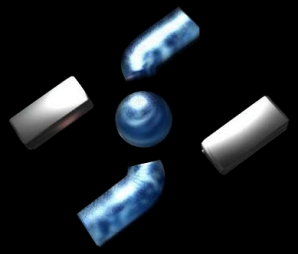


Hispasec

Bootkit vs Windows

… czyli tu, tam, i z powrotem

Mateusz ‘j00ru’ Jurczyk

SecDay, Wrocław 2009

Plan prezentacji

- Bootkity - wprowadzenie
- Windows XP SP3 – ładowanie systemu
- DEMO – bootkit w akcji
- Budujemy kod bootkitu
- Windows Vista SP2 / 7 RC – co nowego?
- Pytania

Kilka słów o mnie…

Mateusz ‘j00ru’ Jurczyk

- Reverse Engineer @ Hispasec
- Pentester / Bughunter
- Vexillium (http://vexillium.org)
- Autor bloga technicznego (http://j00ru.vexillium.org)

Bootkity - wprowadzenie

- Z czym mamy do czynienia?
  - Kod wykonywalny umieszczony na nośniku bootowalnym dowolnego typu (HDD, Floppy, CD/DVD, Flash memory)
    - Nie ogranicza się do nośników lokalnych – PXE (*Preboot Execution Environment*)
  - Pierwsza kontrolowana jednostka egzekucji
    - Pomijając możliwość modyfikacji BIOS
  - Niezależność od konkretnej platformy systemowej
  - Idealne wsparcie dla rootkita (lub jego integralna część)

Bootkity – wprowadzenie cd.

- Praktyczne możliwości
  - Kontrola zachowania systemu operacyjnego
    - Ominięcie zabezpieczeń/ograniczeń systemowych (DRM, PatchGuard itp.)
    - Ważne kwestie kompatybilności
  - Zdalna infekcja maszyn (PXE)
  - Ominięcie software'owych mechanizmów szyfrowania dysku (TrueCrypt)
  - W skrócie – możemy dokładnie tyle, ile sam system operacyjny, a nawet więcej

Bootkity – historia

Nazwa	Rok powstania	Autor	System operacyjny
Wirus Stoned	1987r.	?	DOS
BootRoot	2005r.	Derek Soeder, Ryan Permeh	Windows XP
PiXie	2006r.	Derek Soeder	Windows XP
BOOT-KIT	2007r.	Nitin Kumar, Vipin Kumar	Windows 2000, XP, 2003
Wirus Sinowal (aka Mebroot)	2007r.	?	Windows 2000, XP, 2003, Vista

Bootkity – historia

Nazwa	Rok powstania	Autor	System operacyjny
Vbootkit	2007r.	Nitin Kumar, Vipin Kumar	Windows Vista
TPMKit	2007r.	Nitin Kumar, Vipin Kumar	-
KON-BOOT	2008r. (?)	Piotr Bania	**Windows:** wszystkie od XP **Linux:** Gentoo, Ubuntu, Debian, Fedora
Vbootkit 2.0	2009r.	Nitin Kumar, Vipin Kumar	Windows 7 RC x64
Stoned Bootkit	2009r.	Peter Kleissner	Wszystkie od XP

Bootkity – wprowadzenie cd.

- Jak możemy się bronić?
  - Zabezpieczenia software nie wystarczą – sytuacja „kto pierwszy, ten lepszy”
    - Pomimo to, Windows Vista utrudnia nam życie sumami kontrolnymi
  - Zapobieganie a wykrywanie infekcji – uprzywilejowany dostęp do dysku twardego
  - Wprowadzenie zabezpieczeń sprzętowych – Trusted Platform Module jednym ze skutecznych rozwiązań

Windows XP SP3 – ładowanie

Windows XP SP3 – ładowanie

Ważne informacje

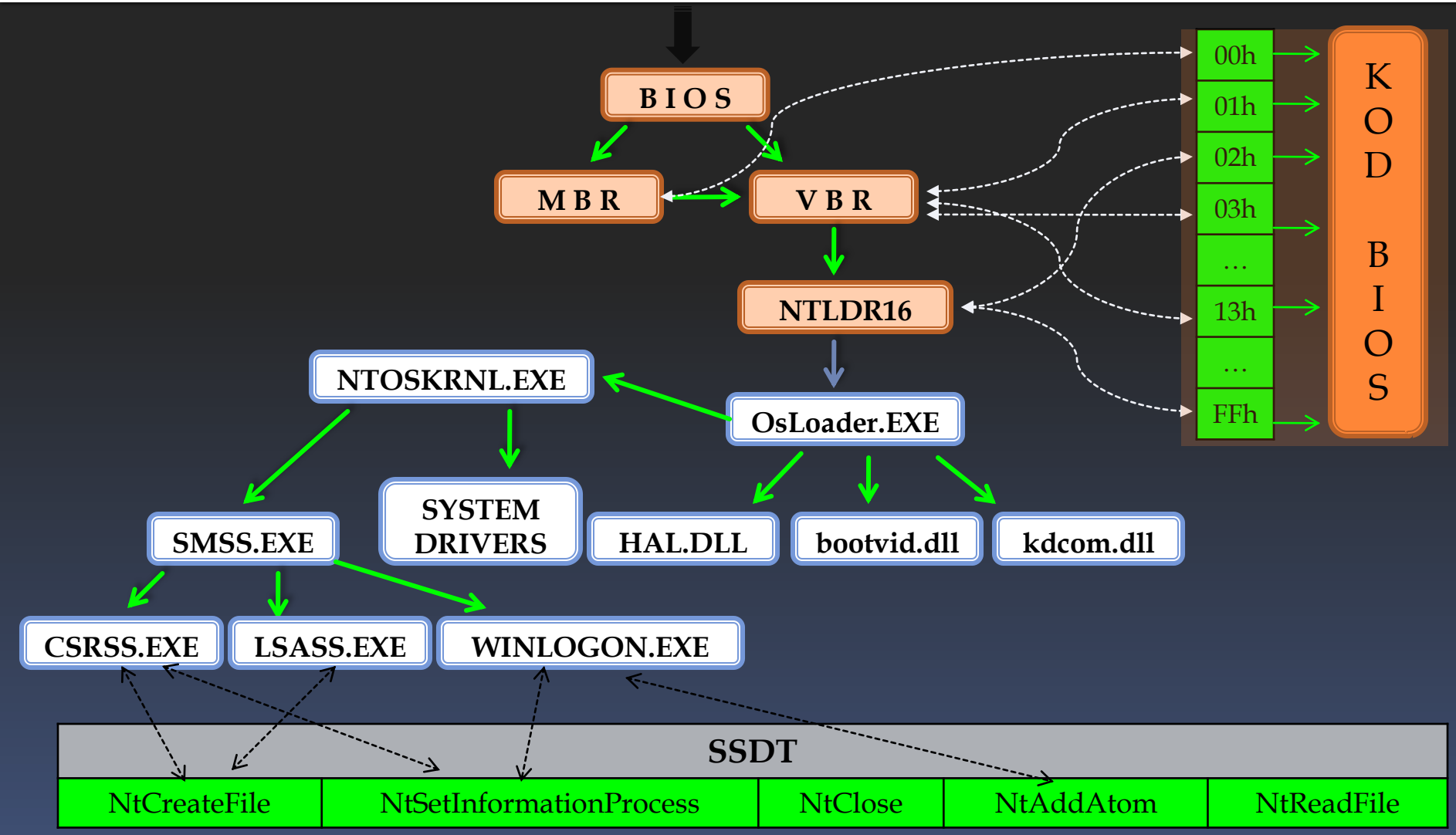
- System uruchamiany zawsze z HDD
  - MBR i VBR w akcji
- Fazy ładowania – tryb rzeczywisty i chroniony
  - Zaczynamy w *real-mode*
  - Za przełączanie w PM odpowiada wyznaczony moduł bootowania
  - Piszemy kod obsługujący oba tryby działania procesora

Windows XP SP3 – ładowanie

Ważne informacje

- Schemat działania pojedynczej fazy
  - Wykonaj konieczne operacje ładowania
  - Wczytaj kolejny etap do pamięci wirtualnej
  - Skocz do adresu docelowego
- Sposób wczytywania danych z HDD
  - Tryb rzeczywisty – przerwanie INT13h
  - Tryb chroniony – sterowniki jądra

Windows XP SP3 - ładowanie



Budzimy się!

D E M O

Microsoft Windows XP SP3

gets pwned hard

Piszemy bootkita – technikalia

Budujemy kod bootkita

- Podstawowe założenia
  - Całkowity brak operacji na dysku twardym
  - Kod znajduje się na przenośnym nośniku – wykluczamy podmianę oryginalnego MBR
  - Jak najmniejszy objętościowo – oczekiwany rozmiar to ok. 1kB kodu maszynowego
- Efekt docelowy: modyfikacja kodu jądra w sposób zwiększający nasze przywileje w systemie

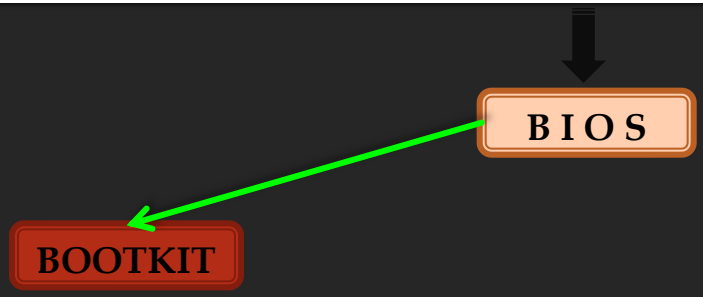
Budujemy kod bootkita

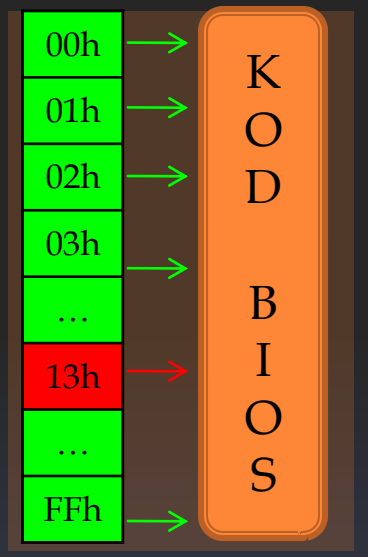
- Istotne spostrzeżenia
  - Nie można przejść bezpośrednio z kodu trybu rzeczywistego do modyfikacji kernela
    - W każdym momencie ładowania widzimy pewną część starych oraz aktualnych modułów
    - Zmusza to do tworzenia „łańcuchów” modyfikacji, przechodzących po kolejnych etapach
    - Na końcu docieramy do obrazu wykonywalnego jądra
  - Sposób na gwarancję przechwycenia wykonywania w kolejnych etapach ?

Budujemy kod bootkita

- Istotne spostrzeżenia cd.
  - Minimalizacja liczby modyfikowanych faz
    - „Pole widzenia” pamięci jest zwykle szersze niż 1 poziom
  - Przykładowe praktyczne „ulepszenia” jądra
    - Modyfikacja *boot logo* wyświetlanego przez bootvid.dll
    - Ominięcie procesu autoryzacji użytkownika
    - Podniesienie uprawnień dowolnego procesu użytkownika (i.e. cmd.exe)

Bootkit – część druga





Bootkit – część pierwsza

- Bootsektor (512b) ładowany pod adres 0x07C00
- Ładowanie MBR pod ten sam adres, w celu uruchomienia Windows
  - Realokacja w „bezpieczne” miejsce
    - Bezpieczny sposób na alokację pamięci?
  - Doczytanie pozostałych danych z napędu
  - Odczytanie pierwszego sektora HDD pod zwolniony adres, przy użyciu przerwania INT13h

Bootkit – część pierwsza

- Sposoby na utrzymanie kontroli w systemie
  - Modyfikacja kodu BIOS w pamięci
    - Niebezpieczne
    - Niewygodne
    - Niekompatybilne
  - Modyfikacja Tabeli Deskryptorów Przerwań
    - Zawiera 256 elementów – wskaźników na funkcje BIOS obsługujące konkretne funkcjonalności
    - Każdy z nich zajmuje 4 bajty – adresowanie segmentowe (16:16)
    - Znajduje się pod adresem 0000:0000h

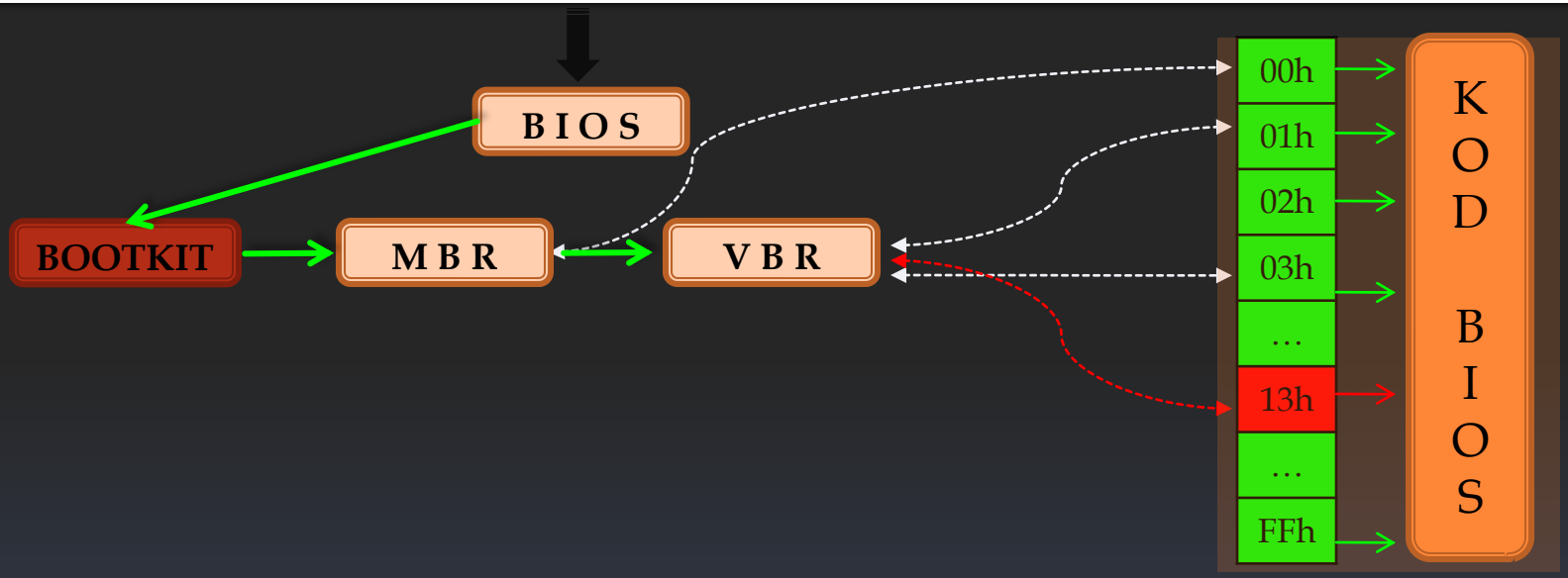
Bootkit – część pierwsza

Które przerwanie chcemy kontrolować?

Bootkit – część pierwsza

- Interesujące możliwości
  - INT 10h – Video services
  - INT 13h – Low Level Disk Services
  - INT 15h – Miscellaneous
  - INT 1Ah – Real Time Clock Services
- Wybór zależny od sposobu i miejsca modyfikacji bootloadera
- Najczęściej (zawsze?) stosowany hook na przerwanie obsługi dysku (INT 13h)

Bootkit – część pierwsza



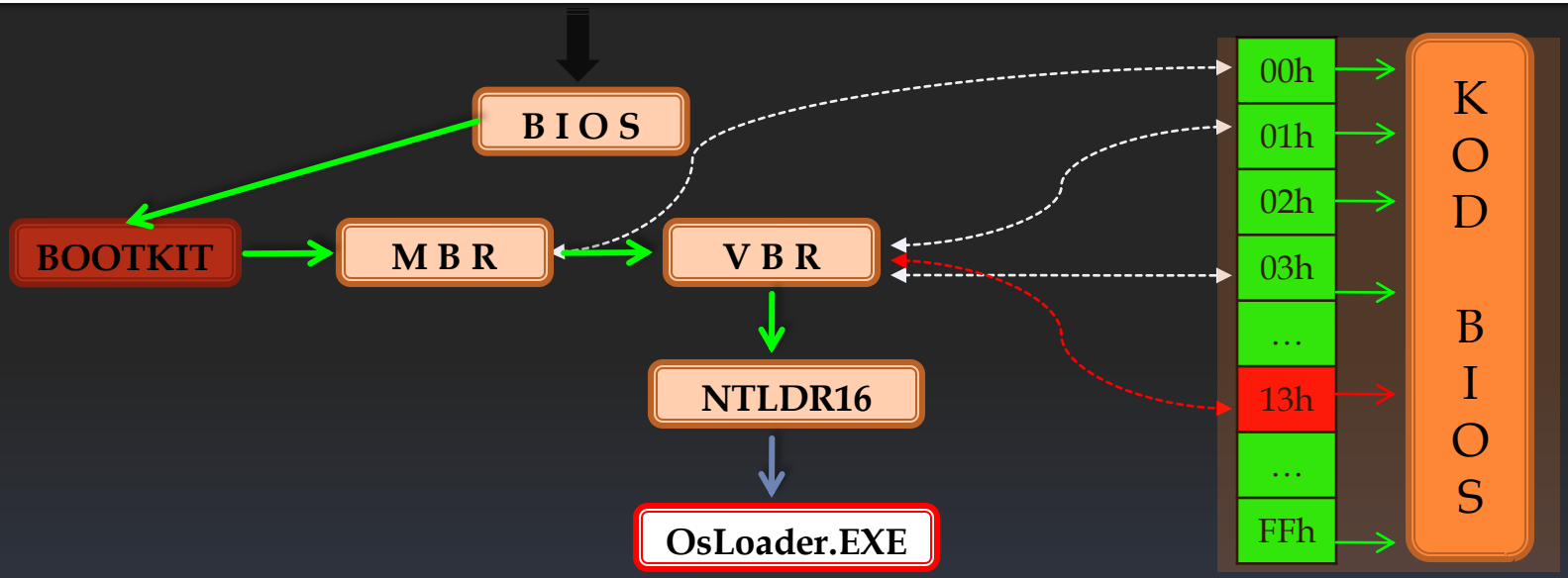
Bootkit – część druga

- INT13h hook
  - Kontrola wszystkich danych przepływających przez dysk
    - Master Boot Record
    - Volume Boot Record
    - NTLDR16
    - OsLoader.EXE
  - Nie wiemy, z jakim plikiem powiązane są odczytywane dane – operujemy na adresowaniu CHS (Cylinder-Head-Sector)

Bootkit – część druga

- Konieczność posługiwania się sygnaturami plików
  - Kwestie kompatybilności
  - Problem danych brzegowych
    - Pliki wczytywane są blokami o określonej wielkości
  - Unikalność sygnatury
- Najdalszym elementem bezpośrednio wczytywanym przez INT13h jest C:\NTLDR

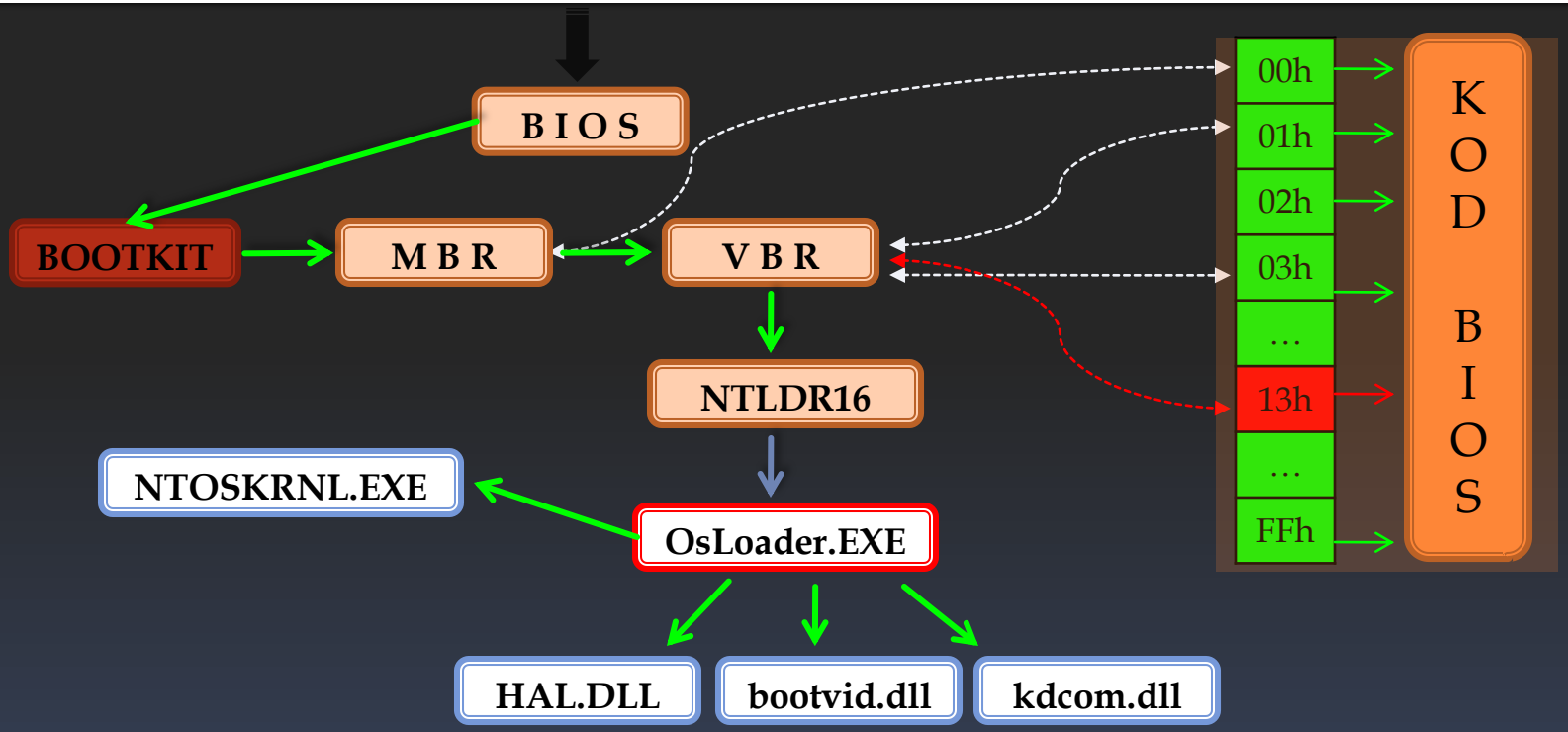
Bootkit – część druga



Bootkit – część trzecia

- Kontekst OsLoader.exe – tryb chroniony
  - Pomimo 32-bitowego adresowania, istnieje dostęp do niskich obszarów pamięci
  - Korzystając z okazji, relokujemy kod w wolne miejsce po stronie jądra (0x8??????)
  - Decyzja → co dalej?
    - Dostęp do obrazu jądra systemu (NTOSKRNL)
    - Dostęp do obrazu biblioteki obsługi sprzętu (HAL)
    - Dostęp do obrazów podstawowych sterowników

Bootkit – część trzecia



SSDT				
NtCreateFile	NtSetInformationProcess	NtClose	NtAddAtom	NtReadFile

Bootkit – część czwarta

- Pierwsza modyfikacja – podmiana logo Windows
  - Cel bezpośredni – obraz jądra NT
  - Ntoskrnl.exe zawiera bitmapę bootowania w zasobach pliku PE
  - Do wyświetlania i aktualizacji obrazu używa funkcji *VidBitBlt* i *VidBufferToScreenBlt*
  - Dwie metody zmiany bitmapy
    - Wyszukiwanie sygnatury logo w pamięci kernela i nadpisanie nowymi danymi
    - Przechwytywanie wywołań eksportów BOOTVID.DLL i wprowadzanie poprawek run-time

Bootkit – część czwarta

- Druga modyfikacja – eskalacja uprawnień linii poleceń
  - Cel bezpośredni – wszystkie instancje **cmd.exe**
  - Format przechowywania tokenów bezpieczeństwa procesów
    - Obiekty opisujące wątki i procesy znajdują się w przestrzeni jądra
    - Interesują nas konkretne pola struktury EPROCESS każdego istniejącego w systemie procesu **cmd.exe**:

```
+0x084 UniqueProcessId  : Ptr32 Void
+0x088 ActiveProcessLinks : _LIST_ENTRY
+0x0c8 Token            : _EX_FAST_REF
+0x174 ImageFileName    : [16] UChar
```

Bootkit – część czwarta

- Procesy połączone w dwukierunkowej liście łączonej
  - Podstawowa technika ukrywania w pierwszych rootkitach jądra
- Pobranie adresu struktury EPROCESS aktualnego procesu:

```
mov eax, [fs:0x124]

mov eax, [eax+0x44]
```

- Przeszukiwanie listy w poszukiwaniu procesu o PID=4 (proces SYSTEM)
- Skopiowanie znalezionego wskaźnika *Token* do struktur opisujących wszystkie procesy o nazwie **cmd.exe**

Bootkit – część czwarta

- Trzecia modyfikacja – ominięcie autoryzacji użytkownika
  - Cel bezpośredni – proces LSASS.EXE
  - Local Security Authority Subsystem Service - odpowiada za weryfikację danych logowania użytkownika
  - Jeden z używanych modułów to msv1_0.dll
    - Zawiera kod odpowiedzialny za porównywanie wprowadzonego oraz poprawnego hasha LM
  - Oczekiwany rezultat – modyfikacja dwóch bajtów w kontekście pamięci LSASS

Bootkit – część czwarta

Kod autoryzacyjny msv1_0.dll

```
.text:77C6989D push 10h
.text:77C6989F add  ebx, 34h
.text:77C698A2 push ebx
.text:77C698A3 push esi
.text:77C698A4 call RtlCompareMemory
.text:77C698AA cmp eax, 10h
.text:77C698AD jnz short loc_77C698C0
```

Bootkit – część czwarta

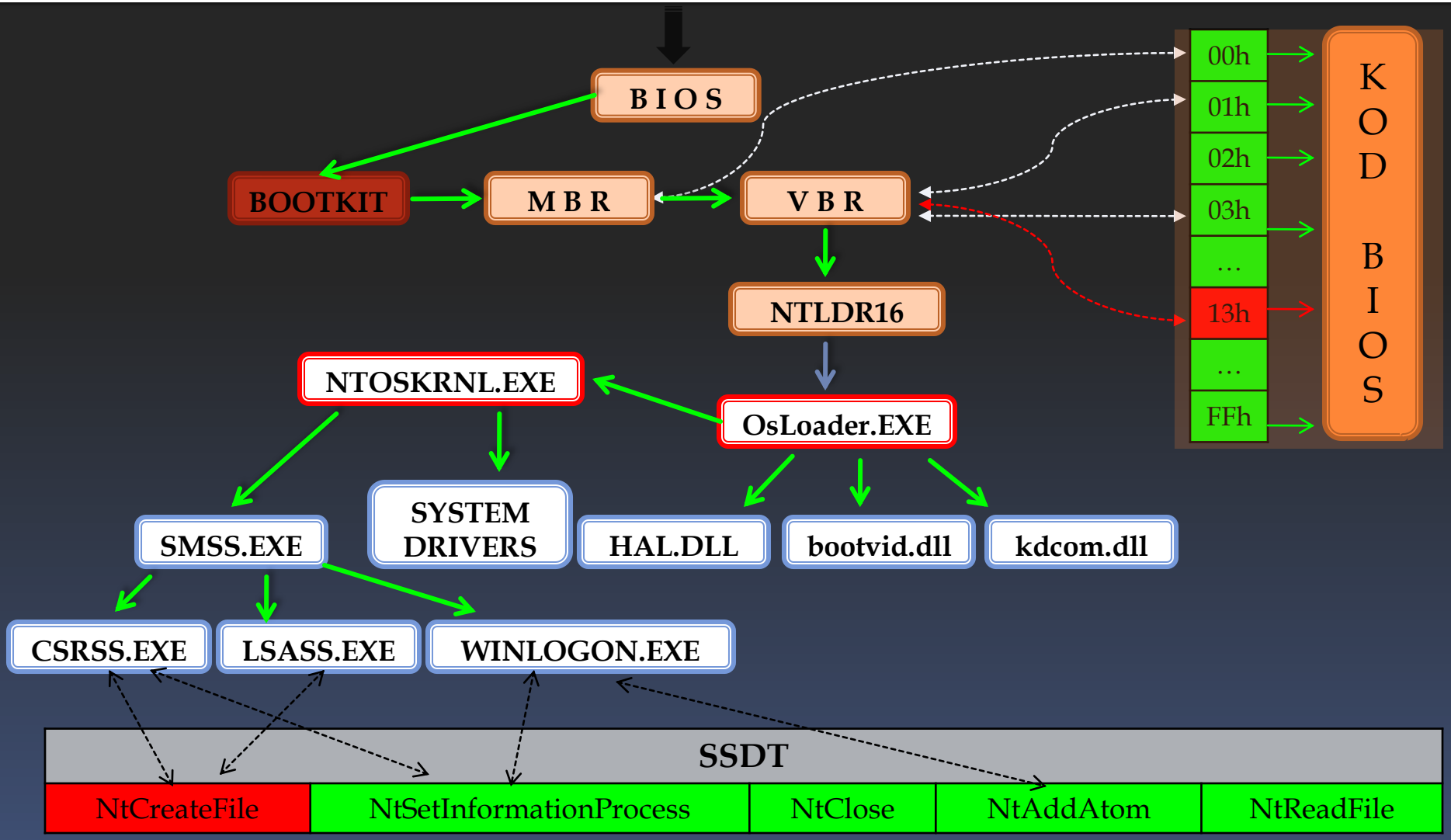
- W jaki sposób aplikujemy patche?
  - Modyfikacja logo – podmiana bitmapy możliwa na etapie OsLoader.EXE
    - Trudniejszy wariant zakłada podczepienie się pod funkcje bootvid.dll
  - Przejmowanie wykonywania dla konkretnych aplikacji
    - Patch kernel-side w kontekście cmd.exe
    - Patch user-side w kontekście lsass.exe
    - Procesy win32 korzystają z wywołań systemowych, które możemy modyfikować!

Bootkit – część czwarta

Modyfikacja pamięci wybranego procesu – schemat

1. Ustaw hook na często używany syscall (np. *NtClose, NtCreateFile*)
2. W momencie wywołania funkcji
   - Sprawdź, czy pole ImageFileName struktury EPROCESS jest równe oczekiwanemu LUB
   - Sprawdź, czy przekazane argumenty są charakterystyczne dla zadanej aplikacji
     - Przykład → Ciąg „\??\NETLOGON” jako argument funkcji NtCreateFile, przekazywany (tylko?) przez LSASS.EXE

Bootkit – część czwarta



Windows Vista SP2 / RC 7

Co nowego w procesie ładowania?

Windows Vista SP2 / RC 7

- Podzielony NTLDR
  - Windows Boot Manager – BOOTMGR
    - Zbudowany z dwóch części
      - 16-bitowy nagłówek – kod trybu rzeczywistego
      - 32-bitowy plik Portable Executable – kod trybu chronionego
  - Windows Loader – winload.exe
    - Zajmuje się wyłącznie ładowaniem konkretnego systemu

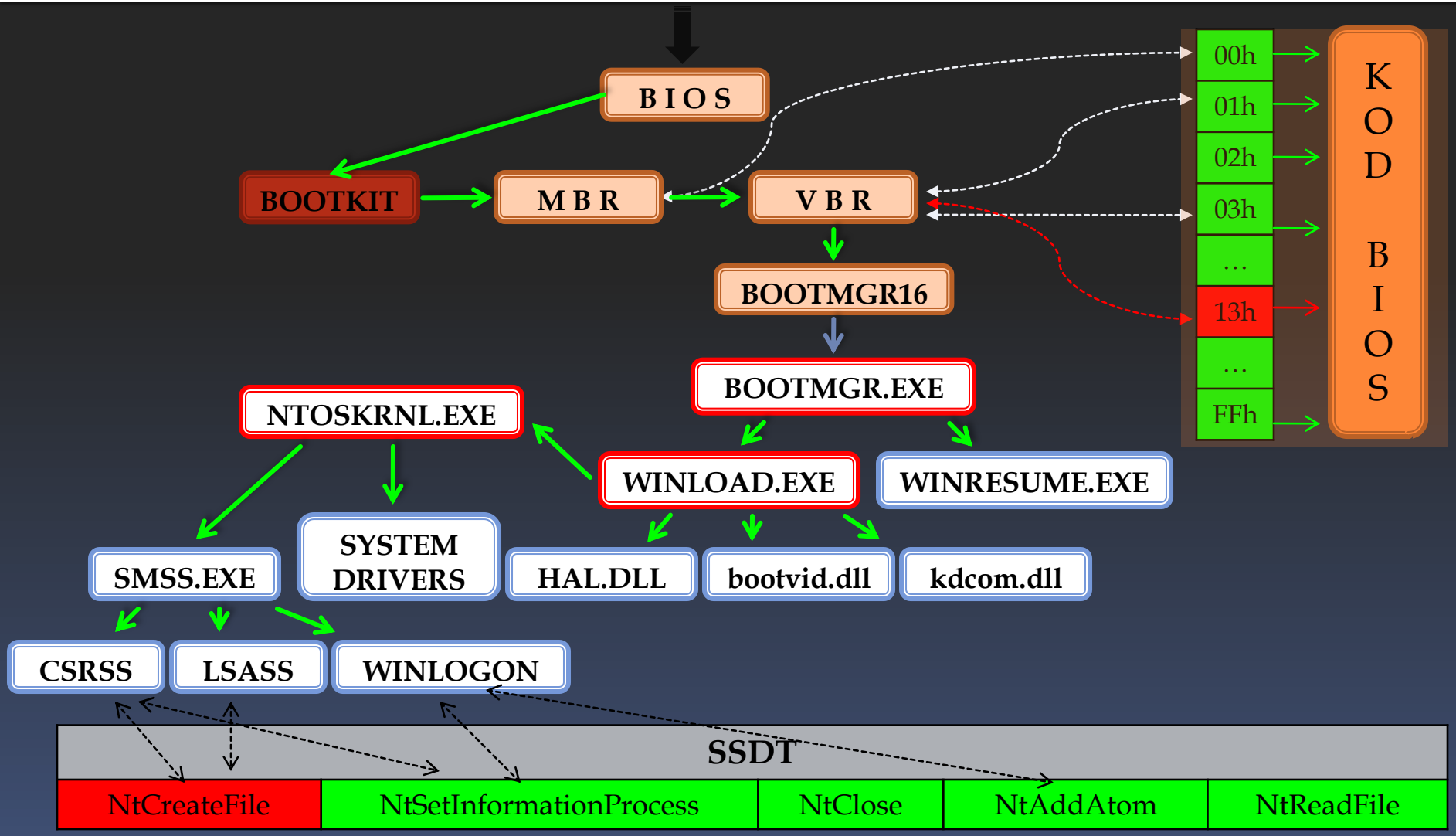
Windows Vista SP2 / RC 7

- BOOTMGR – szczegóły
  - Ładuje Boot Configuration Data z \Boot\Bcd
    - Nowy format konfiguracji – zastępuje boot.ini
  - Wyświetla menu wyboru systemu
  - W zależności od sytuacji, ładuje i uruchamia
    - Winload.exe – loader Windows Vista/7
    - Winresume.exe – wznawianie działania po hibernacji
    - NTLDR – w przypadku wyboru starszej wersji systemu

Windows Vista SP2 / RC 7

- Winload.exe – „druga część” NTLDR
  - Uruchamiany dla konkretnego OS
  - Ładuje do pamięci niezbędne moduły jądra (ntoskrnl.exe, hal.dll, …)
  - Przekazuje wykonywanie do kernela
- Winresume.exe – przywraca działanie systemu po hibernacji

Windows Vista SP2 / RC 7



Uwagi końcowe

Warunki powodzenia bootkita

- Przypadek I – atak jednorazowy z użyciem przenośnego nośnika
  - Wymaga fizycznego dostępu do atakowanej maszyny
  - Wymaga możliwości bootowania z obcego medium (Konfiguracja BIOS, hasło dostępu, zaplombowana jednostka centralna)

Uwagi końcowe

Warunki powodzenia bootkita

- Przypadek II – atak trwały z użyciem dysku twardego
  - Wymaga wykonania operacji na surowych danych dysku twardego – domyślny dostęp tylko adminów
    - Standardowe zagadnienia bezpiecznego korzystania z internetu etc.
  - Trudny (niemożliwy) w detekcji dla lokalnego systemu operacyjnego
  - Banalny w detekcji dla systemu na zewnętrznym HDD

Uwagi końcowe

- Tworzenie bootkita – pouczające doświadczenie – architektura systemów
- Ogromne zamieszanie wokół teoretycznych możliwości
  - Oraz znacznie mniej przykładów praktycznych zastosowań – jest się czego bać?
- Technologie sprzętowe – czyżby koniec „ery” bootkitów?

OBIAD!!!!111

Dziękuję za uwagę!

Q & A

E-mail: j00ru.vx@gmail.com

Tech blog: http://j00ru.vexillium.org/