# Persistent BIOS Infection

*"The early bird catches the worm"*

Anibal L. Sacco (Ssr Exploit writer)

Alfredo A. Ortega (Ssr Exploit writer)

CORE SECURITY TECHNOLOGIES

## Agenda

- Introduction
- A bit of history
- A better choice
- What is the BIOS
- BIOS Structure
- How it works
- Update/flashing process
- A Simple way to patch BIOS
- Where to patch
- What can be done
- Shellcodes
- Virtual machine demo
- Real hardware demo

## Introduction

- Practical approach to generic & reliable BIOS code injection
- True Persistency
- Rootkit(ish) behavior
- OS independant

## A little bit of history:

Commonly used persistency methods:

- User mode backdoor
- Kernel mode backdoor

**How can this be done more effectively?**

## BIOS Level backdoor:

- Takes control before any other software
- Stealth behavior
- Generally forgotten by almost all Antiviruses
- OS Independant (Runs outside the OS context)

## What is the BIOS?

- BIOS stands for Basic Input Output System
- Boot firmware
- Hardware initialization (RAM, North Bridge, etc.)
- Size: 256 Kb and bigger
- Commonly stored on EEPROM or flash memory

## BIOS Structure

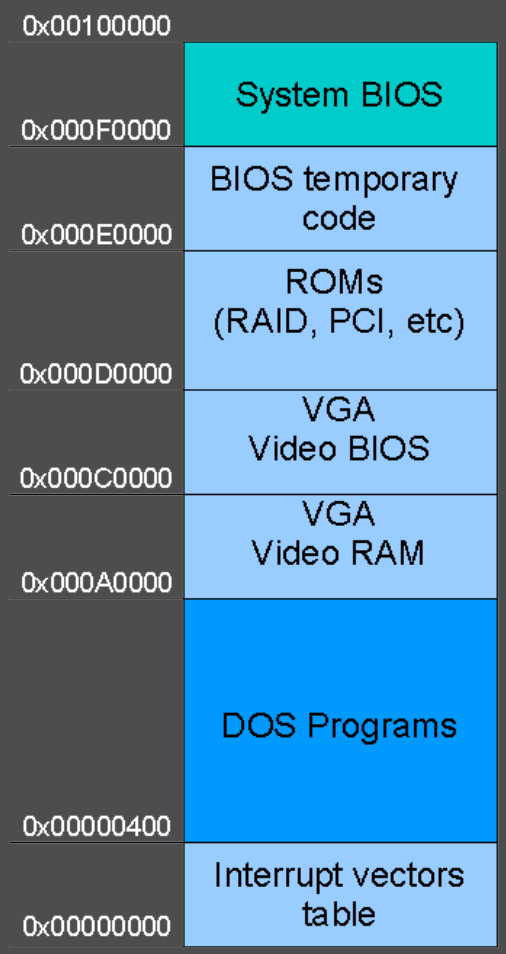
- It is composed of various LZH compressed modules
- Each module has an 8 bit checksum
- There are some uncompressed modules:
  Bootblock: In charge of the POST, and emergency boot
  Decompression routine: decompresses the rest of the modules
- Various checksum checks.

| Class.Instance (Name) | Packed ---> | Expanded | Compression | Offset |
|---|---|---|---|---|
| B.03 ( BIOSCODE) | 06DAF (28079) => | 093F0 ( 37872) | LZINT ( 74%) | 446DFh |
| B.02 ( BIOSCODE) | 05B87 (23431) => | 087A4 ( 34724) | LZINT ( 67%) | 4B4A9h |
| B.01 ( BIOSCODE) | 05A36 (23094) => | 080E0 ( 32992) | LZINT ( 69%) | 5104Bh |
| C.00 ( UPDATE) | 03010 (12304) => | 03010 ( 12304) | NONE (100%) | 5CFDFh |
| X.01 ( ROMEXEC) | 01110 (04368) => | 01110 ( 4368) | NONE (100%) | 6000Ah |
| T.00 ( TEMPLATE) | 02476 (09334) => | 055E0 ( 21984) | LZINT ( 42%) | 63D78h |
| S.00 ( STRINGS) | 020AC (08364) => | 047EA ( 18410) | LZINT ( 45%) | 66209h |
| E.00 ( SETUP) | 03AE6 (15078) => | 09058 ( 36952) | LZINT ( 40%) | 682D0h |
| M.00 ( MISER) | 03095 (12437) => | 046D0 ( 18128) | LZINT ( 68%) | 6BDD1h |
| L.01 ( LOGO) | 01A23 (06691) => | 246B2 (149170) | LZINT ( 4%) | 6EE81h |
| L.00 ( LOGO) | 00500 (01280) => | 03752 ( 14162) | LZINT ( 9%) | 708BFh |
| X.00 ( ROMEXEC) | 06A6C (27244) => | 06A6C ( 27244) | NONE (100%) | 70DDAh |
| B.00 ( BIOSCODE) | 001DD (00477) => | 0D740 ( 55104) | LZINT ( 0%) | 77862h |
| *.00 ( TCPA_*) | 00004 (00004) => | 00004 ( 004) | NONE (100%) | 77A5Ah |
| D.00 ( DISPLAY) | 00AF1 (02801) => | 00FE0 ( 4064) | LZINT ( 68%) | 77A79h |
| G.00 ( DECOMPCODE) | 006D6 (01750) => | 006D6 ( 1750) | NONE (100%) | 78585h |
| A.01 ( ACPI) | 0005B (00091) => | 00074 ( 116) | LZINT ( 78%) | 78C76h |
| A.00 ( ACPI) | 012FE (04862) => | 0437C ( 17276) | LZINT ( 28%) | 78CECh |
| B.00 ( BIOSCODE) | 00BD0 (03024) => | 00BD0 ( 3024) | NONE (100%) | 7D6AAh |

## How it works

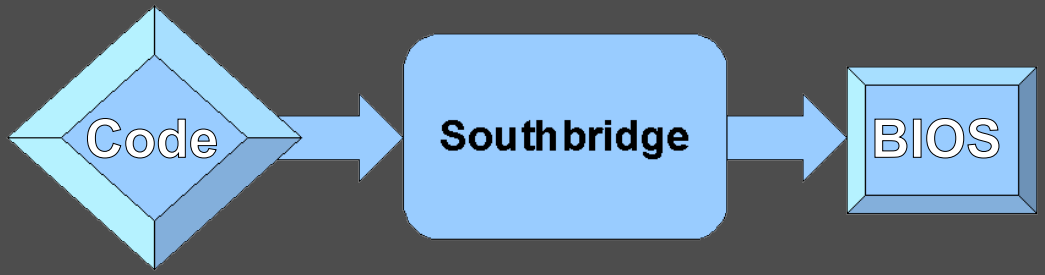
- The first instruction executed by the CPU is a 16 byte opcode located at F000:FFF0
- The Bootblock POST (Power On Self Test) initialization routine is executed.
- Decompression routine is called and every module is executed.
- Initializes PCI ROMs.
- Loads bootloader from hard-disk and executes it.

# BIOS Memory Map



## Update/flashing process

- BIOS is upgradeable.
- Vendors provide perodic updates to add new features and fix bugs. They also provides it's own tools to flash from DOS, windows, and even from ActiveX!
- BIOS update procedure depends on South-Bridge and chip used.
- CoreBOOT project provides a generic BIOS flashing tool: flashrom, that supports most motherboard/chip combination.



## A Simple way to patch BIOS

- BIOS contains several checksums
- Any modification leads to an unbootable system.
- We used two techniques:
  1) Use a BIOS building tool (Pinczakko's method)
  2) Patch and compensate the 8-bit checksum
- Three easy steps:
  1) Dump BIOS using flashrom
  2) Patch and compensate
  3) Re-flash

## Where to patch

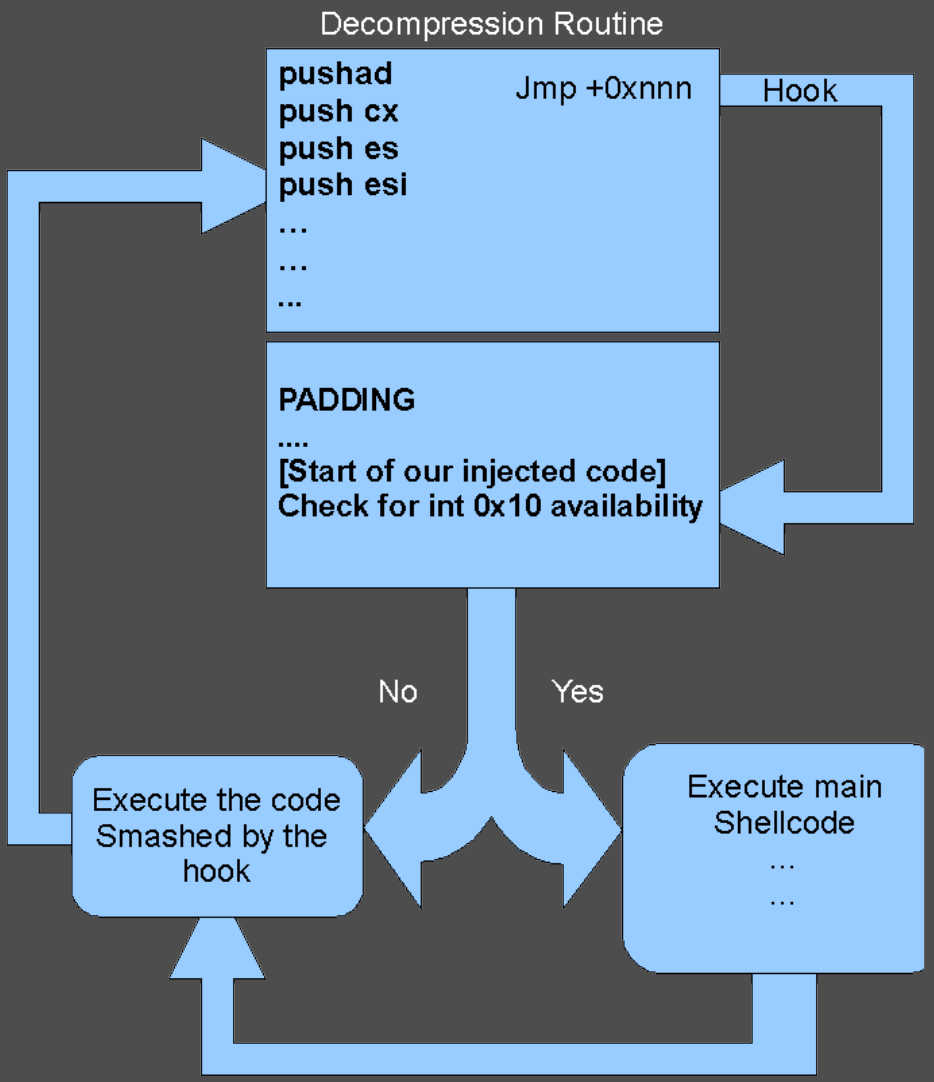
- Anywhere is valid:
  - f000:fff0: First instruction executed.
  - INT 0x19: Exected before booting
  - Insert a ROM module: Executing during POST
- The most practical place: Decompressor
  - It's uncompressed!
  - Located easily by pattern matching
  - Almost never change
  - Called multiple times during boot

## What can be done

- Depends. What resources are available from BIOS?
  - Standarized Hard Disk access (Int 13h)
  - Memory Manager (PMM)
  - network access (PXE, Julien Vanegue technique)
  - Modem and other hardware (Needs a driver)
- Our choice was to modify hard-disk content:
  - 1) Modify shadow file on unix
  - 2) Code injection on windows binaries

## Shellcodes

- Shellcodes are all in 16 bit
- We use BIOS services for everything
- Easy to debug: BIOS execution enviroment can be emulated running the code as a COM file over DOS
- Pseudocode:
  1) Checks ready-signal
  2) Checks for services inicialization
  3) Runs


Decompression Routine
pushad
push cx
push es
push esi
…
…
...
Jmp +0xnnn
Hook
PADDING
....
[Start of our injected code]
Check for int 0x10 availability
No
Yes
Execute the code Smashed by the hook
Execute main Shellcode
…
…

## Virtual machine demo

- Virtual machines also have a BIOS!
  In VMWARE, It's embedded as a section of the main VM process, shared on all Vms.
  Also can be specified on the VMX file for each VM.
  Is a phoenix BIOS.
  Very easy to develop because of the embedded GDB server.
  Using Interrupt Vector Table as ready-signal

- Two attacks:
  OpenBSD shadow file
  Windows code injection

- This method will infect multiple virtual machines.
  –

## Real hardware demo

- We infected an Phoenix-Award BIOS
- Extensively used BIOS
- Using the VGA ROM signature as ready-signal.
- No debug allowed here, all was done by Reverse-Engineering and later, Int 10h (Not even printf!)
- Injector tool is a 100-line python script!

## Future research

- Virtualized Rootkit
- PCI device placement (Modems, VGA, Ethernet and RAID controllers)
- The ultimate BIOS rootkit...

Thank you for your attention!