Vga Persistent Rootkit

by Nicolas A. Economou & Diego Juarez

CORE SECURITY

Intro

-VGA:

- **Definition**: Video graphic adapter (aka “Placa de video”)
- **Vendors**: NVIDIA, ATI, Intel, Matrox, etc
- **Type:** Integrated or discrete (AGP, PCI or PCI-E)
- **Flasheable Firmware:** The most

Intro

-Rootkit:

- **Definition**: Persistent and “invisible” malware
- **Used to**: Spy people, pivot from the target, etc
- **Starting**: OS initialized

-Bootkit:

- **Definition**: Rootkit capable of initializing in **BOOT TIME**
- **Used to**: To persist out of the **TARGET’S FILE SYSTEM**
- **Starting**: BOOT TIME (pre OS)

Intro

-VGA + Bootkit:

- **Persistence**: Independent from the OS, BIOS, Computer
- **Starting:** BIOS POST-time (pre BOOT TIME)
- **Used to:**
  - Take control of target from the pre-beginning
  - POST-time things (change BIOS settings, debugging, etc)
  - Fuck things up (Cagarle la vida a la gente)

Recent History

- "Persistent BIOS Infection"
  - **Authors:** Alfredo Ortega & Anibal Sacco
  - **Place:** CanSecWest 2009
  - **Description**: BIOS ROOTKIT proof of concept

- "Trojan.Bioskit.1"
  - **Authors:** Chinese ???
  - **Place:** In the Wild 2011
  - **Description**: Worm BIOS infector

- "Hardware Backdoring is Practical"
  - **Authors:** Jonathan Brossard
  - **Place:** Black Hat USA 2012
  - **Description**: "Rakshasa" - PCI ROOTKIT proof of concept

BIOS POST TIME

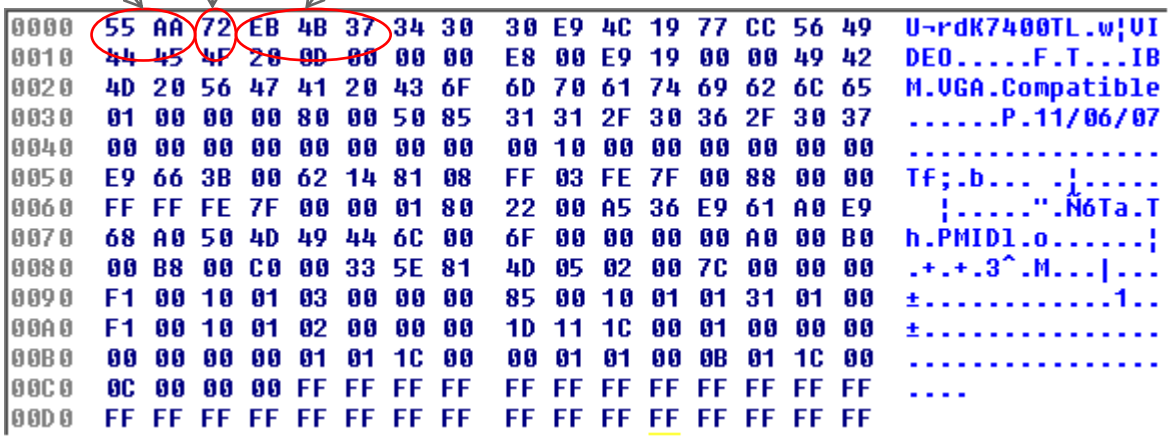
- POST (Power-on self-test):
  - 1. Verify CPU registers
  - 2. Check BIOS integrity
  - 3. Initialize BIOS
  - 4. Check RAM memory
  - 5. **Initialize devices**
  - 6. Enumerate devices (PCI, SCSI, HDDs)
  - 7. Load and execute the BOOT sector

PCI FORMAT

signature

image size

entry point code → “jmp $+N”



```
0000  55 AA 72 EB 4B 37 34 30  30 E9 4C 19 77 CC 56 49  U¬rdK7400TL.w¦VI
0010  44 45 4F 20 0D 00 00 00  E8 00 E9 19 00 00 49 42  DEO.....F.T...IB
0020  4D 20 56 47 41 20 43 6F  6D 70 61 74 69 62 6C 65  M.VGA.Compatible
0030  01 00 00 00 80 00 50 85  31 31 2F 30 36 2F 30 37  ......P.11/06/07
0040  00 00 00 00 00 00 00 00  00 10 00 00 00 00 00 00  ................
0050  E9 66 3B 00 62 14 81 08  FF 03 FE 7F 00 88 00 00  Tf;.b... .¦.....
0060  FF FF FE 7F 00 00 01 80  22 00 A5 36 E9 61 A0 E9    ¦.....".Ñ6Ta.T
0070  68 A0 50 4D 49 44 6C 00  6F 00 00 00 00 A0 00 B0  h.PMIDl.o......¦
0080  00 B8 00 C0 00 33 5E 81  4D 05 02 00 7C 00 00 00  .+.+.3^.M...|...
0090  F1 00 10 01 03 00 00 00  85 00 10 01 01 31 01 00  ±............1..
00A0  F1 00 10 01 02 00 00 00  1D 11 1C 00 01 00 00 00  ±................
00B0  00 00 00 00 01 01 1C 00  00 01 01 00 0B 01 1C 00  ................
00C0  0C 00 00 00 FF FF FF FF  FF FF FF FF FF FF FF FF  ....
00D0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
```

VGA Firmware

-EEPROM:

- SIZE: 32kb ~ 128kb

-Types:

- LEGACY VGA BIOS (PCI format)
- GOP

-Service:

- **INT 10h**, INT 1Fh, INT 42h, INT 43h, INT 6Dh

-VGA RAM:

- **SIZE**: 128kb (A000:0000 – BFFF:FFFF)

VGA boot modes

-LEGACY VGA BIOS:

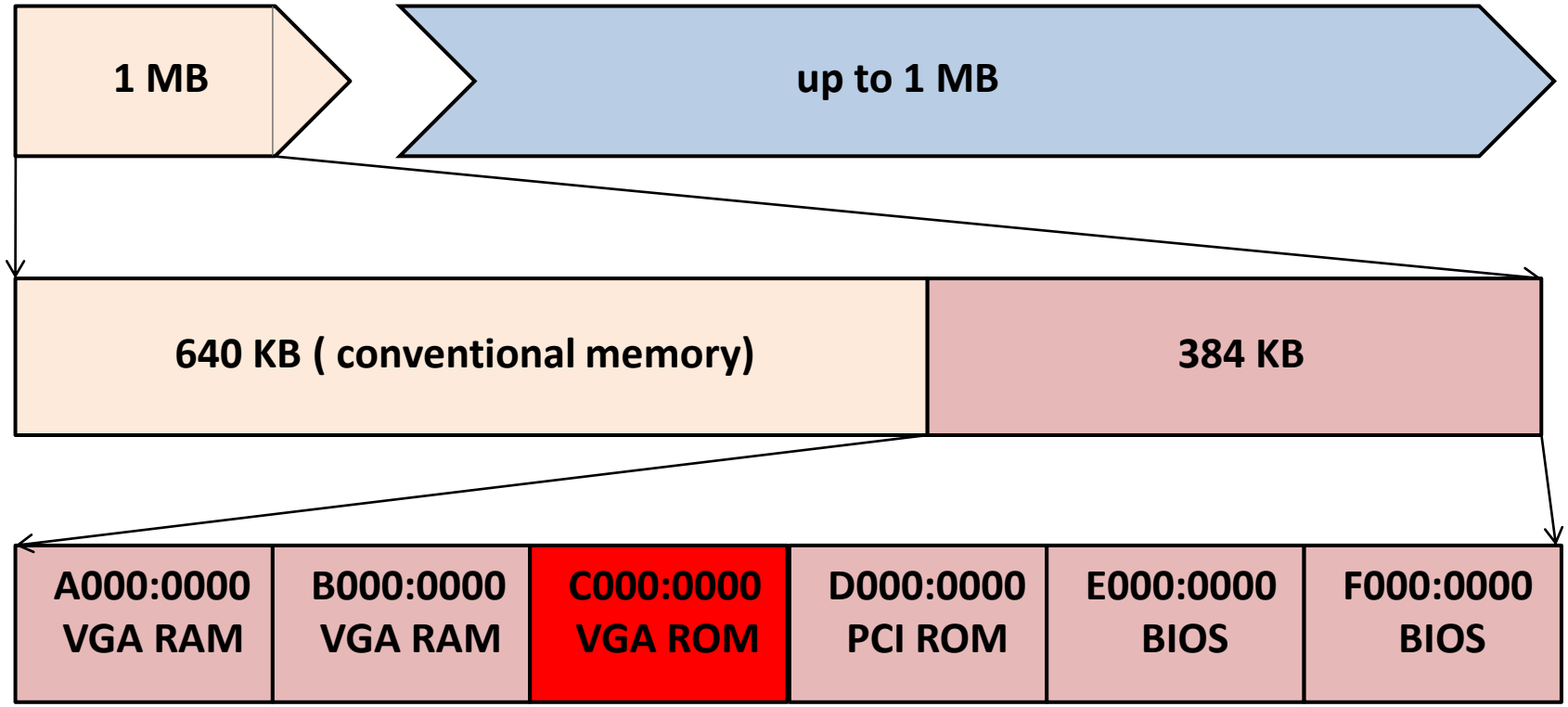
- **TYPE: LEGACY BOOT or UEFI CLASS 2**
- **ROM CODE**: 16 bits UNCOMPRESSED CODE – REAL MODE
- **ROM CODE EXECUTION**: POST TIME, BOOT TIME and in some cases in OS TIME (VMM86)

-GOP:

- **TYPE:** UEFI CLASS 3
- **ROM CODE**: COMPRESSED AND SIGNED
- **ROM CODE EXECUTION**: POST TIME, BOOT

VGA firmware mapping

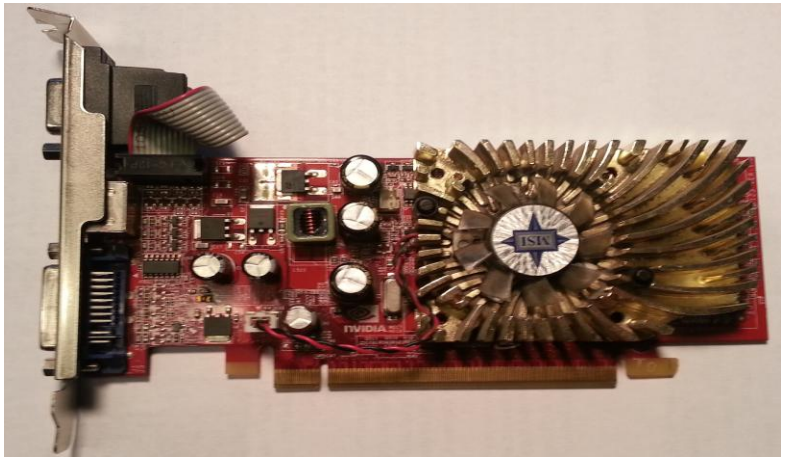
- ROM AREA: C000:0000 ~ C000:FFFF



CORE SECURITY

Research time …

- Placa:
  - MODEL: MSI NVIDIA 8400 GS (G84)
  - EEPROM: 64 kb
  - SLOT: PCI-E



Rootkiting the VGA

- Idea:
  - Use a VGA ROM flasher
  - Put the “rootkit” at the end of the ROM free space
  - Patch and point the PCI entry point code to the “rootkit”

VGA EEPROM flashers

-NVIDIA:

- **NAME:** **“nvflash.exe”**
- **OSs:** Windows, Mac OSX, DOS
- **LINK**: http://www.techpowerup.com/downloads/Utilities/BIOS_Flashing/NVIDIA/

-ATI:

- **NAME:** **“atiflash.exe”**
- **OSs:** Windows, DOS
- **LINK**: http://www.techpowerup.com/downloads/Utilities/BIOS_Flashing/ATI/

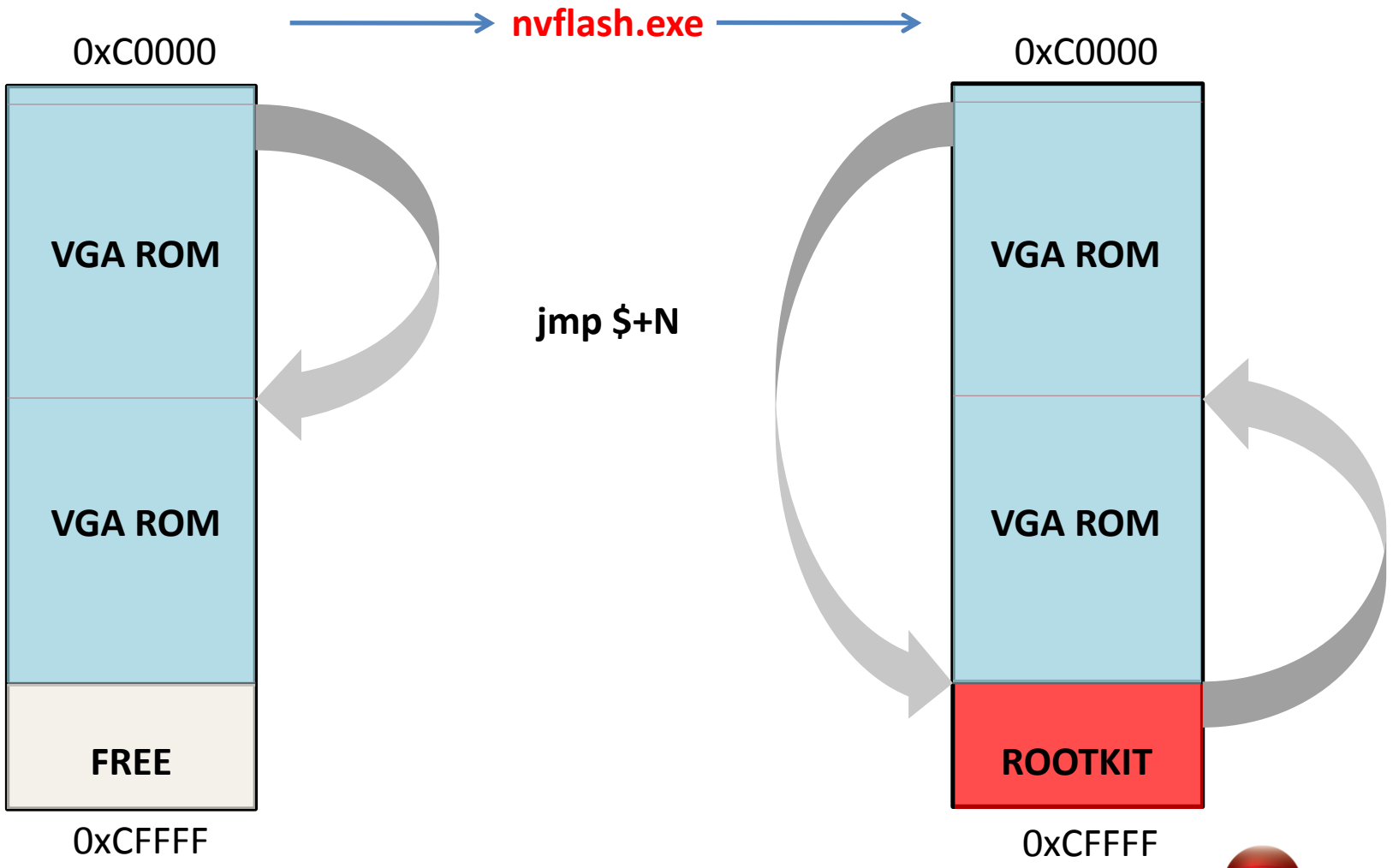
NVIDIA 8400 GS ROM image

- MAX SIZE: 65.536 bytes

- USED SPACE: 58.368 bytes

- FREE SPACE: 65.536 - 58.368 = 7.168 bytes → **ROOTKIT LEN** = 7.168b

Patching the ROM image



Research obstacles

-FIRST STAGE ROOTKIT CODE:
- 16 bits – REAL MODE
- PIC

-NO VIDEO OUTPUT:
- The SCREEN SIGNAL starts in the middle of the VGA initialization

-NO DEBUGGING:
- **UNIVERSAL DEBUGGING** is ”**IMPRACTICAL**” during VGA INITIALIZATION TIME
- NO Keyboard, NO Serial port, NO Screen, NO nothing

-CAN’T USE FIRMWARE HARDCODED ADDRESSES:
- The IDEA is a **GENERIC SHELLCODE**

NVIDIA Research obstacles

-ROM Checksum:
- **The LAST BYTE of the ROM**

-INT 10h enabler:
- **We need to detect in which moment the FIRMARE sets this entry**

-Persistence:
- **The FIRMWARE DOESN'T USE all the 64kb MEMORY SPACE**
- **The END of the ROM is WIPED !**

-PCI IMAGE SIZE RUNTIME MODIFICATION:
- **The BIOS will delete the FREE SPACE ((0x80 – NEW_IMG_SIZE) x 512 bytes)**

-RUN TIME integrity check:
- **Error beeps are shoot if something is WRONG**

ATI Research obstacles

-ROM Checksum:
- **2 bytes – UNUSED (?)**

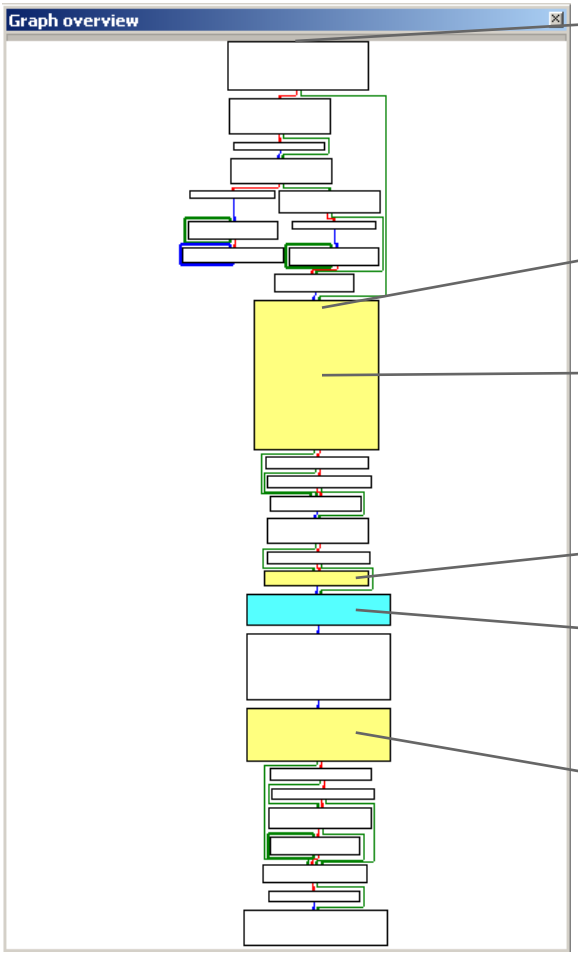
-INT 10h enabler:
- **We need to detect in which moment the FIRMWARE sets this entry**

-Persistence:
- **The FIRMWARE COMPLETELY USES the 64kb (EVEN if it doesn't have CODE)**
- **It means that the ROOTKIT is PROTECTED !**

-PCI IMAGE SIZE RUNTIME MODIFICATION:
- **NO MODIFIED**

-RUNTIME integrity check:
- **We DON'T DEAL WITH IT**

NVIDIA firmware roadmap



VGA initializer (called from JMP $+N)

VGA RAM mapper (RANGE A000 – BFFF)

INT 10H enabler

print (“VGA BIOS …”)

INTEGRITY CHECKER

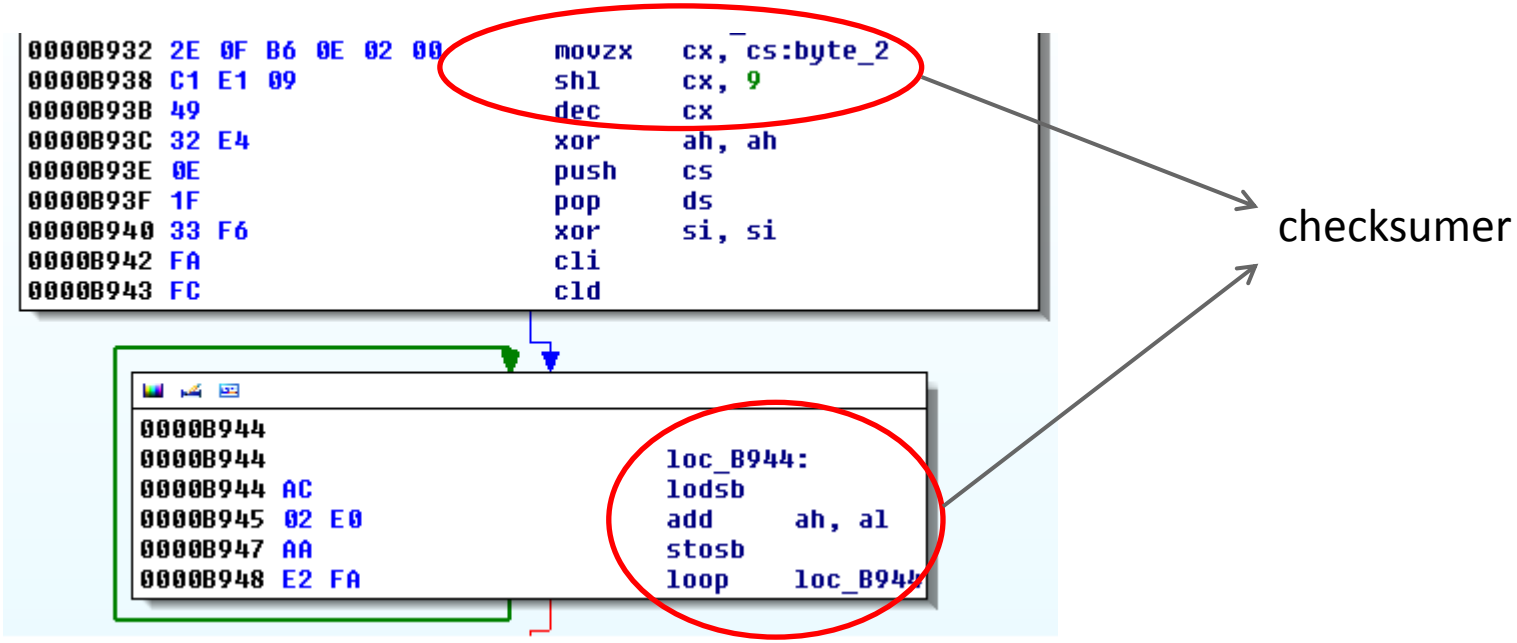
CODE Cleaner

NVIDIA - Bypassing Problems

- 1. ROM Checksum:
  - Take the patched firmware and add all the bytes discarding the HIGH PART
  - Patch the last byte with new checksum

NVIDIA - Bypassing Problems

-2. RUNTIME integrity check:



CORE SECURITY

NVIDIA - Bypassing Problems

- 2. RUNTIME Integrity check bypass:
  - The **current solution** doesn't modify the firmware in RUNTIME
  - The previous solution patched some CODE parts in RUNTIME

NVIDIA - Bypassing Problems

- 3. Detect **INT 10h** initialization:
  - The idea is to detect when the **entry 10h** of the **IVT** is set
  - When this entry is set, the "rootkit" will **CHANGE** this entry

INT 10h enabling detection

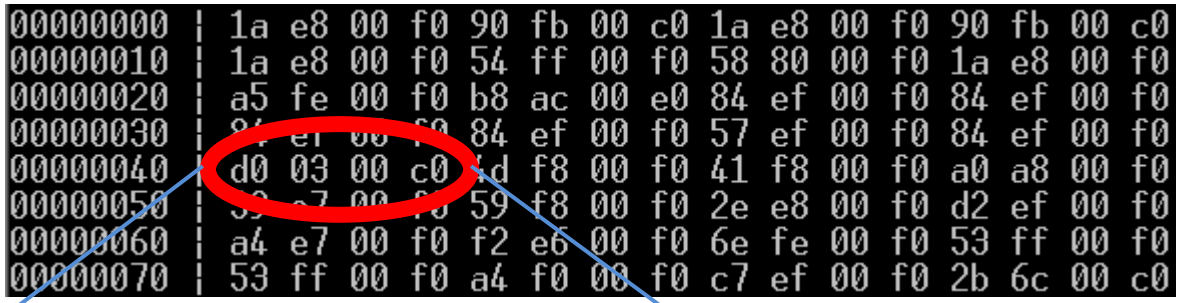
- Use a HARDWARE BREAKPOINT
  - Pointing to the address 0000:0040 (INT 10h entry)

- HARDWARE BREAKPOINT
  - X86 Debug Register used to detect:
    - Memory READs, memory WRITEs, memory EXECUTIONs and I/O reads or writes

IVT Definition

-Interrupt Vector Table

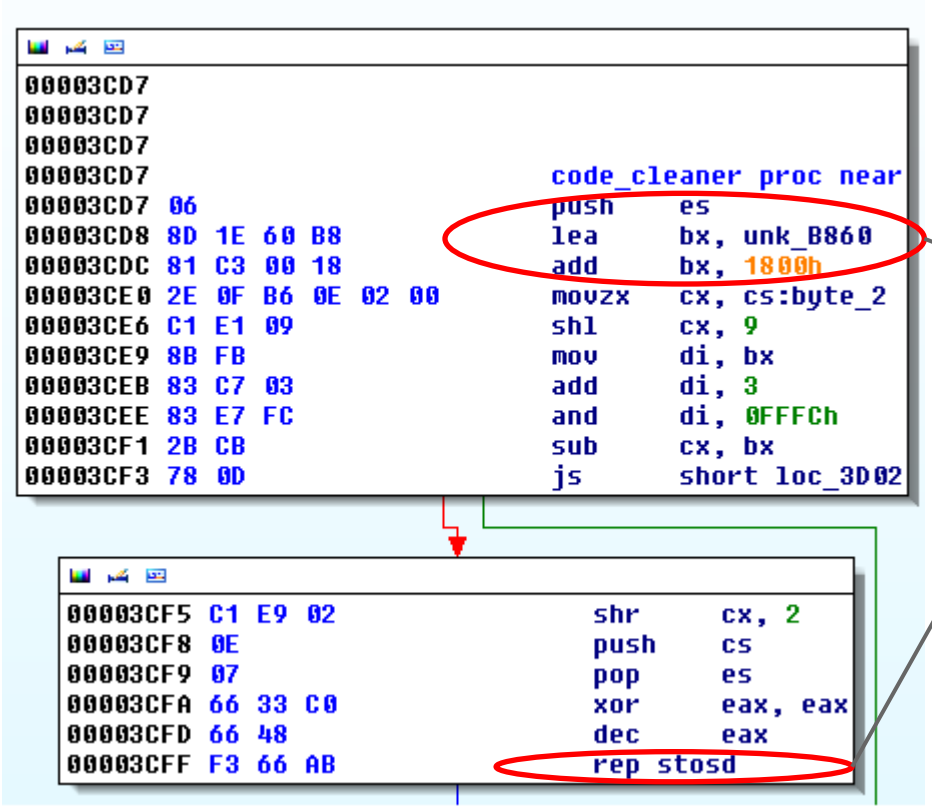
- Pointer table to interrupt rutines
- It's used in 16 bits - REAL MODE
- 4 byte – ENTRIES (offs:segm)



INT 10h vector → 03D0:C000 → C000:03D0
CUSTODIED by a **HARDWARE BREAKPOINT**

NVIDIA - Bypassing Problems

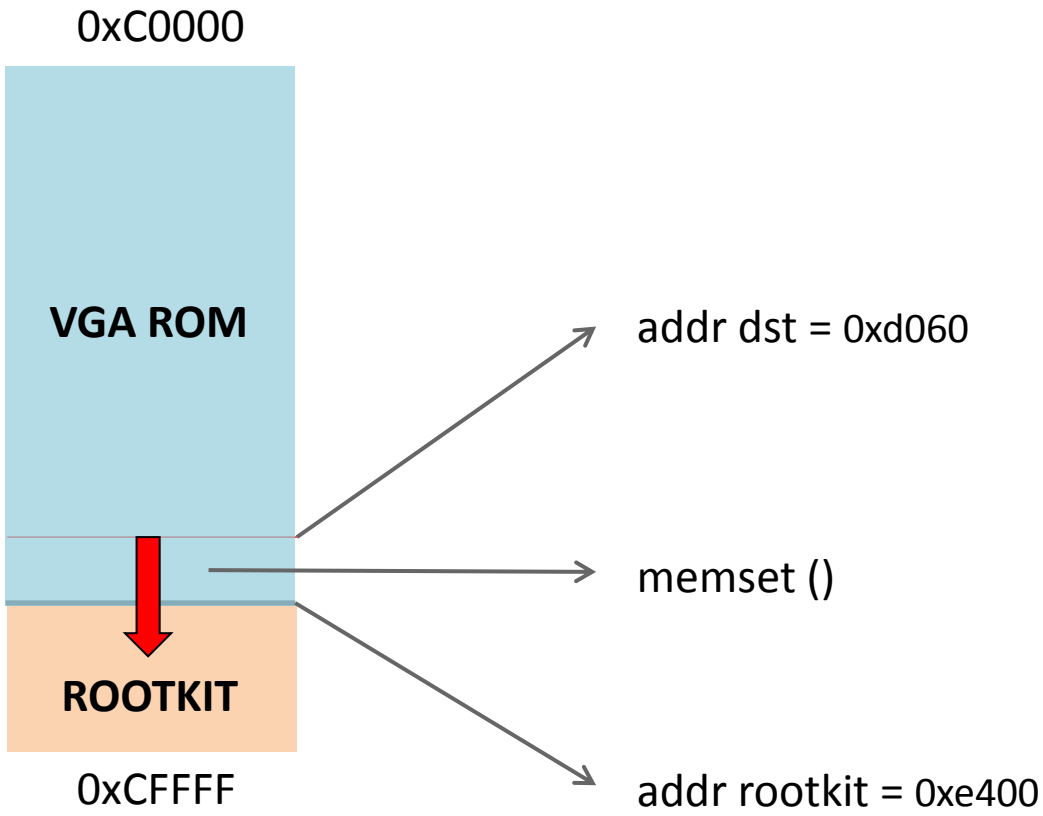
-4. The “code cleaner”



memset (0xd060, 0xff, imgsize-0xd060)

NVIDIA - Bypassing Problems

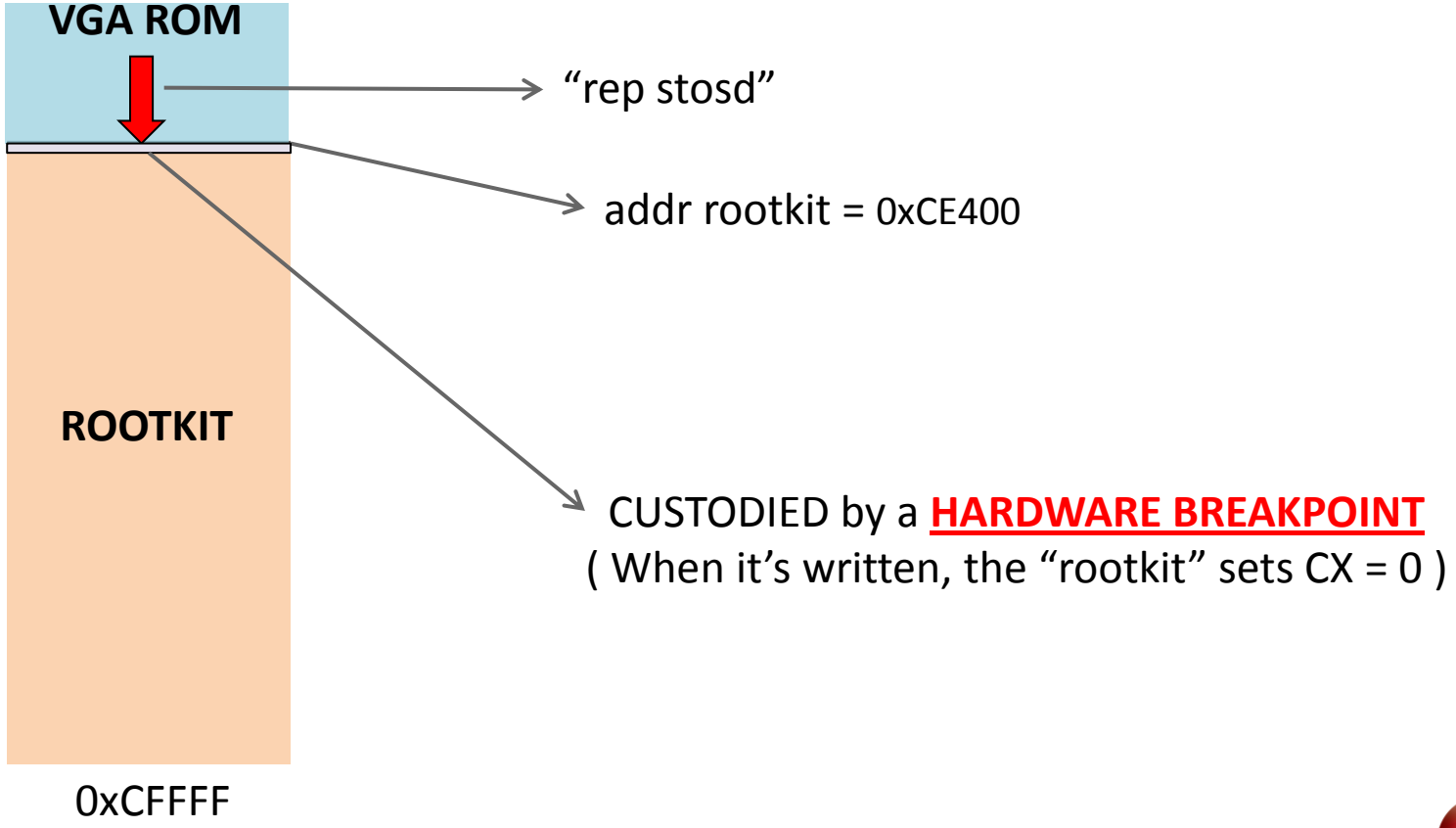
-4. The “code cleaner”



CORE SECURITY

NVIDIA - Bypassing Problems

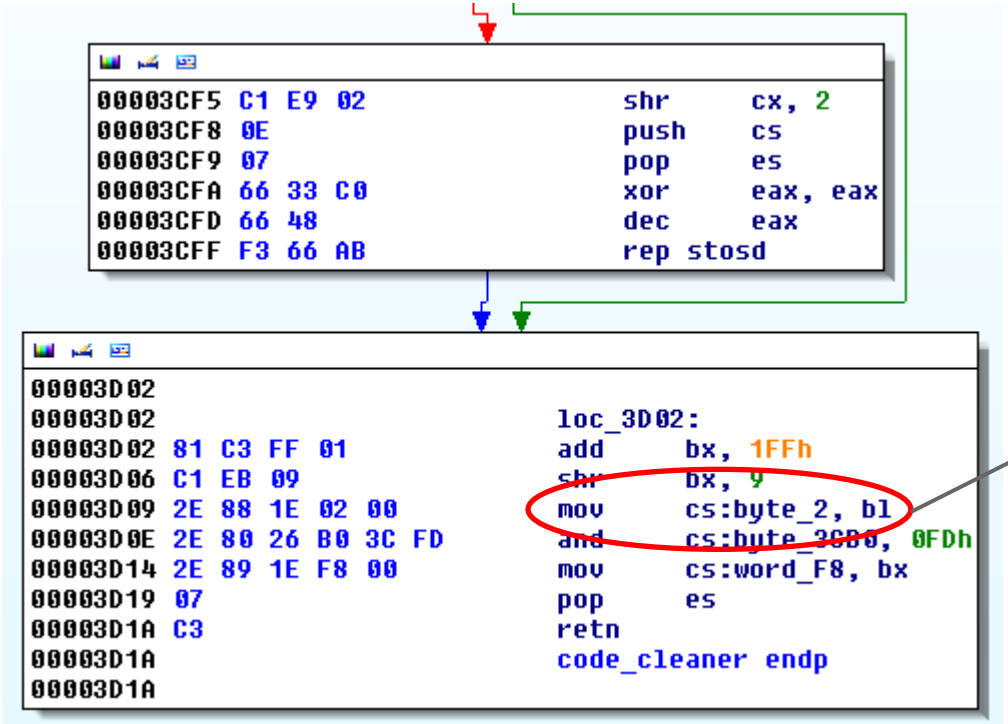
-4. The “code cleaner” bypass



CORE SECURITY

NVIDIA - Bypassing Problems

-5. Image size modification



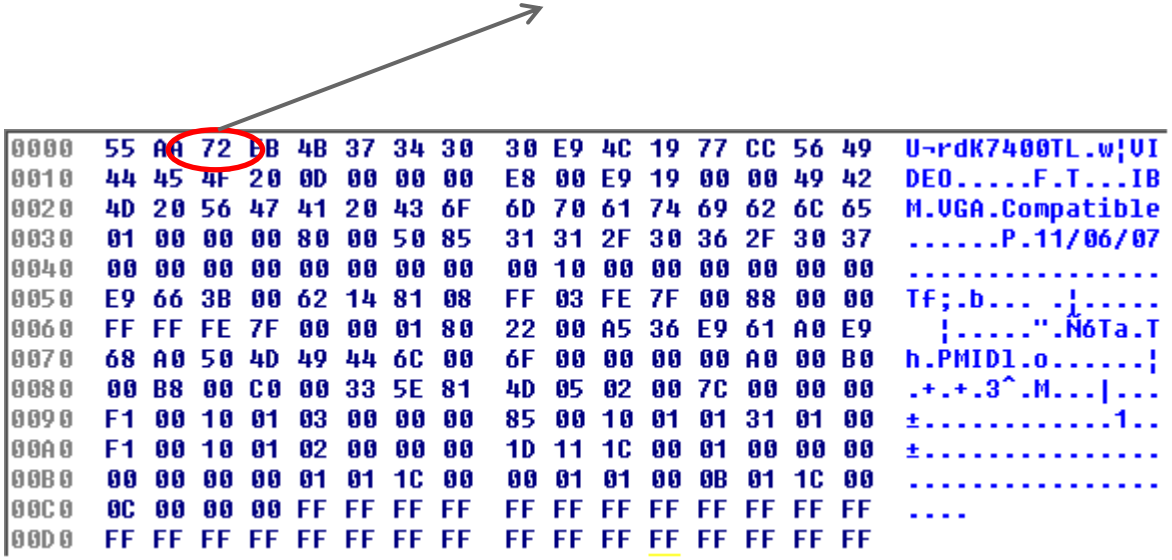
set the new image size

CORE SECURITY

NVIDIA - Bypassing Problems

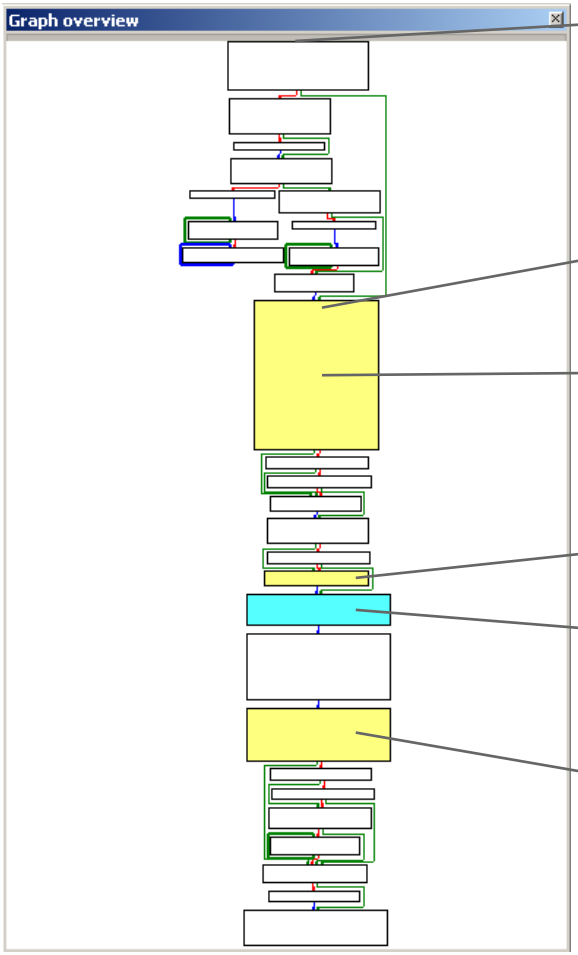
-5. Image size modification bypass

CUSTODIED by a **HARDWARE BREAKPOINT**
(When it's written, this "rootkit" puts 0x80)



CORE SECURITY

The VGA initialization was done …



VGA initializer (called from JMP $+N)

VGA RAM mapper (RANGE A000 – BFFF)

INT 10H enabler

print (“VGA BIOS …”)

INTEGRITY CHECKER

CODE Cleaner

WE ARE ALIVE HERE

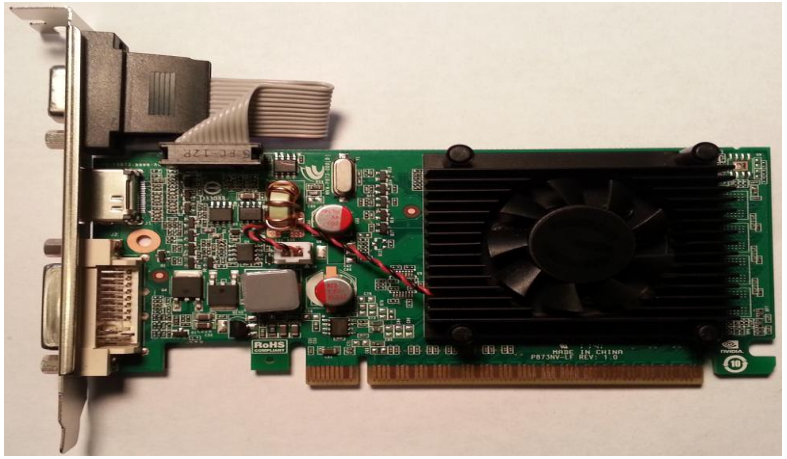
The VGA initialization was done …

- And now ?
  - The BIOS POST will continue initializing the rest of the hardware
  - The rootkit is HOOKING the INT 10h service, so, the control is not **LOST**
  - The next step is to **re-take** control during **BOOT-TIME** …

Demo Time

DEMO 1

- TARGET:
  - Vendor: NVIDIA Geforce
  - Model: 8400GS (GT218)

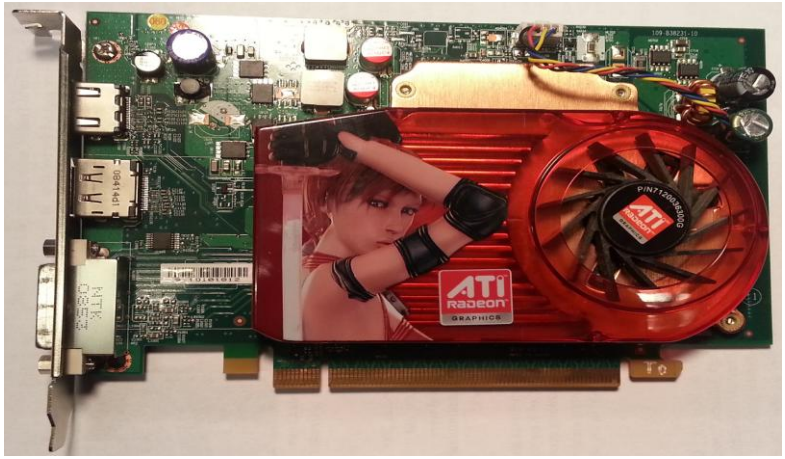


- SHELLCODE SIZE:
  - ~ 1600 bytes

- Objective:
  - Show code execution during POST
  - Show code execution during Windows 7 Boot

DEMO 2

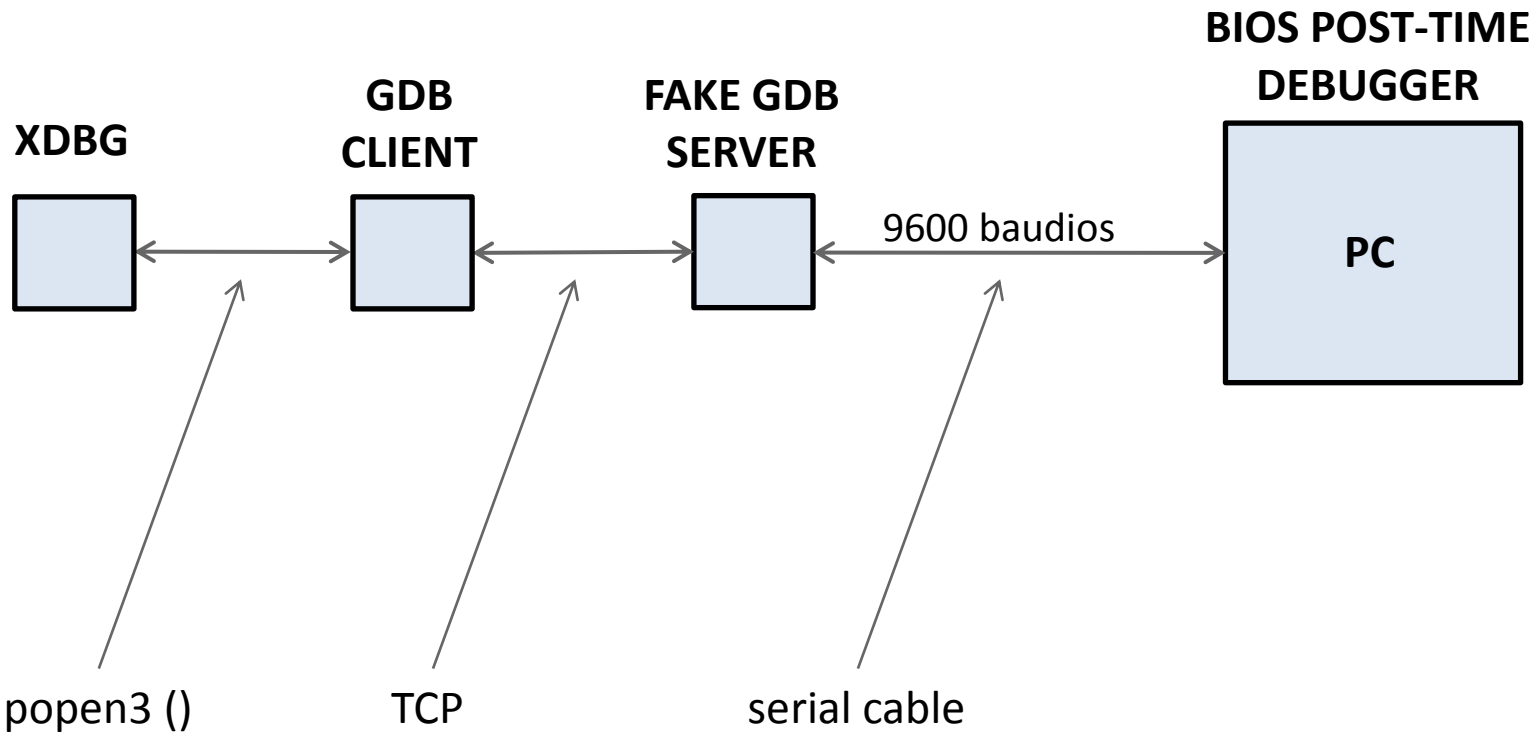
- TARGET:
  - Vendor: ATI Radeon
  - Model: RV635



- SHELLCODE SIZE:
  - ~ 1500 bytes

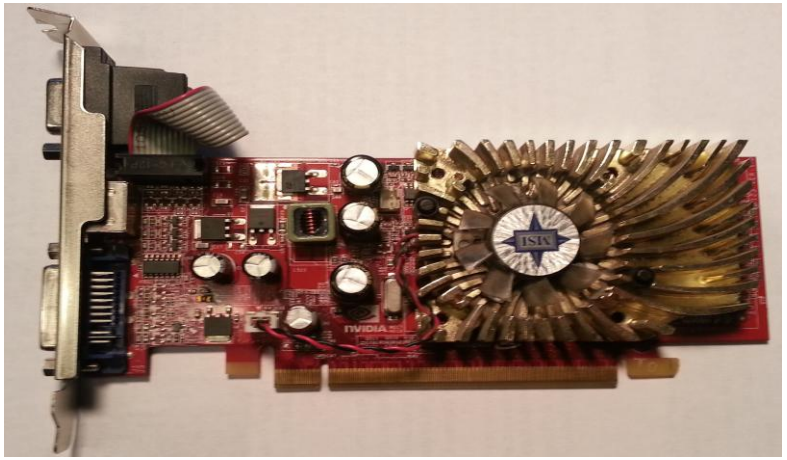
- Objective:
  - Execute a BIOS POST-TIME Debugger …

DEMO 2: Connection schema



DEMO 3

- TARGET:
  - Vendor: NVIDIA Geforce
  - Model: 8400GS (G84)



- SHELLCODE SIZE:
  - ~ 7300 bytes

- Objective:
  - Install **remotely** a VGA ROOTKIT …

DEMO 3: Attack scenario part 1

- Target:
  - Windows 7 SP0
- Vector:
  - Remote
- Vulnerable to:
  - **MS10-061** (Microsoft Windows Print Spooler Impersonation Vulnerability)

DEMO 3: Attack scenario part 2

-Attack's Objective:

- Installs a Bootkit (Deep Boot) in the **VGA FIRMWARE**
- http://corelabs.coresecurity.com/index.php?module=Wiki&action=view&type=publication&name=Deep_Boot

-Bootkit's Objective:

- Installs a **ROOTKIT** in the memory of Windows 7

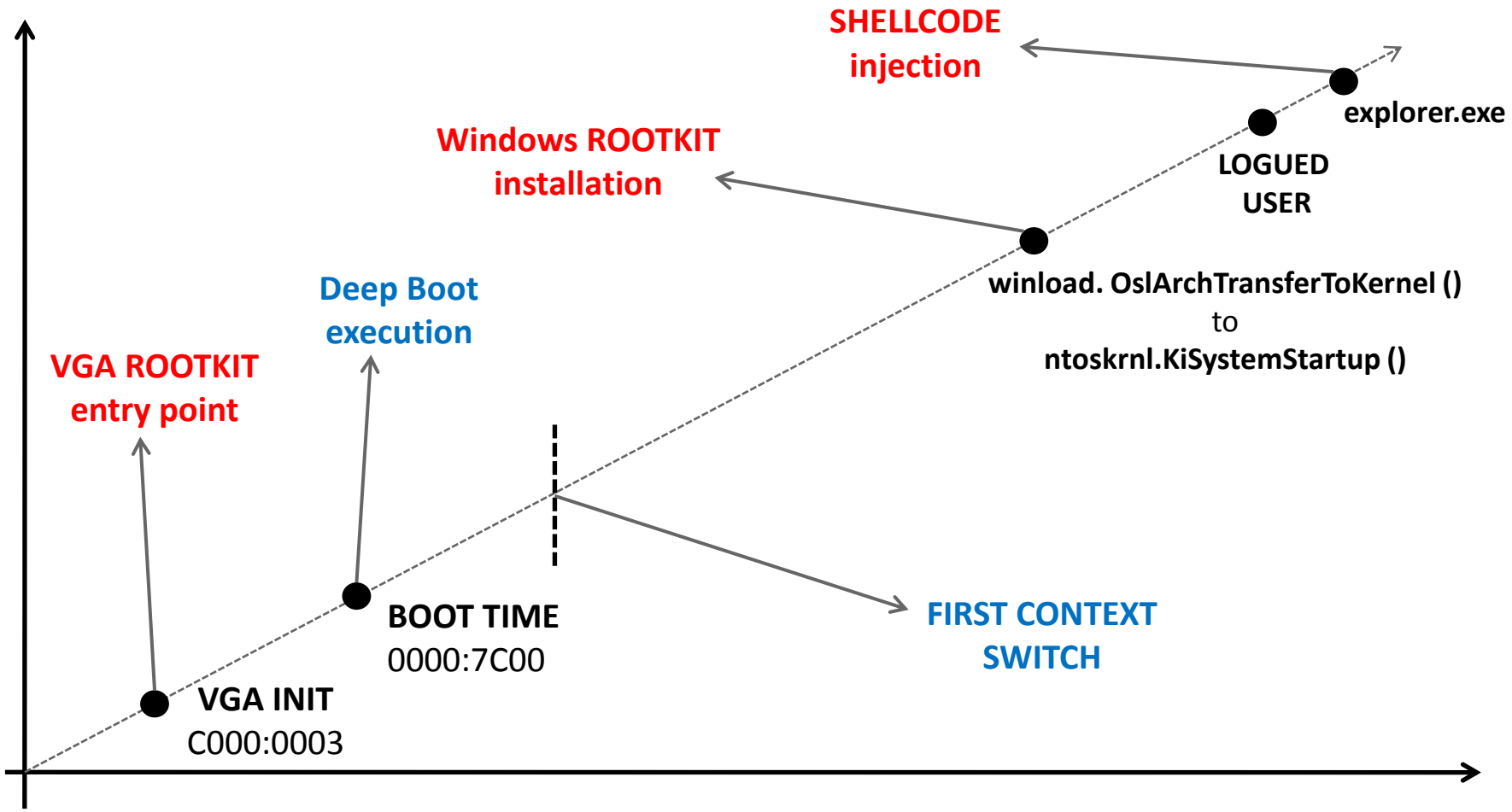
-Rootkit's Objective:

- Injects shellcode in **explorer.exe**

-Shellcode's Objective:

- Connect to **internet**

DEMO 3: VGA Rootkit Time Line



CORE SECURITY

Protections ?

- Use:
  - UEFI 3.0 + TPM
- Don't use:
  - VGA discrete boards (**even Dual Boards**)

Questions ?