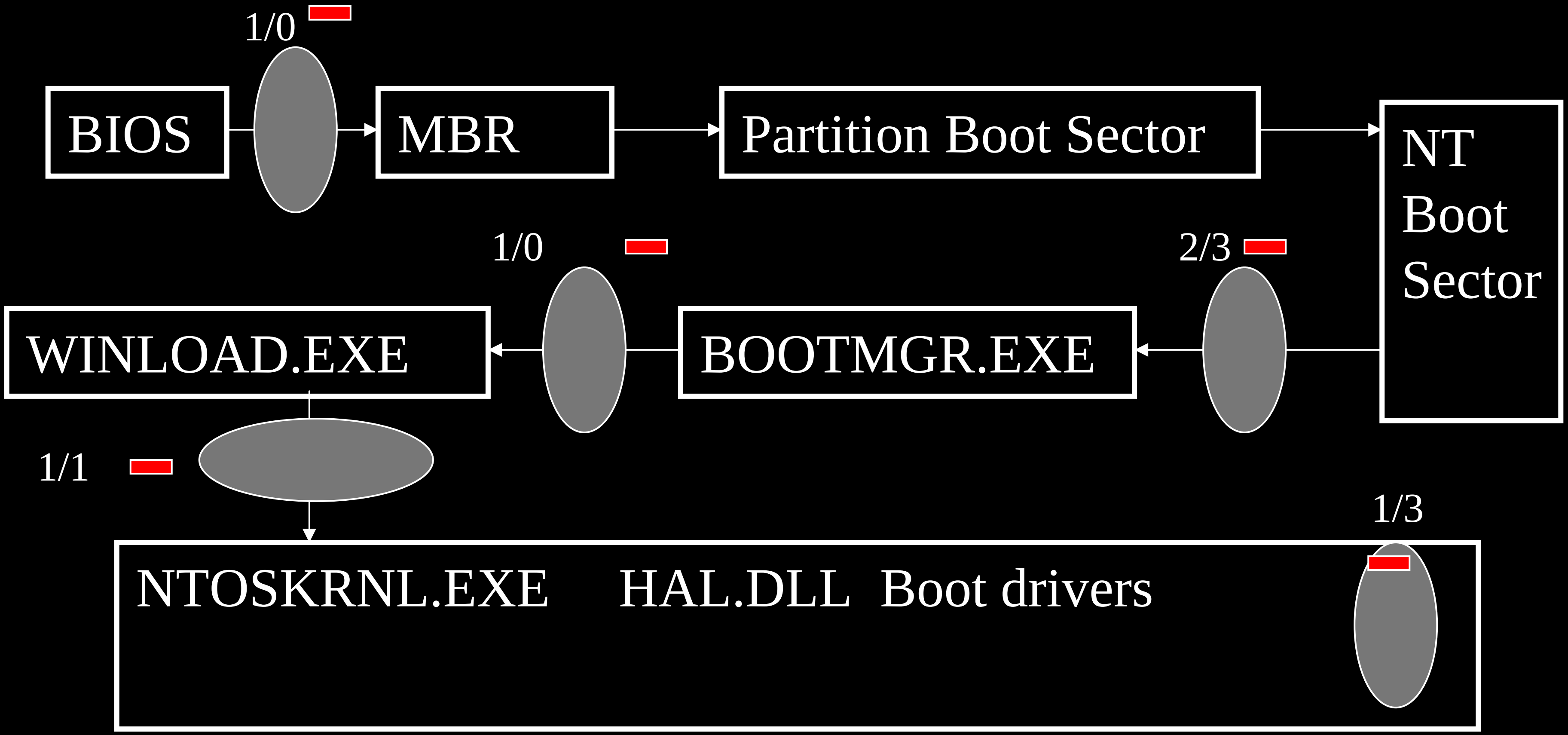
```
kd> dt _LOADER_PARAMETER_BLOCK 0x8081221c
   +0x000 LoadOrderListHead : _LIST_ENTRY [ 0x8082f7d4 - 0x8084f1f0 ]
   +0x008 MemoryDescriptorListHead : _LIST_ENTRY [ 0x80a1f000 - 0x80a20630 ]
   +0x010 BootDriverListHead : _LIST_ENTRY [ 0x80833c64 - 0x80832228 ]
   +0x018 KernelStack      : 0x81909000
   +0x034 ArcBootDeviceName : 0x80812e24  "multi(0)disk(0)rdisk(0)partition(1)"
   +0x03c NtBootPathName   : 0x80812ca8  "\Windows\"
   +0x044 LoadOptions      : 0x8080a410  "/BOOTDEBUG /NOEXECUTE=OPTOUT
      /NOPAE /DEBUG"
   +0x048 NlsData          : 0x8084e200 _NLS_DATA_BLOCK
   +0x054 SetupLoaderBlock : (null)
   +0x058 Extension        : 0x80812e5c _LOADER_PARAMETER_EXTENSION
   +0x068 FirmwareInformation : _FIRMWARE_INFORMATION_LOADER_BLOCK
```

Vboot kit – Functional workout(continued)

- 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's an 20 bytes patch,replacing stale code of Phase1init and jumping into it.

```
pushfd // save flags
Pushad /save registers
mov esi, NTOS_BASE_ADDRESS + NTOS_BLANK_SPACE
mov edi, NTOS_BASE_ADDRESS +
NTOS_INIT_PHASE_1_INIT_DISCARD
mov ecx, 2048 ; copy the whole vbootkit code
rep movsb
mov eax, NTOS_BASE_ADDRESS +
NTOS_PHASE_DISCARD_PATCH_STARTS
jmp eax
```

Summary of Vboot kit detours/patches



NOTE:- The ovals shows the point where Vboot kit hijacks control.The first number shows detours applied to next stage and second number shows patches applied.A red block shows relocation

Summary of Protections Found & Defeated

- Checksum protection (BOOTMGR) (100 byte fix-up)
- ImgpValidateImageHash (Digital Signature BOOTMGR) (1 byte jmp reverse)
- BmFwVerifySelfIntegrity (Digital Signature BOOTMGR) (1 byte jmp reverse)
- SelfIntegrityCheck(Digital Signature WINLOAD) removed in RC2 (1 byte jmp reverse)
- OslInitializeCodeIntegrity(WINLOAD) (1 byte 1 updated to zero)
- IntegrityChecks (WINLOAD) (1 byte 1 updated to zero)
- DEP protection (NTOSKRNL) (1 byte patch)

The Payload

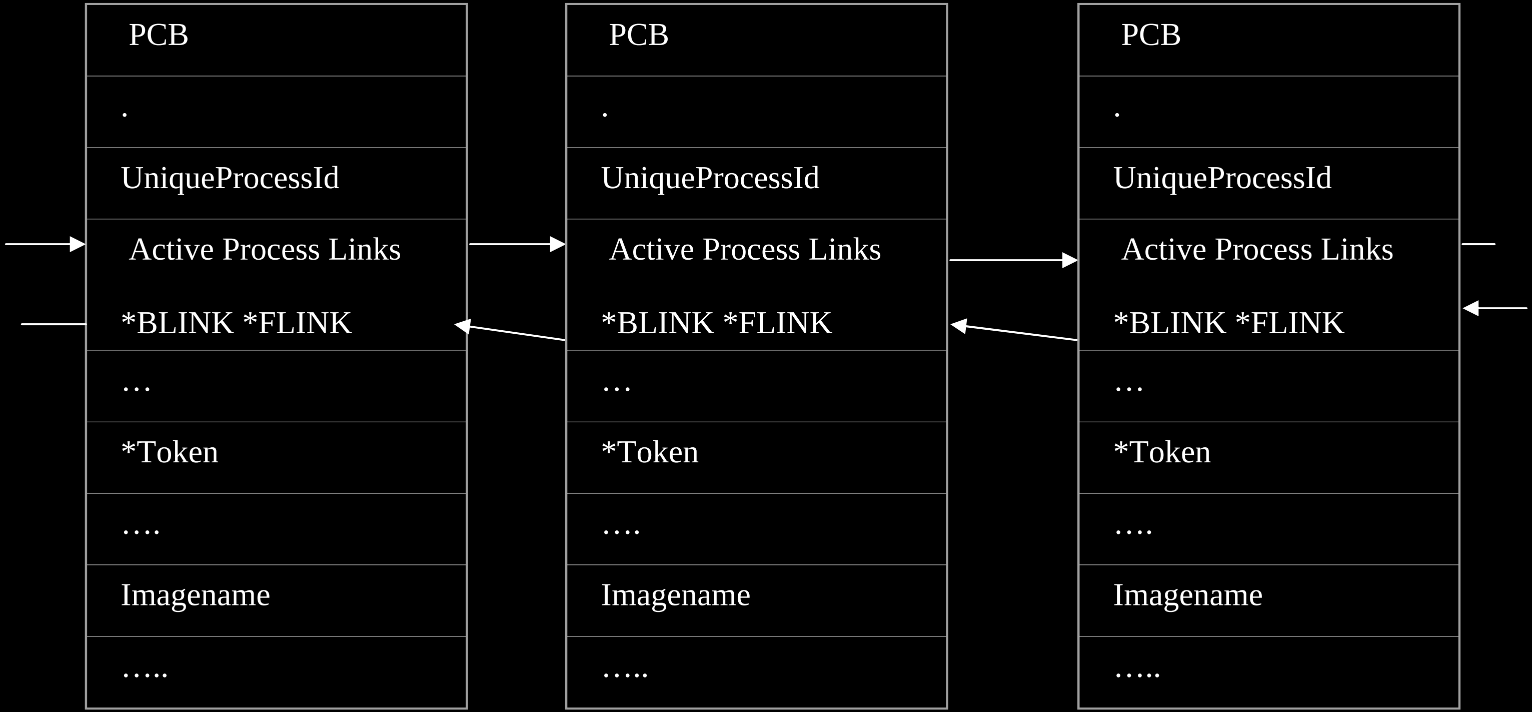
- Every exploit has a payload, so Vboot kit has it's own payload
- The payload for Vboot kit is a privilege escalation shellcode which increases privileges of less privileged process.The payloads runs in ring 0 (kernel land).
- The payload also writes the signature string to the kernel land , user land shared memory

Privilege Escalation Shell code (overview)

- Vboot Kit POC code periodically raises every CMD.EXE to privileges of SERVICES.EXE
- A thread is created which uses KeDelayExecution to sleep for say 30 seconds
- Since all threads started by Drivers are run in the context of System Process, our thread too gets the privileges.
- We traverse the _EPROCESS structure one by one to find services.exe, copy it’s security token and then replace security token of CMD.EXE

Looping through Processes

- Every process has an associated _EPROCESS structure which is linked to each other as shown below.This contains all security parameters of a process



PCB
.
UniqueProcessId
Active Process Links
*BLINK *FLINK
…
*Token
….
Imagename
…..

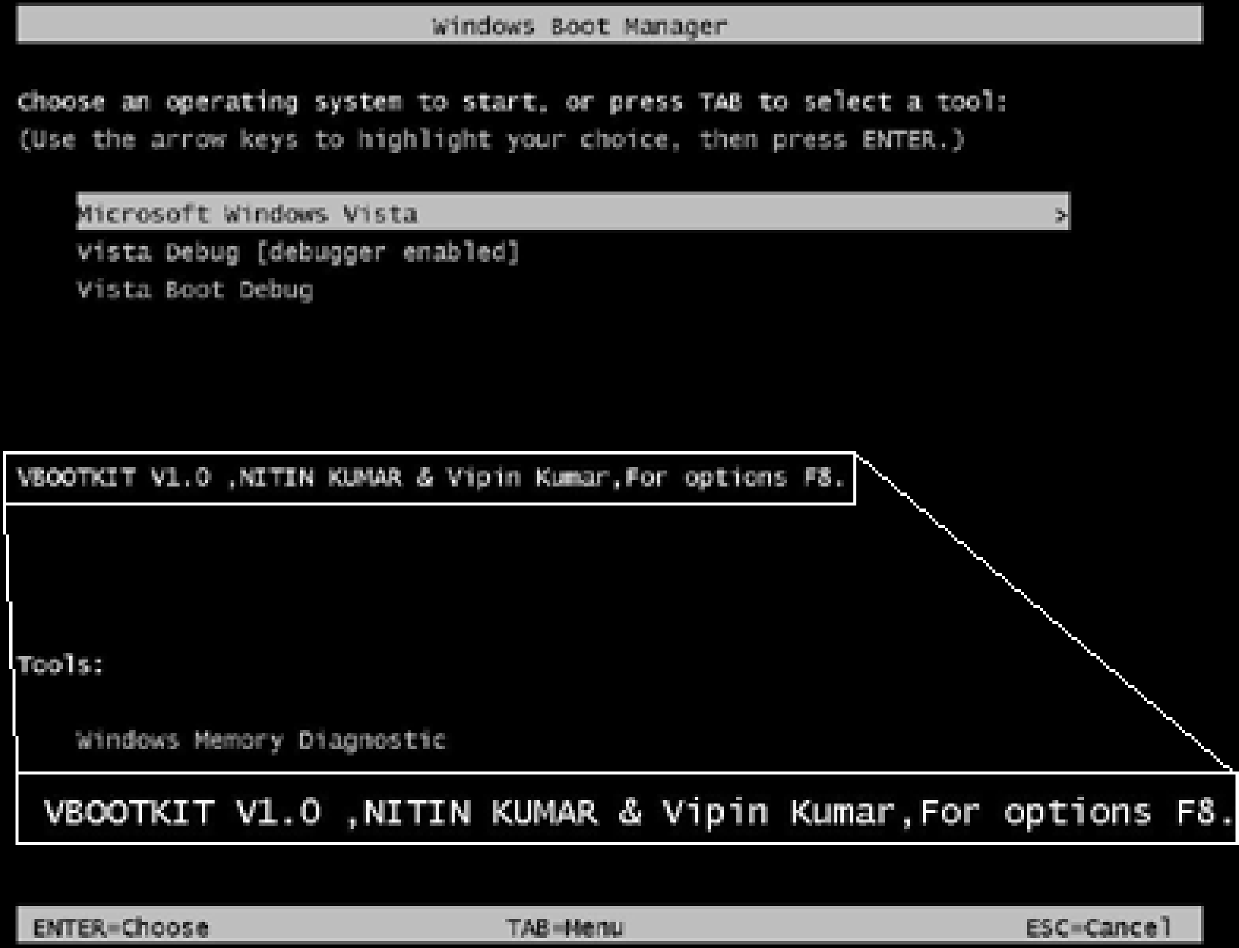
PCB
.
UniqueProcessId
Active Process Links
*BLINK *FLINK
…
*Token
….
Imagename
…..

PCB
.
UniqueProcessId
Active Process Links
*BLINK *FLINK
…
*Token
….
Imagename
…..

Time for the Demonstration

Demonstration Time(Signature)

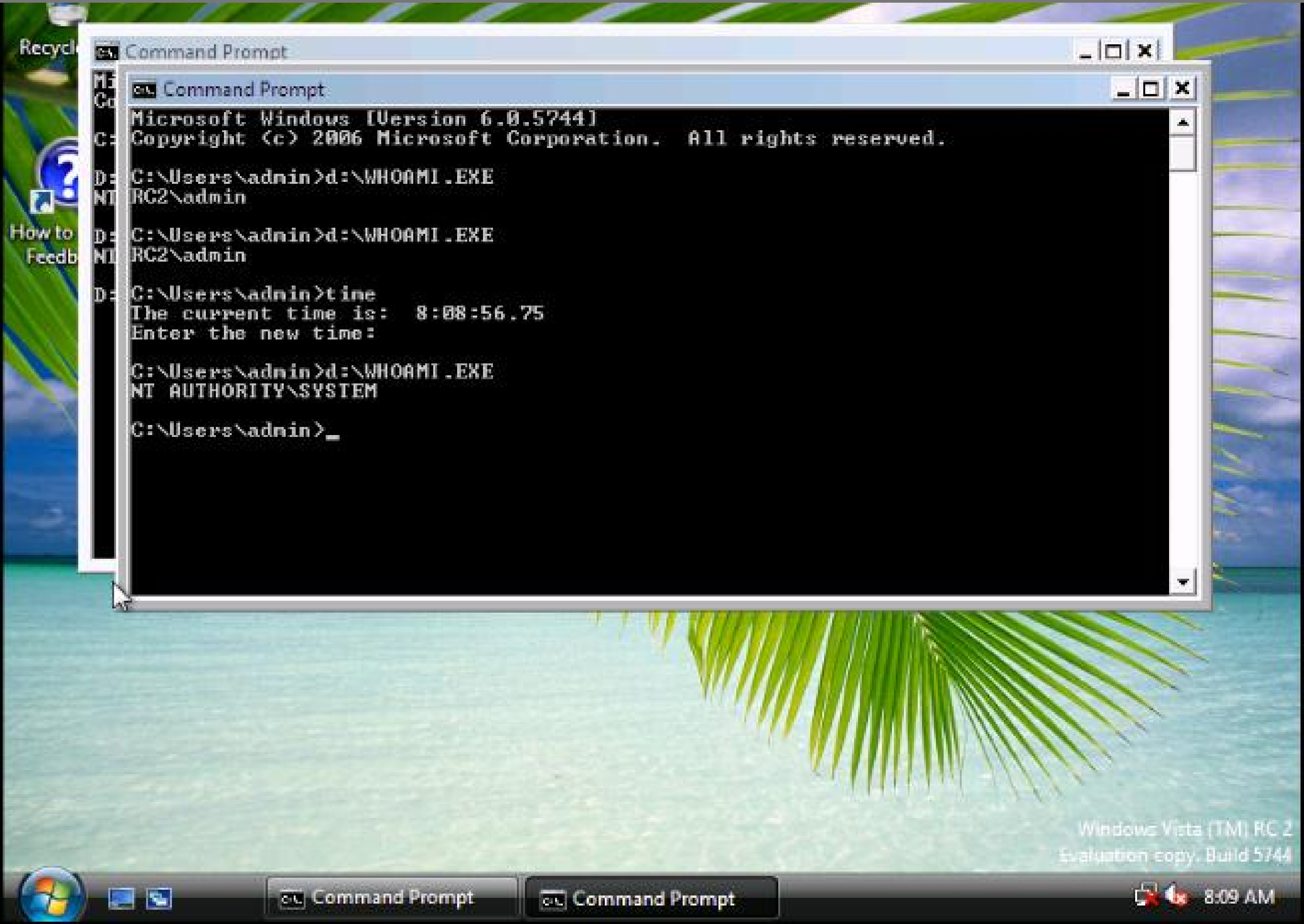
- From CD-ROM



```
                         Windows Boot Manager

Choose an operating system to start, or press TAB to select a tool:
(Use the arrow keys to highlight your choice, then press ENTER.)

    Microsoft Windows Vista                                        >
    Vista Debug [debugger enabled]
    Vista Boot Debug

VBOOTKIT V1.0 ,NITIN KUMAR & Vipin Kumar,For options F8.

Tools:

    Windows Memory Diagnostic

VBOOTKIT V1.0 ,NITIN KUMAR & Vipin Kumar,For options F8.

ENTER=Choose                 TAB=Menu                    ESC=Cancel
```

Screenshot showing signature

Demonstration Time (Shell code in action)



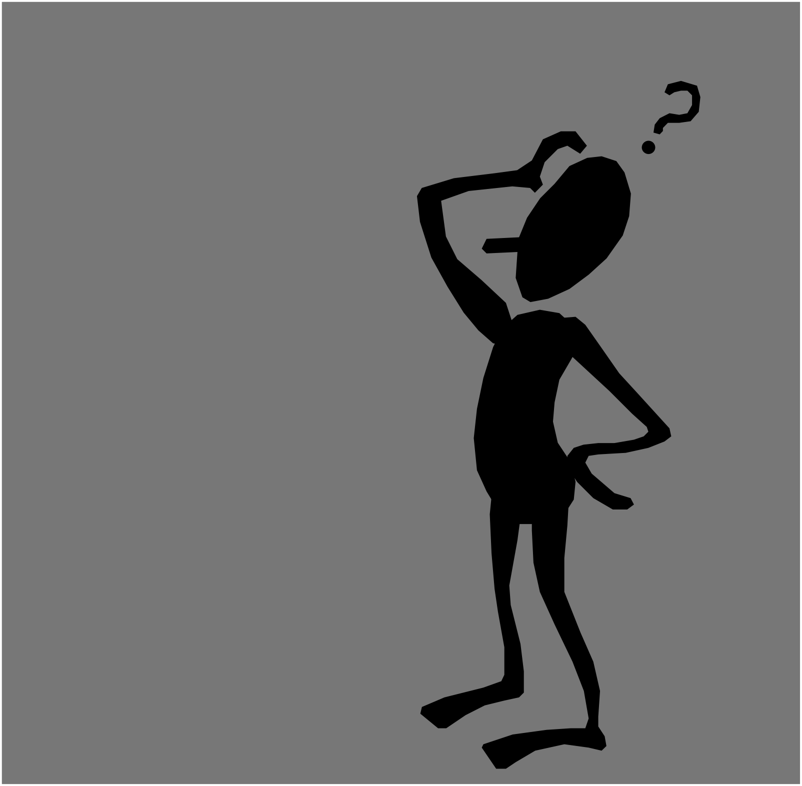
```
Microsoft Windows [Version 6.0.5744]
Copyright (c) 2006 Microsoft Corporation.  All rights reserved.

C:\Users\admin>d:\WHOAMI.EXE
RC2\admin

C:\Users\admin>d:\WHOAMI.EXE
RC2\admin

C:\Users\admin>time
The current time is:  8:08:56.75
Enter the new time:

C:\Users\admin>d:\WHOAMI.EXE
NT AUTHORITY\SYSTEM

C:\Users\admin>_
```

Time for the LIVE demonstration

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, Seatlle, 2004, 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
- 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

Questionare ?



Questions ?

Comments ?

E-mail us

nitin.kumar@nvlabs.in

vipin.kumar@nvlabs.in

http://www.nvlabs.in