1. Zawartość dokumentu

Celem dokumentu jest przedstawienie sposobu konfiguracji jądra systemu OpenBSD-3.6 z naciskiem na zmiany wpływające na zwiększenie bezpieczeństwa systemu. Adresowany jest on do administratora systemu, zaś opisywane czynności powinny być wykonywane jako jedne z pierwszych po instalacji oprogramowania.

2. Materiały źródłowe

Wykorzystywane materiały źródłowe:

- Hardening OpenBSD Internet Servers: Building a Custom Kernel, http://geodsoft.com/howto/harden/OpenBSD/kernel.htm

  Dokument ten zawiera dobre wprowadzenie do tematyki hartowania jądra, wiele cennych uwag szczegółowych (jak np. problem z sendmail(8) po wyłączeniu obsługi IPv6). Autor jednak zdecydowanie nie zna opcji, które modyfikuje, co widać choćby na przykładzie `GPL_MATH_EMULATE`, która jest potrzebna jedynie dla procesorów i386.

- OpenBSD FAQ: Building the System from Source, http://www.openbsd.org/faq/faq5.html

  Dokument referencyjny ze względu na pochodzenie od autorów systemu. Polityka twórców stosowania jedynie kernela `GENERIC` budzi duże wątpliwości.

- Dokumentacje systemów OpenBSD, FreeBSD, NetBSD, Linux.

3. Wprowadzenie

Systemy typu UNIX tradycyjnie oparte są na monolitycznym jądrze, które zawiera sterowniki obsługujące sprzęt (taki jak np. dyski twarde, klawiatura) jak również implementuje protokoły sieciowe, systemy plików itp.

System OpenBSD standardowo instalowany jest z jądrem, które ma na celu obsługę jak największej ilości z popularnych urządzeń (np. dyski, karty sieciowe). Zazwyczaj jednak komputer wyposażony jest w jedynie bardzo niewielką część sprzętu obsługiwanego przez system. Powstaje zatem naturalne pytanie, czy nie było by zasadnym wyłączenie wsparcia dla elementów, które w danej konfiguracji sprzętowej nie występują, jakie będą tego koszty (jeżeli chodzi o nakład czasu i pracy) oraz spodziewane zyski.

W przypadku systemu Linux, konfiguracja jądra jest dla nawet średnio zaawansowanych użytkowników podstawową czynnością, do której przystępują po instalacji. Jest to często niezbędne, ponieważ obrazy jądra dostarczane z dystrybucjami często nie wspierają danego typu urządzenia, jakie posiada użytkownik (np. karta graficzna czy dźwiękowa). Sytuacja ta częściowo uległa zmianie dzięki stosowaniu modułowej budowy oraz

modyfikacji, które lepiej przystosowują system do typowych zastosowań (Kernel Rebuilding Guide).

Podobnie w systemie FreeBSD tworzenie własnego dostosowanego jądra jest zalecane przez twórców (FreeBSD Handbook: Why Build a Custom Kernel).

Autorzy OpenBSD przedstawiają jednak zupełnie inne podejście, zdecydowanie odradzając tworzenie własnych kerneli. Uzasadniają to przede wszystkim stwierdzeniem, że korzystanie z własnej, nieprzetestowanej konfiguracji jest w istocie zmniejszeniem bezpieczeństwa, zaś zysk w wydajności i szybkości działania jest niewielki. Wśród powodów, dla których dopuszczają korzystanie z przygotowanego samodzielnie obrazu jądra, jest uruchamianie OpenBSD na maszynie z wyjątkowo małą ilością pamięci RAM (domyślnie sam obraz to plik o rozmiarze ponad 5MB).

Przedstawię poniżej opis przystosowania jądra OpenBSD do pracy na wirtualnej maszynie VMWare Workstation 4.5. Emulowany komputer odpowiada typowemu zastosowaniu jako serwer produkcyjny, gdzie nie ma konieczności korzystania z systemu graficznego, czy karty dźwiękowej. Konfiguracja wirtualnego komputera:

- klawiatura, mysz,
- HDD, CDROM, karta sieciowa,
- 64MB pamięci RAM.

Zakładam więc, że celem nie jest uruchomienie sprzętu, którego nie obsługuję jądro dostarczane standardowo z systemem, a jedynie usunięcie wszelkich niewykorzystywanych elementów celem zwiększenie bezpieczeństwa systemu.

4. Tworzenie jądra systemu według własnej konfiguracji

Budowanie jądra w systemach BSD składa się z trzech etapów:

1. Edycja pliku konfiguracyjnego
   Zakładając, że w systemie zainstalowano źródła jądra oraz narzędzia potrzebne do kompilacji, ustawienia domyślnego kernela znajdują się w pliku
   `/usr/src/sys/arch/i386/conf/GENERIC`
   Korzystając z dowolnego edytora tekstowego można dokonywać zmian poprzez dodawanie i usuwanie wpisów dla poszczególnych urządzeń i opcji, a w przypadku okrajania systemu poprzez poprzedzanie ich komentarzami.
   Dobrze jest dokonywać modyfikacji nie w oryginalnym pliku, lecz w jego kopii, którą można utworzyć poleceniem:
   ```
   cd /usr/src/sys/arch/i386/conf/
   cp GENERIC MYCONF
   ```
2. Skonfigurowanie i przygotowanie do kompilacji
   Po przygotowaniu konfiguracji należy wykonać polecenie
   `config MYCONF`
   Jeżeli w pliku nie ma błędów, to utworzony zostanie katalog
   `/usr/src/sys/arch/i386/compile/MYCONF`
   zawierający wszelkie niezbędne pliki potrzebne do kompilacji nowego jądra. Zaletą takiego rozwiązania (np. w porównaniu do systemu LINUX) jest możliwość

przechowywania wyników pośrednich kompilacji dla różnych konfiguracji sprzętowych (np. różnych maszyn a nawet architektur).

3. Kompilacja i instalacja
   Po przejściu do odpowiedniego katalogu:
   `cd ../MYCONF`
   należy wydać polecenie:
   `make depend all install`
   Aktualny obraz jądra zostanie skopiowany do pliku `/obsd` natomiast nowy zachowany jako `/bsd`.
   Dzięki temu na wypadek, gdyby coś poszło źle istnieje możliwość powrócenia do poprzedniej konfiguracji podając ścieżkę do obrazu na etapie bootowania systemu.

Okrajanie jądra można podzielić na dwa etapy. Pierwszy to usuwanie sterowników urządzeń, które nie występują w systemie, natomiast drugi to usuwanie funkcjonalności realizowanej przez system operacyjny, takiej jak np. implementowane protokoły oraz systemy plików.

5. Modyfikacja listy sterowników

Etap pierwszy – znajdowanie wpisów urządzeń, które faktycznie występują w systemie zabiera stosunkowo najwięcej czasu. Na szczęście możemy skorzystać z narzędzi, które ten proces automatyzują. Dostosowane z systemu NetBSD narzędzie *adjustkernel(1)* automatycznie modyfikuje plik konfiguracyjny na podstawie logu bootowania *dmesg(8)*.
Po zainstalowaniu użycie sprowadza się do wydania komendy:

```
cd /usr/src/sys/arch/i386/conf
adjustkernel -f GENERIC -o ADJGENERIC
```

Plik `ADJGENERIC` zawiera wpisy dla tych urządzeń, które zostały wykryte w fazie inicjalizacji sterowników. Pozostałe opcje są zakomentowane znakami `#(#`.

Efekt działania tego narzędzia jest praktycznie identyczny z ręcznie stworzoną konfiguracją, jednak ta druga wymaga dokładnej wiedzy na temat nie tylko wnętrza komputera oraz wnikliwej analizy *dmesg(8)*, ale i z góry zaplanowanego przeznaczenia maszyny. Różnica sprowadza się do pierwszych linii:

```
#(# option        I386_CPU    # CPU classes; at least one is REQUIRED
option            I486_CPU
option            I586_CPU
option            I686_CPU
option            GPL_MATH_EMULATE  # floating point emulation.
```

gdzie skrypt wyłączył wsparcie dla tylko jednego typu CPU. Bez zmian pozostała także opcja `APERTURE` (dla systemu X Windows) oraz wsparcie dla binarnej kompatybilności z innymi systemami BSD (`COMPAT_*`).

Tak przygotowane jądro po kompilacji ma rozmiar 2.5MB, czyli jest ponad dwukrotnie mniejsze niż instalowane domyślnie o rozmiarze 5.1MB.

6. Modyfikacja modyfikacja funkcjonalności systemu

Etap drugi – okrajanie funkcjonalności systemu z niewykorzystywanych elementów najlepiej rozpocząć od skopiowania ustawień GENERIC niezależnych od architektury do osobnego pliku:

```
cp /usr/src/sys/conf/GENERIC /usr/src/sys/conf/BSS_GLOBAL
```

do którego ścieżka zastąpi wpis z właściwego pliku konfiguracyjnego jądra:

```
#include "../../../conf/GENERIC"
include "../../../conf/BSS_GLOBAL"
```

W tym pliku wpisy dotyczące funkcjonalności, która nie będzie wykorzystywana należy poprzedzić komentarzem. Wątpliwości może budzić usunięcie opcji `DDB`. Ma ona bardzo duże znaczenie (ok. 800kb) dla rozmiaru obrazu kernela. Przy wystąpieniu błędu w kodzie jądra w maszynie serwerowej często nie będziemy mieć dostępu do lokalnej konsoli, gdzie możliwa by była identyfikacja usterki. Tę opcję proponuję usunąć dopiero w ostatnim kroku okrajania jądra, kiedy mamy pewność, że cała reszta systemu działa prawidłowo.

Wątpliwości nie powinno natomiast powodować usunięcie wsparcia dla niewykorzystywanych systemów plików oraz protokołów sieciowych. Według mnie to one stanowią największe potencjalne zagrożenie (ze względu na stopień skomplikowania, a co za tym idzie objętość kodu narażonego na błędu) dla bezpieczeństwa systemu.

Podobnie usunięcie wpisu opcji `LKM` (loadable kernel modules) jest według mnie elementem, który znacząco poprawia bezpieczeństwo systemu. Jako przykład niech posłuży moduł, który zainstalowany przez włamywacza (po uzyskaniu praw superużytkownika) monitoruje ruch TCP/IP.

7. Podsumowanie

Po wprowadzeniu tych dodatkowych zmian (łącznie z usunięciem opcji `DDB`) rozmiar obrazu jądra zmniejszył się do 1.6MB.

Ostatecznie udało sie uzyskać jądro, które uruchamia się nawet przy ograniczeniu ilości pamięci wirtualnej maszyny do 8MB. Przy takich ograniczeniach jądro twórców systemu nawet się nie uruchamiało. W systemie z okrojonym nie zaobserwowano żadnych nieprawidłowości ani anomalii.

Poniższa tabela zawiera porównanie działania systemu z jądrem GENERIC oraz ostatecznie okrojonym BSS. Czas startu systemu był mierzony od włączenia wirtualnej maszyny do pojawienia się komunikatu logowania. Zajętość pamięci po zalogowaniu i wykonaniu komendy

```
sleep 20 && vmstat
```

jako kolumna *memory/free*.

Konfiguracja	*Rozmiar obrazu jądra*	*Czas startu systemu*	*Wolna pamięć*
GENERIC	5.3MB	48 sek	38.8MB
BSS	1.6MB	32 sek	42.8MB
różnica	3.7MB	16 sek	4.0MB

8. Wnioski

Przedstawione w tabeli różnice w działaniu systemów z domyślnym i okrojonym jądrem, chociaż wyraźne, to jednak nie powinny stać się kryterium decydującym o zastosowaniu własnej konfiguracji. Stanowisko autorów OpenBSD jest tutaj znacząco różne od tego, które prezentują twórcy innych systemów i do jakiego ja się skłaniam.

Jako uzasadnienie dla usuwania zbędnej funkcjonalności może posłużyć lista błędów wykrytych w podobnym do OpenBSD systemie FreeBSD (FreeBSD, Security Advisores). Spośród 17 wykrytych w roku 2004 błędów 4 z nich (o oznaczeniach: *shmat*, *ipv6*, *linux*, *procfs*) nie wpływały na system, który byłby odpowiednio okrojony (bez uszczerbku na funkcjonalności dla typowych zastosowań).

9. Bibliografia

1. George Shaffer, Hardening OpenBSD Internet Servers, Building a Custom Kernel; http://geodsoft.com/howto/harden/OpenBSD/kernel.htm
2. OpenBSD FAQ: Building the System from Source; http://www.openbsd.org/faq/faq5.html
3. FreeBSD Handbook: Why Build a Custom Kernel; http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/kernelconfig-custom-kernel.html
4. Kwan Lowe: Kernel Rebuilding Guide; http://www.digitalhermit.com/linux/Kernel-Build-HOWTO.html#WHY-REBUILD
5. VMWare Workstation 4.5; http://www.vmware.com/products/desktop/ws_features.html
6. The NetBSD Packages Collection: *sysutils/adjustkernel;* ftp://ftp.netbsd.org/pub/NetBSD/packages/pkgsrc/sysutils/adjustkernel/README.html
7. OpenBSD: Manual Pages; http://www.openbsd.org/cgi-bin/man.cgi

10.Konfiguracja okrojonego jądra systemu

Poniżej znajdują się pliki konfiguracyjne do budowy okrojonego jądra systemu. Ze względu na objętość usunięte zostały linie puste oraz te poprzedzone komentarzami.

10.1. Konfiguracja sterowników urządzeń, plik *BSS*

```
machine        i386           # architecture, used by config; REQUIRED

include "../../../conf/BSS_GLOBAL"

option         I686_CPU

option         USER_PCICONF   # user-space PCI configuration
option         USER_LDT       # user-settable LDT; see i386_set_ldt(2)

maxusers       32             # estimated number of users

config         bsd    swap generic

mainbus0 at root

cpu0    at mainbus? apid ?
bios0   at mainbus0 apid ?
apm0    at bios0 flags 0x0000 # flags 0x0101 to force protocol version 1.1
pcibios0 at bios0 flags 0x0000       # use 0x30 for a total verbose

isa0    at pcib?
pci*    at mainbus0 bus ?

option         PCIVERBOSE

pchb*   at pci? dev ? function ?      # PCI-Host bridges
ppb*    at pci? dev ? function ?      # PCI-PCI bridges
pci*    at ppb? bus ?
pcib*   at pci? dev ? function ?      # PCI-ISA bridge

npx0    at isa? port 0xf0 irq 13      # math coprocessor
isadma0 at isa?

option  WSDISPLAY_COMPAT_USL          # VT handling
option  WSDISPLAY_COMPAT_RAWKBD               # can get raw scancodes
option  WSDISPLAY_DEFAULTSCREENS=6
option  WSDISPLAY_COMPAT_PCVT         # emulate some ioctls

pckbc0         at isa?                        # PC keyboard controller
pckbd*         at pckbc?              # PC keyboard
pms*           at pckbc?              # PS/2 mouse for wsmouse
pmsi*          at pckbc?              # PS/2 "Intelli"mouse for wsmouse
vga0           at isa?
option         PCIAGP
vga*           at pci? dev ? function ?
wsdisplay*     at vga? console ?
wskbd*         at pckbd? console ?
wsmouse*       at pms? mux 0
wsmouse*       at pmsi? mux 0

pcppi0         at isa?
sysbeep0       at pcppi?

pciide* at pci ? dev ? function ? flags 0x0000

wd*     at pciide? channel ? drive ? flags 0x0000
```

```
le*     at pci? dev ? function ?             # PCnet-PCI based ethernet

pseudo-device  pctr           1
pseudo-device  mtrr           1       # Memory range attributes control
pseudo-device  bio            1       # ioctl multiplexing device

pseudo-device  wsmux          2
pseudo-device  crypto         1
```

10.2. Konfiguracja sterowników urządzeń, plik *BSS*

```
option         KTRACE         # system call tracing, a la ktrace(1)
option         ACCOUNTING     # acct(2) process accounting
option         KMEMSTATS      # collect malloc(9) statistics
option         PTRACE         # ptrace(2) system call

option         CRYPTO         # Cryptographic framework

option         SYSVMSG                # System V-like message queues
option         SYSVSEM                # System V-like semaphores
option         SYSVSHM                # System V-like memory sharing

option         UVM_SWAP_ENCRYPT# support encryption of pages going to swap

option         COMPAT_35      # 3.5,
option         COMPAT_43      # and 4.3BSD

option         FFS            # UFS
option         FFS_SOFTUPDATES        # Soft updates
option         UFS_DIRHASH    # hash large directories
option         QUOTA          # UFS quotas

option         TCP_SACK       # Selective Acknowledgements for TCP
option         TCP_ECN                # Explicit Congestion Notification for TCP
option         TCP_SIGNATURE  # TCP MD5 Signatures, for BGP routing sessions

option         CD9660         # ISO 9660 + Rock Ridge file system
option         FIFO           # FIFOs; RECOMMENDED

option         INET           # IP + ICMP + TCP + UDP
option         INET6          # IPv6 (needs INET)
option         IPSEC          # IPsec

pseudo-device  pf             # packet filter
pseudo-device  pflog          # pf log if
pseudo-device  pfsync         # pf sync if
pseudo-device  enc    1       # option IPSEC needs the encapsulation interface

pseudo-device  pty    16      # initial number of pseudo-terminals
pseudo-device  ksyms  1       # kernel symbols device
pseudo-device  systrace 1     # system call tracing device

pseudo-device  bpfilter       # packet filter
pseudo-device  loop           # network loopback

option         BOOT_CONFIG    # add support for boot -c
```