Kurs administrowania systemem Linux

Zajęcia nr 11: Środowisko uruchomieniowe procesorów x86-64
Część 2: UEFI. Bootloadery

Instytut Informatyki Uniwersytetu Wrocławskiego

24 maja 2022

Było: BIOS z lat 1980-tych w komputerze z XXI wieku

- Tablica partycji nieadekwatna do dzisiejszych dysków.
  - Kodowanie CHS (24 bity) pozwala zaadresować $\leq$ 8GiB — od lat nie używane.
  - Numeracja LBA (32 bity) pozwala zaadresować $\leq$ 2TiB.
  - Podział tylko na 4 partycje (plus niewygodne partycje rozszerzone).
- Ładowanie programu z MBR i konieczność jego *bootstrapowania* z sektorów zapisanych poza partycjami.
- Konieczność uruchamiania programu w trybie 16-bitowym.

Jak uwolnić się od zaszłości z lat '80-tych?

Unified Extensible Firmware Interface

- Nowy format partycjonowania dysków: GPT.
- UEFI uruchamia jądro bootloadera w rodzimym trybie procesora (64- lub 32-bit) — nowe procesory mogą pomijać tryb 16-bitowy.
- Uruchamiany program jest w całości zapisany w pliku umieszczonym w systemie plików FAT w specjalnej partycji (ESP) — wygoda instalacji bootloadera.
- Wbudowany *boot manager*: może być wiele programów w wielu partycjach ESP.
- Nowa wersja API usług: *boot* i *runtime services*.
- Niezależny od architektury procesora.
- Łatwy i ustandaryzowany dostęp do *variable services* (*key-value*) z poziomu systemu operacyjnego — możliwość odczytywania i zmiany ustawień z poziomu OS.
- *Compatibility Support Module* (CSM).
- Dobre wsparcie dla *Preboot eXecution Environment*, w szczególności obsługa sieci.
- *Secure boot*

Krytyka UEFI

- Nie rozwiązuje problemu podwójnych sterowników — OS, podobnie jak w przypadku BIOS-u, implementuje własną obsługę urządzeń.
- Podobnie jak ACPI odbiera użytkownikowi kontrolę nad sprzętem (Ronald G. Minnich (Coreboot), Cory Doctorow).
- Secure Boot przyjęty (niesłusznie) jako próba odebrania użytkownikowi możliwości wyboru oprogramowania — jedyne centrum certyfikacji prowadzi Microsoft.
- Duże i skomplikowane oprogramowanie o dużych możliwościach (dostęp do dysków, kart sieciowych itd.), niebędące *Open Source*.
- Wielka liczba błędów w implementacjach, szczególnie w początkowym okresie.

GUID Partition Table

- Sektor 0: *Hybrid MBR* — Legacy BIOS nie zauważa różnicy. *Protective MBR* — stare oprogramowanie rozumie, że nie rozumie.
- Sektor 1 i ostatni dysku: nagłówek tablicy partycji. Zawiera m. in. rozmiar tablicy partycji (zwykle 128 wpisów).
- Sektory 2–33: tablica 128 partycji (powtórzona na końcu dysku).
- Opis partycji: 128 bajtów (w MBR — 16 bajtów).

hex	rozmiar	Znaczenie
0- f	16	Typ partycji (GUID)
10-1f	16	GUID partycji
20-27	8	LBA pierwszego sektora (little endian)
28-2f	8	LBA ostatniego sektora (włącznie)
30-37	8	Atrybuty (np. bit 60: read-only)
38-7f	72	Nazwa (etykieta) partycji (36 znaków UTF-16LE)

Typy partycji GPT

00000000-0000-0000-0000-000000000000	Unused
C12A7328-F81F-11D2-BA4B-00A0C93EC93B	EFI System Partition
21686148-6449-6E6F-744E-656564454649	BIOS boot partition
4F68BCE3-E8CD-4DB1-96E7-FBCAF984B709	Linux root partition (x86-64)
0657FD6D-A4AB-43C4-84E5-0933C84B4F4F	Linux swap
0FC63DAF-8483-4772-8E79-3D69D8477DE4	Linux data
7FFEC5C9-2D00-49B7-8941-3EA10A5586B7	plain dm-crypt
CA7D7CCB-63ED-4C53-861C-1742536059CC	LUKS
E6D6D379-F507-44C2-A23C-238F2A3DF928	LVM
933AC7E1-2EB4-4F13-B844-0E14E2AEF915	/home
3B8F8425-20E0-4F3B-907F-1A25A76F98E8	/srv

BIOS boot partition GUID zapisany w tablicy partycji (little endian) i przeczytany jako ASCII daje: `Hah!IdontNeedEFI`.

Używanie GPT

- Dzięki *Hybrid MBR* GPT można stosować także w starszych komputerach nie wyposażonych w UEFI.
- Dobre wsparcie ze strony większości współczesnych systemów — nie warto używać partycji DOS.
- Narzędzia: `fdisk`, `parted`.
- Dzięki 64-bitowym LBA jest niezbędny w dyskach o pojemności $\geq$ 2TiB.
- UEFI wymaga GPT. GPT nie wymaga UEFI, ale w niektórych komputerach (Lenovo) *legacy BIOS* nie jest dostępny w razie użycia GPT.

UEFI

Zawartość partycji ESP dysku twardego:

`\EFI\system-operacyjny\bootloader.EFI`

np.

`\EFI\DEBIAN\GRUBX64.EFI`

Wymaga skonfigurowania. *Removable path* na dyskach wymiennych (FAT32, FAT16 lub FAT12) — tylko jeden bootloader na dysku:

`\EFI\BOOT\BOOTX64.EFI`

Konfigurowanie UEFI:

- Interfejs graficzny podobny do interfejsu BIOS.
- UEFI Shell.
- UEFI variable services.

EFI shell — polecenia

```
alias attrib bcfg cd cls comp connect cp date dblk del devices devtree dh
dir disconnect dmem dmpstore drivers drvcfg drvdiag echo edit eficompress
efidecompress else endfor endif exit for getmtc goto help hexedit if
ifconfig load loadpcirom ls map mem memmap mkdir mm mode mv openinfo parse
pause pci ping reconnect reset rm sermode set setsize setvar shift
smbiosview stall time timezone touch type unload ver vol
```

Najciekawsze *boot config*:

```
bcfg [driver|boot] dump [-v]
bcfg [driver|boot] add N file "desc"
bcfg [driver|boot] addp N file "desc"
bcfg [driver|boot] addh N handle "desc"
bcfg [driver|boot] rm N
bcfg [driver|boot] mv N M
```

EFI shell

- Możliwość wygodnej edycji ustawień (zamiast interfejsu graficznego).
- Możliwość tworzenia skryptów wykonywanych podczas uruchamiania komputera.
- Wiele płyt głównych (szczególnie starszych) zawiera niekompletne lub niedziałające powłoki (np. bez `bcfg`).

Variable services

- Dostęp w Linuksie w katalogach `/sys/firmware/efi/efivars/` (punkt montowania efivarfs) i `/sys/firmware/efi/vars/` (legacy).
- Można czytać i pisać.
- Wygodne narzędzie: `efibootmgr`.

Listowanie zawartości tablicy bootowania (`-v` = verbose):

```
efibootmgr [-v]
```

Usuwanie wpisu:

```
efibootmgr -b NNNN -B
```

Dodawanie wpisu (`-c` = create):

```
efibootmgr -c -d /dev/device -l '\EFI\BOOT\BOOTX64.EFI' -L 'label'
```

Kolejność bootowania:

```
efibootmgr -o XXXX,YYYY,ZZZZ
```

Rozruch komputera z UEFI

- UEFI samodzielnie przełącza procesor do *long mode* (lub *protected mode* z *flat segmentation*) i wstępnie konfiguruje stronicowanie (*identity paging*).
- *Bootloader* jest programem wykonywalnym w formacie PE/COFF (Microsoft Windows 64 lub 32-bit) składowanym w pliku w systemie FAT w partycji ESP.
- UEFI uruchamia go w trybie użytkownika (*ring 2*). W trybie nadzorcy pracuje UEFI.
- UEFI udostępnia *bootloaderowi boot services*, np. `GetMemoryMap()`.
- *Bootloader* przejmuje tryb nadzorcy wykonując `ExitBootServices()`. Od tej chwili *boot services* są niedostępne. *Bootloader* może zwolnić pamięć zarezerwowaną dla *boot services*, ale nie wolno mu ruszać pamięci *runtime services*.
- Podczas pracy *bootloadera* i systemu operacyjnego są dostępne *runtime services*, np. *variable services*.

UEFI → *bootloader* → jądro = problemy?

- *Bootloader* wywołuje `ExitBootServices()` uniemożliwiając jądru poważną rozmowę z *EFI firmware*.
- W początkowym okresie wdrażania (∼2012) *firmware* UEFI zawierał dużą liczbę błędów, wymagających interwencji jądra.

H. Peter Anvin (Intel), 2011-06-08, linux-kernel list

However, I suspect that what we *should* do is carry a kernel EFI stub to go along with the BIOS stub... otherwise we're forever at mercy of getting all the boot loader authors to change in lockstep, and there are specific ones which are notoriously hard to work with.
The "kernel carries its own stub" approach been very successful in dealing with BIOS, and would make a lot of sense to me for EFI as well.

Linux EFI stub

- Zwykły *vanilla kernel* potrafi się uruchomić bezpośrednio z UEFI.
- UEFI zna tylko partycje FAT, dlatego jądro i *initramfs* trzeba skopiować do ESP.
- Jądro (które jeszcze nie ma dostępu do dysków) może poprosić UEFI o załadowanie *initramfs* (poprzez *UEFI file services*).

```
cp /boot/{vmlinuz,initrd.img} /boot/efi/EFI/debian/
efibootmgr -c -g -L "Debian (EFI stub)"
   -l '\EFI\debian\vmlinuz' -u "root=UUID=$UUID ro quiet
      rootfstype=ext4 add_efi_memmap
      initrd=\\EFI\\debian\\initrd.img"
```

Matt Fleming (Intel)

The EFI boot stub is a particularly good solution if you wish to simply boot a Linux kernel as quickly as possible, as it cuts out the bootloader stage of the traditional boot process. Furthermore, [...] due to the rapid development cycle of the Linux kernel the EFI boot stub has become the most robust UEFI boot solution.

Aplikacje działające pod kontrolą UEFI

- Póki aplikacja nie wywoła `ExitBootServices()` platformą zarządza UEFI. Aplikacja może np. zwrócić sterowanie do *firmware'u* (np. opcja *System Setup* w Grub-ie).
- Wywoływana aplikacja otrzymuje od UEFI *System Table* i nie musi samodzielnie skanować platformy (wykrywanie *Extended BIOS Data Area*, *System Management BIOS*, tablic ACPI, magistral PCI itd.)
- Zamiast archaicznych przerwań BIOS-u — usługi pogrupowane w *protokoły*.
- UEFI posiada nawet interpreter *bytecode'u* pozwalający na pisanie sterowników niezależnych od platformy.

Tworzenie aplikacji działających pod kontrolą EFI

- EDK II (*EFI Development Kit*) Intela (TianoCore): tworzenie aplikacji działających pod UEFI oraz tworzenie *firmware'u* UEFI.
- GNU-EFI — biblioteki pozwalające tylko na tworzenie aplikacji dla EFI.

Tworzenie aplikacji z użyciem GNU-EFI

```
#include <efi.h>
#include <efilib.h>
EFI_STATUS
EFIAPI
efi_main (EFI_HANDLE ImageHandle, EFI_SYSTEM_TABLE *SystemTable) {
   InitializeLib(ImageHandle, SystemTable);
   Print("Hello, world!\n");
   return EFI_SUCCESS;
}
```

Zob.:

- Roderick W. Smith, *Programming for EFI*, `http://www.rodsbooks.com/efi-programming/`
- UEFI Bare Bones `https://wiki.osdev.org/UEFI_Bare_Bones`

Klasy urządzeń UEFI

CSM (*Compatibility Support Module*): opcjonalna usługa UEFI pozwalająca na uruchomienie komputera w trybie zgodności z BIOS.

UEFI class 0 device — urządzenie nie posiadające UEFI, tylko legacy BIOS.

UEFI class 1 device — UEFI pracuje wyłącznie w trybie CSM.

UEFI class 2 device — urządzenie pracuje w trybie UEFI, ale ma możliwość włączenia CSM.

UEFI class 3 device — urządzenie UEFI nie wyposażone w CSM.

Dawniej bywało źle...

Booting Linux via UEFI can 'brick' some Samsung laptops
Katherine Noyes, PCWorld, Jan 31, 2013

A problem with a kernel driver for Samsung laptops has caused numerous users to find their machines "bricked" after trying to boot Linux on them.

That's according to several reports on the Ubuntu Linux bug tracker that name the 530U3C, NP700Z7C, NP700Z5C, and 300E5C series as among the Samsung PCs involved in the problem, which occurs when users boot Linux via the Unified Extensible Firmware Interface (UEFI).

One user filing a report has apparently lost two laptops due to the problem.

Dawniej bywało źle (2)

Samsung Laptops Bricked by Booting Linux Using UEFI
Jarred Walton, Jan. 30, 2013

Samsung's UEFI implementation appears to be faulty. It was most likely tested with Windows only and found to work, but thorough testing with other operating systems doesn't appear to have been a priority—or perhaps a consideration at all. [...]

Long-term, we expect Samsung to release BIOS and firmware updates for the affected laptops, though how long that might take is unknown. Short-term, the workaround is for Linux to boot these Samsung models using the Compatibility Support Module (CSM), which basically bypasses the UEFI bootloader, but dual-booting via CSM appears to be a bit complex. [...]

It appears the problem stems from NVRAM corruption. Removing power, opening the laptop up, and disconnecting the CMOS battery appears like it will clear the problem, but that's a pretty serious set of steps to take for most laptops.

Znowu Lenovo...

Matthew Garrett: More in the series of bizarre UEFI bugs

Every UEFI boot entry has a descriptive string. This is used by the firmware when it's presenting a menu to users — instead of "Hard drive 0" and "USB drive 3", the firmware can list "Windows Boot Manager" and "Fedora Linux". There's no reason at all for the firmware to be parsing these strings. But the evidence seemed pretty strong — given two identical boot entries, one saying "Windows Boot Manager" and one not, only the first would work. At this point I downloaded a copy of the firmware and started poking at it. Turns out that yes, actually, there is a function that compares the descriptive string against "Windows Boot Manager" and appears to return an error if it doesn't match. What's stranger is that it also checks for "Red Hat Enterprise Linux" and lets that one work as well. (Nov. 14th, 2012)

Znowu Lenovo...

Matthew Garrett: More in the series of bizarre UEFI bugs

Every UEFI boot entry has a descriptive string. This is used by the firmware when it's presenting a menu to users — instead of "Hard drive 0" and "USB drive 3", the firmware can list "Windows Boot Manager" and "Fedora Linux". There's no reason at all for the firmware to be parsing these strings. But the evidence seemed pretty strong — given two identical boot entries, one saying "Windows Boot Manager" and one not, only the first would work. At this point I downloaded a copy of the firmware and started poking at it. Turns out that yes, actually, there is a function that compares the descriptive string against "Windows Boot Manager" and appears to return an error if it doesn't match. What's stranger is that it also checks for "Red Hat Enterprise Linux" and lets that one work as well. (Nov. 14th, 2012)

I recommend drinking, because as far as I know they haven't actually got around to doing anything useful about this yet. (Feb. 1, 2013)

Secure Boot

Cztery zmienne UEFI:

PK (Platform Key) Część publiczna głównego klucza, który służy do podpisywania kluczy w KEK. Posiadanie części prywatnej PK daje władzę nad maszyną.

KEK (Key Exchange Key) Część publiczna klucza służącego dawniej do podpisywania aplikacji, a obecnie do podpisywania elementów baz db i dbx.

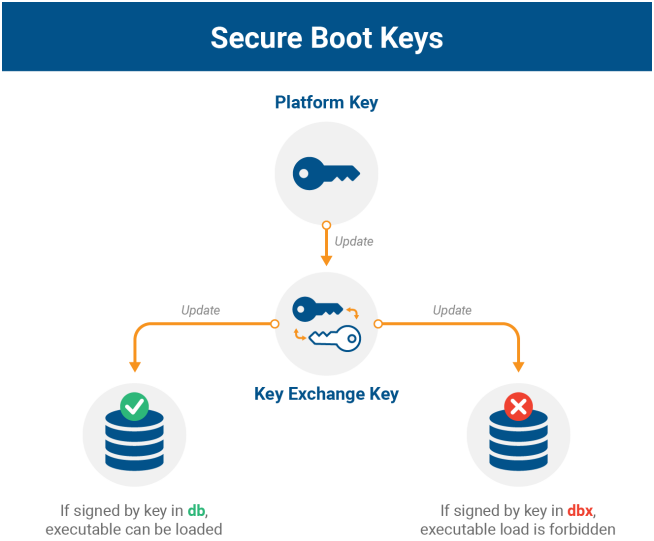
db (Whitelist database) Lista dodatkowych kluczy, sygnatur i haszy akceptowanych przez platformę.

dbx (Blacklist database) Lista wycofanych kluczy, sygnatur i haszy.

Aplikacje UEFI (w formacie PE/COFF) mogą zawierać sygnatury X.509. Jeśli SB jest włączone, to aplikacja UEFI zostanie uruchomiona tylko wtedy, gdy:

- zawiera sygnaturę podpisaną przez klucz KEK lub znajdujący się w db i nie znajdujący się w dbx,
- jej sygnatura znajduje się w db i nie znajduje w dbx,
- jej hasz znajduje się w db i nie znajduje w dbx.

Klucze Secure Boot



Tryby Secure Boot

Setup Mode Brak klucza PK. Można modyfikować zmienne PK, KEK, db i dbx bez potrzeby uwierzytelniania.

User Mode Wpisany klucz PK. Modyfikacje zmiennych SB możliwe jedynie przez aplikacje podpisane przez PK.

Deployed Mode Wpisany klucz PK. Brak możliwości modyfikacji zmiennych SB.

Audit Mode Platforma pozwala na weryfikację sygnatur, ale nie odmawia uruchomienia aplikacji nie zawierających poprawnych sygnatur.

Nie bójcie się!

- W specyfikacji UEFI moduł *Secure Boot* jest opcjonalny. Producent może go pominąć, pozwalać na jego wyłączenie lub włączyć na stałe.
- Microsoft prowadzi jedyne centrum certyfikacji (tłumaczy się, że nikt inny nie chciał).
- Obawiano się, że Microsoft uniemożliwi na platformach windowsowych instalację innych systemów (SB = *hardware restriction*, *hardware DRM*).

Sebastian Anthony, *Microsoft could lock out Linux on Windows 8 PCs, but it won't*, September 21, 2011

Ultimately, there's no valid reason for Microsoft locking Linux out of the UEFI firmware, and so it won't. In doing so it would spark the most monumental antitrust lawsuit possible, and it's incredibly unlikely that Microsoft cares that much about the 1 or 2% of users who install Linux on a Windows PC.

`https://www.extremetech.com/computing/96909-microsoft-could-lock-out-linux-on-windows-8-pcs-but-it-wont`

. . . *Secure Boot* da się wyłączyć! (zwykle)

Hardware Compatibility Specification for Systems for Windows 10

System.Fundamentals.Firmware.UEFISecureBoot

(Optional for systems intended to be locked down) Enable/Disable Secure Boot. **A physically present user must be allowed to disable Secure Boot via firmware setup without possession of PKpriv**. A Windows Server may also disable Secure Boot remotely using a strongly authenticated (preferably public-key based) out-of-band management connection, such as to a baseboard management controller or service processor. Programmatic disabling of Secure Boot either during Boot Services or after exiting EFI Boot Services MUST NOT be possible.

Own your platform!

19. (Optional for systems intended to be locked down) The platform MUST implement the ability for a physically present user to select between two Secure Boot modes in firmware setup: “Custom” and “Standard”. Custom Mode allows for more flexibility as specified in the following:

A. It shall be possible for a physically present user to use the Custom Mode firmware setup option to modify the contents of the Secure Boot signature databases and the PK. This may be implemented by simply providing the option to clear all Secure Boot databases (PK, KEK, db, dbx), which puts the system into setup mode.

B. If the user ends up deleting the PK then, upon exiting the Custom Mode firmware setup, the system is operating in Setup Mode with SecureBoot turned off.

C. The firmware setup shall indicate if Secure Boot is turned on, and if it is operated in Standard or Custom Mode. The firmware setup must provide an option to return from Custom to Standard Mode which restores the factory defaults.

Nie zawsze da się wyłączyć

18. For devices which are designed to always boot with a specific Secure Boot configuration, the two requirements below to support Custom Mode and the ability to disable Secure Boot are optional.

Dalej:

Optional for systems intended to be locked down.

Co to jest „system intended to be locked down”?

Do wersji 1703 (2015) także:

Disabling Secure Boot MUST NOT be possible on ARM systems.

Dygresja: Hasła do urządzeń

3Com SuperStack 3300

- Producent: “In case of lost password the device has to be sent to the manufacturer for password reset.” ($200)
- Google: “You can type in username 3comcso and password RIP000”.

Opcje dla Linuksa

- Wyłączyć *Secure Boot* w ustawieniach UEFI.
- Przejść do trybu *Setup* i zainstalować własny klucz PK, po czym samodzielnie podpisywać oprogramowanie.
- Skorzystać z usług Microsoftu i poprosić o podpisanie oprogramowania:
  - Jedyne publiczne centrum certyfikacji (które posiada PKpriv), prowadzi Microsoft.
  - Dla platformy x86-64 Microsoft podpisuje aplikacje UEFI za niewielką opłatą.

Dystrybucje podpisujące jądro

- Fedora (od wersji 18, 15/01/2013), RHEL (od wersji 7, 10/06/2014), i CentOS (od wersji 7, 7/07/2014),
- Ubuntu (od wersji 12.04.2, 14/02/2013),
- openSUSE (od wersji 12.3, 13/03/2013),
- Debian (od wersji 10 Buster, 6/07/2019),
- FreeBSD ma w planach.

Oryginalne rozwiązanie Ubuntu

- *shim* — bardzo mały *pre-bootloader* podpisany przez Microsoft. Zawiera klucz publiczny Ubuntu. Weryfikuje sygnaturę GRUB-a przed załadowaniem.
- GRUB nie używa modułów. Weryfikuje sygnaturę jądra przed załadowaniem.
- Moduły jądra nie są podpisane.

Obecnie Ubuntu zapewnia pełną weryfikację, łącznie z modułami.

Secure Boot w Debianie

- Głównym celem SB jest blokowanie *rootkitów*.
- Aby SB spełniało swoją rolę **cały kod**, począwszy od aplikacji UEFI, poprzez *bootloader*, jądro, aż po moduły jądra powinien być zweryfikowany przed uruchomieniem — to wymaga przerobienia mechanizmów ładowania modułów jądra (i GRUB-a).
- Sygnatury są generowane przez *Debian Archive Key* – wymaga dostosowania mechanizmów tworzenia pakietów źródłowych i kompilowania pakietów binarnych.
- Stąd sześcioletnie opóźnienie w stosunku do innych dystrybucji, które często stosowały rozwiązanie „na skróty" (np. brak weryfikacji modułów jądra w starszych dystrybucjach Ubuntu).
- Wykorzystuje *shim* i `sbtools` z Ubuntu.
- Zapewnia pełną weryfikację od *bootloadera* po moduły jądra.
- Od wersji Buster (6/07/2019) instalator Debiana pozwala zainstalować system z włączonym SB.

Secure Boot w Debianie — problemy

- Moduły kompilowane ze źródeł, np. za pomoca *Module Assistanta* (m-a) lub *Dynamic Kernel Module Support Framework* (DKMS) nie są podpisywane, więc nie można ich załadować przy włączonym SB.
- Wiele pakietów, które kompilują moduły jądra podczas instalacji (np. moduł jądra `vboxdrv` dla Virtualboksa) nie będzie działać — jądro odmówi załadowania takiego modułu.
- Rozwiązanie: wygenerowanie i zaintstalowanie (`mokutil`) własnego klucza PK w UEFI i podpisywanie nim skompilowanych modułów.
- Oracle Virtualbox od wersji 6.0.10 (16/07/2019) wspiera podpisywanie modułów jądra.

There is a hole in the boot (2020)!

Chain of trust w UEFI

- PK wgrany na płytę główną przez producenta.
- KEK (podpisany przez PK).
- Klucz publiczny Microsoft UEFI CA w db (podpisany przez KEK).
- `shim` w ESP — Vendor Certification (np. Canonical, Debian itp.), podpisany przez klucz z db.
- `grubx64.efi` podpisany przez Vendor Cert z `shim`-a.

CVE-2020-10713

- „Crafted `grub.cfg` file can lead to arbitrary code execution during boot process."
- Plik konfiguracyjny `grub.cfg` nie jest weryfikowany (bo jak?)
- Parser pliku `grub.cfg` cierpi na *buffer overflow vulnerability*.
- Atakujący może zamienić oryginalny bootloader Windows na (podpisanego) Gruba z podatnością i uruchomić system z rootkitem.

Boothole

- Grub2: flex+bison użyte do parsowania pliku `grub.cfg`.
- Ograniczenie na długość tokenów we fleksie: `YYLMAX`.
- Użytkownik fleksa musi zaprogramować własne makro `YY_FATAL_ERROR`, które zrywa działanie, jeśli nie można dalej parsować.

Kod `flex`-a:

```
#define YY_DO_BEFORE_ACTION \
   yyg->yytext_ptr = yy_bp; \
   yyleng = (int) (yy_cp - yy_bp); \
   yyg->yy_hold_char = *yy_cp; \
   *yy_cp = '\0'; \
   if ( yyleng >= YYLMAX ) \
      YY_FATAL_ERROR( "token too large, exceeds YYLMAX" ); \
   yy_flex_strncpy( yytext, yyg->yytext_ptr, yyleng + 1 , yyscanner); \
   yyg->yy_c_buf_p = yy_cp;
```

Boothole

Implementacja makra YY_FATAL_ERROR w Grub2:

```
#define YY_FATAL_ERROR(msg)                                  \
   do {                                                      \
      grub_printf (_("fatal error: %s\n"), _(msg));          \
   } while (0)
```

Teraz wystarczy popatrzeć co i czym można nadpisać...

Wybór *bootloaderów* w Linuksie

Bootloader + boot manager (BIOS, UEFI)

- GNU Grub 2
- SYSLINUX & Co.

Bootloader (UEFI)

- EFI Stub

Boot manager (UEFI)

- systemd-boot aka Gummiboot
- rEFInd (dawniej rEFIt)

Inne

- kexec

Historyczne

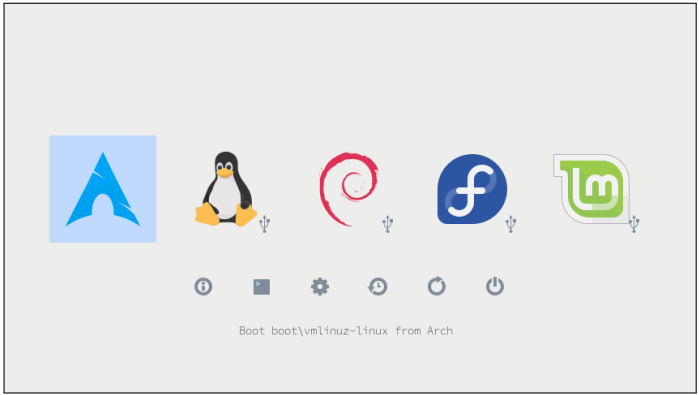
- loadlin (DOS)
- GRUB Legacy
- LILO (Linux Loader)
- ELILO

kexec

- *kernel* `exec`
- Jądro Linuksa ładuje inne jądro Linuksa i przekazuje mu sterowanie.
- Sposób na *hot reboot* bez konieczności powracania do BIOS/UEFI.
- Wygodne w środowisku serwerowym dla skrócenia czasu *rebootu* po aktualizacji jądra.

rEFInd

- Graficzny *boot manager* dla UEFI
- Pozwala konfigurować UEFI *preboot environment*
- *themeable*
- Chętnie instalowany na komputerach Apple z dual-boot Linuksem
- Może uruchamiać *efi stub* Linuksa lub np. Grub-a



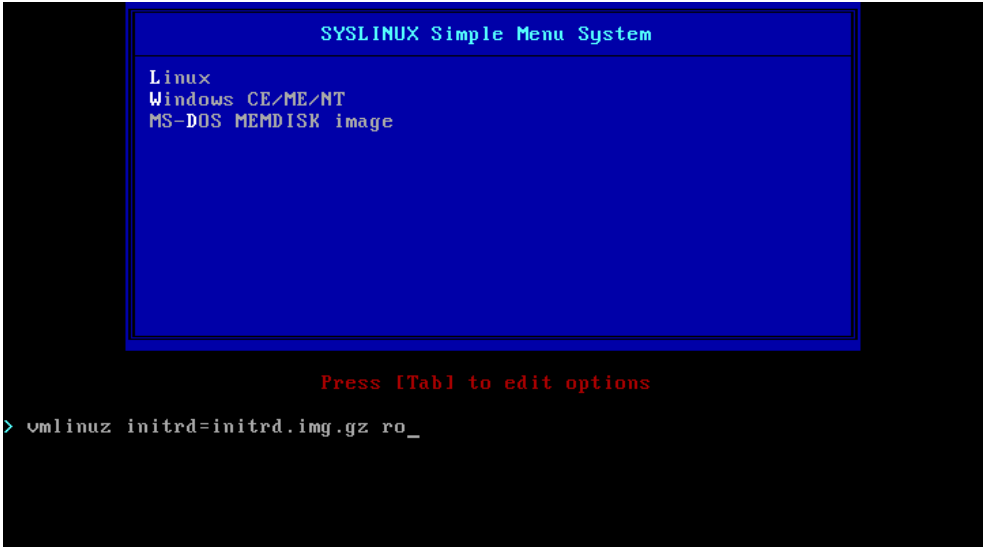
SYSLINUX

Zbiór specjalizowanych *boot managerów* i *bootloaderów* dla Linuksa (BIOS, UEFI) które uruchamiają Linuksa:

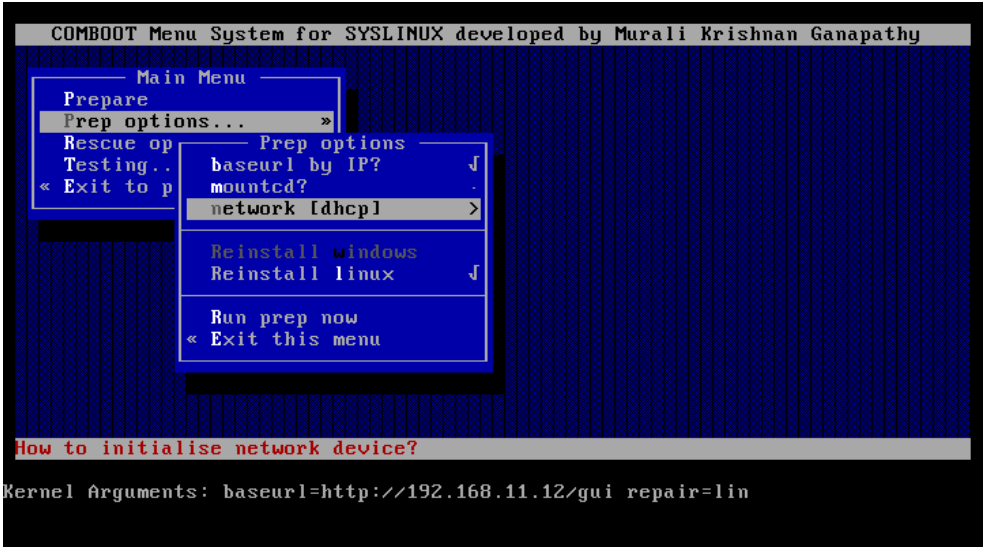
- SYSLINUX: z partycji FAT,
- EXTLINUX: z partycji ext2/3/4,
- ISOLINUX: z płyty CD,
- PXELINUX: poprzez PXE (*network boot*).

W odróżnieniu od Grub-a (*one size fits all*), filozofia typowa dla Uniksa (małe narzędzie, które robi dobrze jedną rzecz).

Proste menu SYSLINUX-a



Zaawansowane menu SYSLINUX-a



Grub 2 w Debianie

- Dla każdej architektury: `pc` (PC BIOS), `efi-amd64` (PC UEFI), `efi-ia32`, `ieee1275`, `emu`, `xen` po dwa pakiety: `grub-`*`arch`*`-bin` i `grub-`*`arch`*.
- Pakiety `grub-`*`arch`*`-bin` zawierają binaria na konkretną architekturę.
- Pakiety `grub-`*`arch`* zawierają *instalator* na konkretną architekturę. Wykluczają się wzajemnie.
- Pakiety `grub-`*`arch`*`-bin` pociągają `grub-common` i `grub2-common`.
- Ciekawe pakiety opcjonalne: `grub-doc`, `grub-splashimages`, `grub2-splashimages`, `grub-theme-starfield`.
- Instalacja wybranej wersji Grub-a: zainstalować wybrany pakiet `grub-`*`arch`*.

Alternatywna instalacja Grub-a w Debianie

- Zainstalować (nawet więcej niż jeden) pakiet `grub-`*`arch`*`-bin` i wykonywać samodzielnie `grub-install`.
- `update-grub` jest w pakiecie `grub2-common` ale `/etc/kernel/postinst.d/zz-update-grub` instalują pakiety `grub-`*`arch`*.
- Odpowiedni *hook* w `/etc/kernel/postinst.d/` lepiej napisać samemu.
- Można łatwo skonfigurować pendrive uruchamiany zarówno z BIOS-u (MBR) jak i UEFI.

Grub *home directory* w UEFI

W Debianie tradycyjnie:

- Partycja ESP zawiera tylko `grubx64.efi`. Montowana w `/boot/efi`.
- Partycja `/boot` niezaszyfrowana, zawiera jądro i *initramfs* oraz katalog domowy Grub-a (`/boot/grub`).

W Fedorze:

- Partycja ESP zawiera `grubx64.efi` oraz katalog domowy Grub-a. Może też zawierać jądro.

To drugie rozwiązanie wydaje się wygodniejsze.

Konfiguracja Gruba

- Grub ma *jądro* i *katalog domowy*.
- Jądro jest zapisane w MBR i poza partycjami (partycje MS-DOS) lub w partycji *lagacy BIOS bootloader* (GPT) albo w pliku `grubx64.efi` w partycji ESP.
- Katalog domowy zawiera moduły jądra Gruba, pliki pomocnicze (np. czcionki) i plik konfiguracyjny Gruba.
- Zmienna `$prefix` określa ścieżkę do katalogu domowego Gruba. Jest zapisana w jądrze Gruba.
- Zmienna `$root` określa ścieżkę do katalogu zawierającego jądro systemu i initramfs. Może być automatycznie wyszukana na podstawie UUID (polecenie `search`).
- Grub korzysta z dysków tylko w trybie do odczytu. Wyjątek: plik `grubenv` i polecenie `save_env`.
- Moduły: polecenie `insmod`.
- Polecenie `menuentry`.
- Uruchamianie Linuksa: polecenia `linux` i `initrd`.

Grub jest *themeable*