# Wirtualny *honeypot* w oparciu o kernel UML

Grzegorz Banasiak

Działania związane ze zwiększaniem bezpieczeństwa systemów informatycznych mają przeważnie charakter defensywny. Zakłada się w nich, że przeciwnikowi należy za pomocą wszelkich środków utrudnić dostęp oraz przejęcie kontroli nad chronionym zasobem. Nie sposób się z zasadnością takiego postępowania nie zgodzić, jednakże sztuka wojenna oraz zdrowy rozsądek nakazują równocześnie poznawać wroga, tak aby obrona była jeszcze bardziej skuteczna. A czyż istnieje lepszy sposób na zgłębienie sposobów działania hakerów niż obserwacja ich poczynań na kontrolowanym przez nas systemie, do którego pozwolimy im się włamać?

Niniejszy artykuł stanowi krótki wstęp do tematyki honeypotów, czyli systemów, których jedynym sensem istnienia jest bycie ofiarą ataku. Czytelnik znajdzie tutaj przepis na stworzenie własnego wirtualnego honeypota w oparciu o system Linux oraz inne komponenty publicznie dostępne w sieci.

Architektura planowanej instalacji została schematycznie przedstawiona na Rysunku 1.

Rolę ukrytego nadzorcy (Host OS) jak i honeypota (Guest OS) spełni system z odpowiednio spreparowanym jądrem w wersji 2.4.20. Host OS będzie pracował jako filtrujący pakiety *bridge*, a zatem będzie całkowicie niewidoczny w sieci. Uruchomiony na nim skrypt poprzez odpowiednie reguły netfiltra pozwoli kontrolować ruch wchodzący i wychodzący z honeypota. Wykorzystana zostanie również możliwość przekazywania monitorowanych pakietów do procesu zewnętrznego (w tym przypadku będzie to znany wszytkim snort), aby identyfikować ataki wychodzące lub im zapobiegać. Do stworzenia Guest OS posłuży jądro typu UML (*User Mode Linux*), które może pracować jako zwykły proces użytkownika w systemie macierzystym.

### UML (User Mode Linux)

UML powstał przede wszystkim z myślą o programistach oraz zaawansowanych użytkownikach systemu Linux, którzy w bezpieczny sposób chcieli rozwijać nowe wersje jądra oraz z nimi eksperymentować. Pozwala on wykreować wirtualną maszynę o dowolnych parametrach sprzętowych oraz programowych, z dostępem do wybranych komponentów realnie istniejącego komputera. System plików UML zawiera się w pojedynczym pliku systemu rzeczywistego, więc wszelkie szkody mające miejsce w wirtualnym środowisku nie stanowią żadnego zagrożenia.

UML posiada niezaprzeczalne walory edukacyjne. Jeżeli Czytelnik kiedykolwiek zastanawiał się, co dokładnie wydarzy się po wydaniu polecenia `rm -rf /`, to UML jest właśnie dla niego.

> Kiedy się idzie po miód z balonikiem, to trzeba się starać, żeby pszczoły nie wiedziały, po co się idzie. (...) Z pszczołami nigdy nic nie wiadomo – Kubuś Puchatek

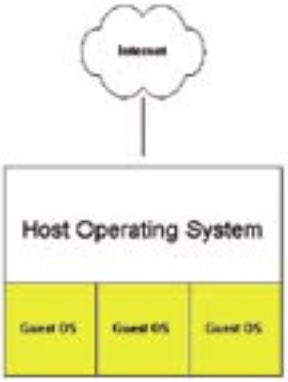
## Cele i środki

Oprócz podanego na wstępie powodu, jakim jest chęć poznania metod i zachowania hakerów w systemie, który udało im się już pokonać, honeypot może być traktowany jako przedłużenie infrastruktury IDS (*Intrusion Detection System*). Z założenia stanowi łatwy łup, więc najprawdopodobniej jako pierwszy stanie się celem ataku, a ponieważ będzie monitorowany, pozwoli uzyskać jasną odpowiedź: kto, kiedy i w jaki sposób się do niego włamał. Wykrycie aktywności wewnątrz honeypota jest gwarancją tego, że chronione zasoby są pod obstrzałem. Omawiane systemy nie cierpią na znany problem fałszywej detekcji (ang. *false positives*), który nieodwołalnie wiąże się z mniej radykalnymi odmianami IDS.

Oczywiście można argumentować, że atak na honeypota świadczy tylko i wyłącznie o tym, że nastąpił atak na honeypota, nie zaś o tym, że chroniona sieć jest celem realnego ataku. Bez wątpienia konstrukcja systemu, który da miarodajne wyniki w optyce IDS, wymaga doświadczenia i pewnej dozy wyczucia.

Honeypot może być również postrzegany jako sposób na odwrócenie uwagi od prawdziwie cennych obiektów. Przeciwnik po wstępnym rozpoznaniu zwróci się w stronę systemu najbardziej rokującego licząc na to, że przejęcie nad nim kontroli otworzy szersze horyzonty penetracyjne. To da nam czas, aby odpowiednio zareagować, informując cho-

*Autor jest studentem Wydziału Elektroniki i Technik Informacyjnych Politechniki Warszawskiej. Od wielu lat interesuje się różnymi aspektami bezpieczeństwa systemów Unix. Obecnie pracuje w Centrum Badawczo-Rozwojowym operatora dominującego. Kontakt: inferno@w.pl.*




**Rysunek 1**. *Architektura systemu typu wirtualny honeypot*

ciażby podmioty odpowiedzialne za infrastrukturę, z której przeprowadzono atak.

Z powyższymi celami wiążą się zróżnicowane wymagania, jakim powinien sprostać planowany system. Inaczej bowiem wyglądać będzie honeypot, który wystawi w sieci pojedynczą podatną na atak usługę (np. BIND w wersji pasującej do dostępnego w Internecie zestawu *exploitów*) – tutaj wystarczy nam zapewne odpowiednio spreparowane *jailowane* środowisko, a inaczej honeypot służący do rzetelnych badań nad sposobami działań włamywaczy. W tym drugim przypadku honeypot powinien być seryjnie instalowanym systemem o znanej historii błędów, bez jakichkolwiek modyfikacji lub z modyfikacjami, które są bardzo trudne do wykrycia. Można posunąć się o krok dalej instalując kilka takich honeypotów w jednej wydzielonej podsieci – tworząc *honeynet*. Twór ten posiada dwie niebagatelne zalety: sprawia wrażenie naturalnego (najlepiej wieloplatformowego) środowiska oraz pozbawiony jest szumu informacyjnego w postaci normalnego ruchu użytkowego.

## Bezpieczeństwo

Niezależnie od stopnia wyrafinowania honeypota, wystawienie go w sieci publicznej zwiększa ryzyko naruszenia bezpieczeństwa – jeśli nie u nas, to u innych użytkowników. Zawsze istnieje szansa na to, że przeciwnik przewyższy nas swymi umiejętnościami i zadrwi z przygotowanej dla niego przynęty. Nie wolno zapominać, że oprogramowanie służące do stworzenia środowiska mniej lub bardziej przez nas kontrolowanego również może zawierać błędy.

Należy dołożyć wszelkich starań, żeby honeypot był możliwie bezpieczny dla otoczenia poprzez:

- alarmowanie obsługi systemu o udanym włamaniu (wykrycie ruchu wychodzącego),
- uniemożliwienie ataków na inne systemy (blokowanie ruchu wychodzącego pasującego do wzorców ataków, reguły antyspoofing, ograniczenie pasma),
- ograniczenie swobody włamywacza (ograniczenie ilości połączeń wychodzących),
- monitorowanie akcji podjętych przez włamywacza niezależnie od wykorzystywanych przez niego technik utajniania transmisji (*key logging* na honeypocie),
- zorganizowanie prostej metody deaktywacji honeypota poprzez jedną szybką czynność.

Wymienione działania powinny być realizowane przez elementy infrastruktury zupełnie niewidoczne dla włamywacza. W wypadku monitorowania, haker nie może być jego świadomy, zaś w przypadku modyfikacji ruchu wychodzącego pasującego do wzorców ataku powinien uznawać, że z niewyjaśnionych powodów takowy się nie powiódł.

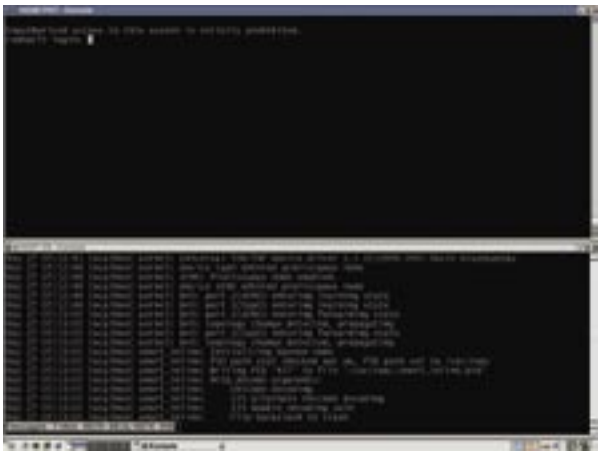
Stosunkowo trudne w realizacji jest tajne nagrywanie sekwencji klawiszy wprowadzanej w trakcie zdalnej sesji. Ponieważ niewskazane jest przechowywanie informacji w samym honeypocie, w przypadku wolno stojącego systemu konieczna jest jakaś forma transmisji sieciowej. Niestety, może ona zostać podsłuchana przez włamywacza i skojarzona z własną aktywnością. Ciekawym pomysłem na rozwiązanie tego problemu jest przesyłanie danych buforowanych przez ukrywający się w systemie moduł w postaci ruchu, który cyklicznie występuje w sieci (np. NetBIOS), a co za tym idzie – nie wyda się podejrzany hakerowi.

Alternatywną koncepcję stanowi zastosowanie wirtualnego honeypota, którego instalację opisano w dalszej części artykułu. W tym przypadku logowaniem klawiszy może zająć się ukryty system nadzorcy udostępniający swoje zasoby systemowi wirtualnemu (Guest OS).

## Tworzymy własny *honeypot*

Lista niezbędnych do instalacji elementów (w nawiasie podano nazwy plików z CD dołączonego do numeru):

- źródła kernela 2.4.20, ulubiony kompilator C (*linux-2.4.20.tar.bz2*),
- iptables (*iptables-1.2.8.tar.gz*),
- *patch* do stworzenia jądra Host OS
- (*host-skas3-bridge-nf-0.0.10-against-2.4.20.diff.gz*),
- pakiet *bridge-utils* (*bridge-utils-0.9.6.tar.gz*),
- *patch* do stworzenia jądra Guest OS/UML (*uml-patch-2.4.20-3.gz*),
- programy pomocnicze do jądra UML (*uml_utilities_20030312.tar.bz2*),
- obraz systemu plików Guest OS ([4], *root_fs.rh-7.2-full.pristine.20020312.bz2*),
- biblioteki *libnet* i *libpcap* (*libnet-1.0.2a.tar.gz*, *libpcap-0.7.2.tar.gz*),
- snort *inline* (*snort_inline-1.9.1-2.tar.gz*),
- skrypt konfigurujący firewalla na Host OS (*rc.firewall*),
- skrypt uruchamiający snorta (*snort_inline.sh*),
- perl5 z modulem Time::HiRes (*Time-HiRes-1.46.tar.gz*).



**Rysunek 2**. *Honeypot po uruchomieniu – emulacja Linuksa w Linuksie*

### Host OS

Jeżeli nie zaznaczono inaczej, podane w artykule instrukcje wykonujemy z prawami administratora. Do stworzenia jądra Host OS należy wykorzystać jądro kompilowane wcześniej na naszym systemie. Będąc w katalogu ze źródłem, instalujemy *patch* i inicjujemy konfigurację:

```
# cd /usr/src/linux-2.4.20

# patch -p1 < sciezka_do/host-skas3-bridge-nf-0.0.10-against-2.4.20.diff
   # make config
```

Wskazany *patch* jest kombinacją dwóch poprawek – pierwszej, pozwalającej uruchamiać jądra UML w trybie SKAS (*Separate Kernel Address Space*), tak aby procesy UML nie miały dostępu do przestrzeni adresowej jądra UML, i drugiej, pozwalającej filtrować pakiety w trybie *bridge*. Do własnej, działającej już konfiguracji jądra dodajemy opcje: `CONFIG_PROC_MM`, `CONFIG_BRIDGE` (ale nie jako moduł), `CONFIG_DEVFS_FS`, `CONFIG_DEVFS_MOUNT`, `CONFIG_BLK_DEV_LOOP`, `CONFIG_TUN` oraz opcje netfiltra `CONFIG_IP_NF_*`. Kompilujemy oraz instalujemy jądro:

```
# make dep && make bzImage && \
make modules && make modules_install
```

Po instalacji obrazu jądra i restarcie systemu przystępujemy do kompilacji elementów pomocniczych:

```
# gzip -cd  bridge-utils-0.9.6.tar.gz | tar xf -
# cd bridge-utils
# ./configure
# make && make install
# cd ..
# bzip2 -cd uml_utilities_20030312.tar.bz2 | tar xf -
# cd tools
# make && make install
```

Jeżeli jądro zostało poprawnie skompilowane, to w tym momencie powinniśmy mieć już możliwość definiowania interfejsów odpowiadających połączonej na poziomie warstwy drugiej sieci:

```
# brctl addbr br0
# brctl show
# brctl delbr br0
```

Mamy też dostęp do narzędzi UML: `uml_mconsole`, `uml_moo`, `uml_net`, itd.

Pozostaje kompilacja oraz instalacja programu snort w wersji *inline* współpracującej z opcją kolejkowania netfiltra. Przed instalacją należy sprawdzić, czy w naszym systemie zainstalowane są biblioteki *libpcap* oraz *libnet*. Jeżeli nie, to należy je doinstalować z odpowiedniej dystrybucji Linuksa lub skorzystać z pakietów źródłowych zamieszczonych na krążku.

```
# gzip -cd snort_inline-1.9.1-2.tar.gz | tar xf -
# cd snort_inline-1.9.1-2
# ./configure --enable-inline
# make && make install
```

Pliki *rules/*.rules* oraz pliki **.config* i *snort_inline.conf* z katalogu *etc* kopiujemy do wybranego katalogu (np. */usr/local/etc/snort_inline*). Następnie dostosowujemy skrypt startowy *snort_inline.sh* zamieszczony na CD, tak aby odzwierciedlał położenie zbiorów w naszym systemie.

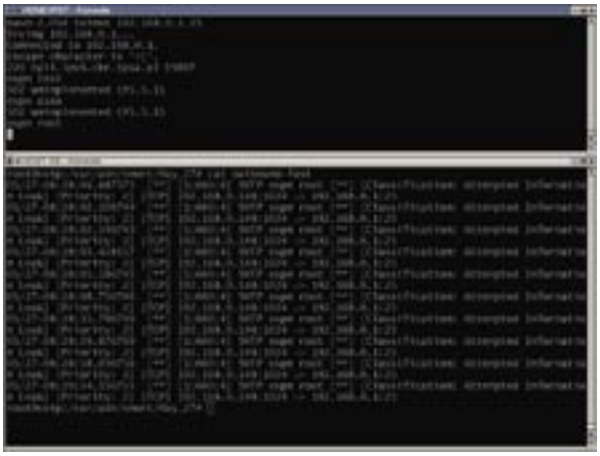
W pliku *snort_inline.conf* dostosowujemy zmienną `RULE_PATH` oraz definiujemy dodatkowe zmienne, które z jakichś powodów pominięto w pakiecie źródłowym:

```
var HOME_NET $HONEYNET
var SMTP_SERVERS $HOME_NET
var TELNET_SERVERS $HOME_NET
var HTTP_SERVERS $HOME_NET
var DNS_SERVERS $HOME_NET
var SQL_SERVERS $HOME_NET
```

Warto również dodać reguły pozwalające blokować (a nie jedynie wykrywać) ataki wychodzące, chociaż pozostają one jeszcze w fazie testowej:

```
include $RULE_PATH/drop.rules
```

Ostatnim elementem po stronie Host OS jest skrypt kontrolujący działanie firewalla. Należy go oczywiście dostosować do naszych potrzeb. Zmienną `PUBLIC_IP` ustalamy na adres IP, pod którym honeypot będzie widoczny w sieci. `INET_IFACE` to interfejs sieciowy Host OS (najczęściej *eth0*), zaś `LAN_IFACE` to interfejs sieciowy, przez który dostępna jest monitorowana podsieć (w naszym przypadku *tap0*). Ze względu na wykorzystanie snort-inline, zmienną `QUEUE` zmieniamy na wartość *yes*. Poniżej znajdziemy zmienne kontrolujące ilość możliwych



**Rysunek 3**. *Przykład blokady ataku wychodzącego – komenda "expn root" pasuje do wzorca wprowadzonego do systemu Snort, więc sesja TCP ulega zerwaniu*

połączeń wychodzących – ich przekroczenie blokuje czasowo wszystkie pakiety transmitowane na zewnątrz (w obrębie danego protokołu). Jeżeli planujemy mieć zdalny dostęp do Host OS (za pomocą dodatkowej karty sieciowej), to wskazana jest również odpowiednia modyfikacja zmiennych w sekcji `VARIABLES FOR MANAGEMENT INTERFACE`.

Skrypt *rc.firewall* jest napisany w ten sposób, że program zewnętrzny (snort-inline) monitorować lub blokować będzie jedynie połączenia wychodzące z honeypota. Nic jednak nie stoi na przeszkodzie, żeby w systemie nadzorcy uruchomić kolejny proces snort, tym razem w trybie standardowym, który będzie identyfikował wzorce ataków przychodzących. Pozostawiam to w gestii Czytelnika.

## Guest OS

Wracamy do oryginalnej postaci jądra 2.4.20 i dodajemy architekturę UML:

```
# gzip -cd linux-2.4.20.tar.gz | tar xf -
# gzip -d uml-patch-2.4.20-3.gz

# cd linux-2.4.20
# patch -p1 < ../uml-patch-2.4.20-3
# make config ARCH=um
```

Najbardziej istotna jest aktywacja opcji `CONFIG_MODE_SKAS`, `CONFIG_HPPFS` (*HoneyPot Proc FileSystem*) oraz `CONFIG_TTY_LOG` (logowanie sesji) i deaktywacja opcji `CONFIG_HOSTFS` odpowiedzialnej za możliwość montowania systemu plików Host OS pod systemem UML. Pozostałe domyślnie ustawione opcje powinny dać nam w pełni sprawny system operacyjny. Na koniec warto w głównym pliku *Makefile* sztucznie zmienić wersję kernela na zgodną z obrazem przechowywanym w systemie plików honeypota (patrz dalej):

```
#KERNELRELEASE=$(VERSION).$(PATCHLEVEL).$(SUBLEVEL)$(EXTR
                        AVERSION)
KERNELRELEASE=2.4.9-31
```

Kompilujemy jądro oraz moduły, o ile takowe zadeklarowaliśmy (domyślnie ich brak):

```
# make dep ARCH=um
# make linux ARCH=um && make modules ARCH=um
```

W efekcie kompilacji otrzymujemy w katalogu głównym źródeł jądra plik wykonywalny o nazwie *linux*, który będzie służył do uruchamiania honeypota. Można go przekopiować do katalogu dostępnego dla użytkowników (np. */usr/local/bin*).

Przechodzimy teraz do przygotowania systemu plików naszej przynęty dla włamywaczy. Można zacząć od przykładowego obrazu mocno okrojonej dystrybucji RedHat umieszczonej na CD (plik *root_fs.rh-7.2-full.pristine.20020312.bz2*). Po rozpakowaniu archiwum w katalogu zwykłego użytkownika (z takimi prawami zostanie uruchomiony honeypot) do pliku o nazwie *root_fs*, możemy go podmontować i zmodyfikować ustawienia systemowe. Tu drobna uwaga – rozkompresowany obraz zajmuje około 711 MB.

```
# mkdir mnt
# mount -o loop root_fs mnt
# cd mnt/etc
```

Postępujemy jak w przypadku konfiguracji typowego systemu Linux. Ustalamy nazwę hosta (*etc/sysconfig/network*), IP, adres bramy, maskę (*etc/sysconfig/network-scripts/ifcfg-eth0*), serwery DNS (*etc/resolv.conf*), lokalnie przechowywane nazwy hostów (*etc/hosts*), ewentualnie informacje o obsługiwanej domenie (*etc/named.conf*, *var/named/**), itd. – tak jakby honeypot był kolejnym elementem zarządzanej przez nas sieci, zaś Host OS w ogóle nie istniał.

Jeżeli w trakcie kompilacji jądra UML zadeklarowano jakiekolwiek moduły, jest to dobry moment, aby przerzucić je do odpowiedniego katalogu w systemie plików honeypo-

### W Sieci

- *http://www.tracking-hackers.com* – strona poświęcona różnym formom honeypotów
- *http://www.honeynet.org* – The Honeynet Project: publikacje, narzędzia, projekty dotyczące technologii honeynetów
- *http://www.honeypots.org* – strona z odsyłaczami na temat honeypotów
- *http://user-mode-linux.sourceforge.net* – The User-mode Linux Kernel Home Page
- *http://www.snort.org* – The Open Source Network Intrusion Detection System
- *http://bridge.sourceforge.net* – Layer 2 Ethernet bridging with Linux

**Listing 1**. *Logi z firewalla (/var/log/syslog)*

```
Apr 21 22:19:44 voip kernel: INBOUND ICMP: IN=br0 OUT=br0 PHYSIN=eth0 PHYSOUT=tap0 SRC=192.168.0.40
   DST=192.168.0.144 LEN=60 TOS=0x00 PREC=0x00 TTL=128 ID=614 PROTO=ICMP TYPE=8 CODE=0 ID=512 SEQ=256
Apr 21 22:19:45 voip kernel: INBOUND ICMP: IN=br0 OUT=br0 PHYSIN=eth0 PHYSOUT=tap0 SRC=192.168.0.40
   DST=192.168.0.144 LEN=60 TOS=0x00 PREC=0x00 TTL=128 ID=616 PROTO=ICMP TYPE=8 CODE=0 ID=512 SEQ=512
Apr 21 22:22:31 voip kernel: OUTBOUND CONN ICMP: IN=br0 OUT=br0 PHYSIN=tap0 PHYSOUT=eth0 SRC=192.168.0.144
   DST=192.168.0.40 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP TYPE=8 CODE=0 ID=35074 SEQ=0
Apr 21 22:22:32 voip kernel: OUTBOUND CONN ICMP: IN=br0 OUT=br0 PHYSIN=tap0 PHYSOUT=eth0 SRC=192.168.0.144
   DST=192.168.0.40 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP TYPE=8 CODE=0 ID=35074 SEQ=256
Apr 21 22:22:33 voip kernel: OUTBOUND CONN ICMP: IN=br0 OUT=br0 PHYSIN=tap0 PHYSOUT=eth0 SRC=192.168.0.144
   DST=192.168.0.40 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP TYPE=8 CODE=0 ID=35074 SEQ=512
Apr 21 22:22:34 voip kernel: OUTBOUND CONN ICMP: IN=br0 OUT=br0 PHYSIN=tap0 PHYSOUT=eth0 SRC=192.168.0.144
   DST=192.168.0.40 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP TYPE=8 CODE=0 ID=35074 SEQ=768
Apr 21 22:22:35 voip kernel: OUTBOUND CONN ICMP: IN=br0 OUT=br0 PHYSIN=tap0 PHYSOUT=eth0 SRC=192.168.0.144
   DST=192.168.0.40 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP TYPE=8 CODE=0 ID=35074 SEQ=1024
Apr 21 22:22:36 voip kernel: Drop icmp after 5 attemptsIN=br0 OUT=br0 PHYSIN=tap0 PHYSOUT=eth0 SRC=192.168.0.144
   DST=192.168.0.40 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP TYPE=8 CODE=0 ID=35074 SEQ=1280
```

Na początku zaobserwowano dwa pakiety ICMP (*Echo Request*) wysłane z hosta 192.168.0.40 na adres 192.168.0.144 (honeypot). Pojawiły się one na interfejsie *bridge* (*br0*) w kierunku od sieci fizycznej (*eth0*) do interfejsu honeypota (*tap0*). 3 minuty później honeypot zaczyna generować zwrotny ruch ICMP, który po 5 pakietach zostaje zablokowany.

ta. W tym celu w katalogu źródłowym kernela wydajemy polecenie:

```
# make modules_install INSTALL_MOD_PATH=sciezka_do_mnt
                       ARCH=um
```

Do pełni szczęścia brakuje jeszcze pliku wymiany:

```
# dd if=/dev/zero of=swap bs=65536 count=1024
# mkswap ./swap
```

Do katalogu z powyższymi dwoma plikami warto przekopiować skrypt o nazwie *honeypot.pl* z pakietu *uml-utilities* (katalog *honeypot* w katalogu źródłowym) wraz z wykorzystywanymi przez niego modułami *(hppfslib.pm, hppfs.pm)*. Pozwala on na modyfikację zawartości katalogu systemowego */proc* honeypota za pomocą plików tworzonych w miejscu, z którego pobierany jest obraz systemu ofiary, tak aby ukryć informacje świadczące o wirtualnym charakterze Guest OS.

Tak naprawdę, uważny obserwator nabierze podejrzeń co do rzeczywistości systemu po zapoznaniu się z logami startowymi lub chociażby nazwami urządzeń pamięci masowej (zamiast znajomego */dev/hda* ujrzymy tutaj */dev/ubd/0*). Można spróbować temu zaradzić poprzez kosmetyczne zmiany w źródle jądra oraz stworzenie odpowiednich urządzeń po stronie systemu UML.

## Finis coronat opus

Czas pozbierać wszystkie elementy układanki w jedną spójną całość. Zmieniamy prawa na zwykłego użytkownika i przechodzimy do katalogu z obrazami plików Guest OS. Uruchamiamy w tle skrypt maskujący *honeypot.pl*, co powinno zaowocować utworzeniem katalogu o nazwie *proc*. Aktywujemy honeypota:

```
$ linux ubd0=root_fs ubd1=swap \
eth0=tuntap,,aa:bb:cc:dd:ee:ff,0.0.0.0 tty_log_fd=3 \
con=pty con0=fd:0,fd:1 3>tty_logfile
```

W tym momencie zabserwujemy coś, co do złudzenia przypomina start systemu RedHat (patrz Rysunek 4) zakończony pojawieniem się znajomego ekranu logowania (login: root, hasło: root). Po stronie Host OS za cały system Guest OS odpowiedzialne są jedynie 4 procesy. W systemie wirtualnym, widzimy oczywiście procesy należące jedynie do honeypota.

Parametry startowe Guest OS wymagają drobnego komentarza:

- *ubdx* – wskazanie plików, które odpowiadają poszczególnym wirtualnym partycjom,
- *eth0* – definicja interfejsu sieciowego o adresie sprzętowym `aa:bb:cc:dd:ee:ff` skojarzonego z urządzeniem *tap0* po stronie Host OS (adres IP nie ma znaczenia, ponieważ pracujemy w trybie *bridge*),
- tty_log_fd – numer deskryptora pliku, w którym zapisane zostaną działania na systemie,
- con=pty – powiązanie konsoli honeypota z pseudoterminalami po stronie Host OS,
- con0 – powiązanie konsoli pierwszej ze standardowym wejściem i wyjściem.

```
# screen /dev/ttyp[0-3]
```

Na tym etapie nie możemy jeszcze skorzystać z dostępu zdalnego, ponieważ Host OS nie pracuje w trybie *bridge*.

Zawartość pliku *tty_logfile* może być odtwarzana z odzwierciedleniem upływu czasu lub bez (przełącznik `-n`) przez kolejne narzędzie z pakietu *uml-utilities* – skrypt w Perlu o nazwie *playlog.pl* (katalog *jail* w katalogu źródłowym):

```
$ perl playlog.pl [-n] sciezka_do/tty_logfile
```

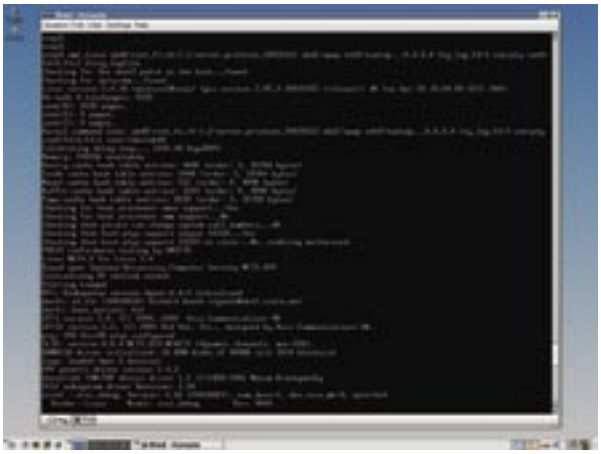
Przy okazji uruchamiania powyższego skryptu może okazać się, że w naszym systemie brakuje modułu o nazwie Time::HiRes. Jego postać źródłową Czytelnik znajdzie na CD (`Time-HiRes-1.46.tar.gz`). Jeżeli plik *tty_logfile* zawierać będzie wiele sesji, to wyświetlone zostaną ich numery identyfikacyjne. Wskazanie konkretnej sesji odbywa się poprzez podanie jej numeru po nazwie pliku logowania.

Zanim zaczniemy na dobre bawić się naszą przynętą warto jeszcze wiedzieć jak szybko wyłączyć honeypota. Może do tego posłużyć program *uml_mconsole* z pakietu *uml-utilities* uruchomiony z poziomu użytkownika, który aktywował Guest OS:

```
$ ls -a ~/.uml
./  ../  qnKAk8/
$ uml_mconsole qnKAk8
(qnKAk8) stop
OK
```

Komenda `go` ponownie przywraca działanie honeypota. Inne przykładowe komendy to `halt` i `reboot`. Zaletą programu *uml_mconsole* jest to, że w żaden sposób nie ujawniamy się w systemie Guest.

Na końcu uruchamiamy z prawami administratora skrypt *rc.firewall* oraz *snort_inline.sh*. W efekcie powinniśmy zaobserwować zdjęcie adresów z interfejsów sieciowych Host OS (o ile nie mamy interfejsu zarządzającego skojarzonego z drugą kartą sieciową) oraz pojawienie się interfejsu *br0* (*bridge*) – teraz honeypot jest pełnoprawnym węzłem naszej sieci. W katalogu */var/log/snort* pojawi się podkatalog o nazwie wywiedzionej z bieżącej daty, w którym możemy spodziewać się logów z ewentualnych ataków wychodzących. Informacje z netfiltra o ruchu wchodzącym i wychodzącym trafiać będą do pliku wynikającego z konfiguracji demona syslogd (najczęściej */var/log/messages* lub */var/log/syslog*). Oczywiście wszelki ruch wychodzący (`OUTBOUND`) świadczy o włamaniu do honeypota. Przykładowe logi Czytelnik znajdzie na Listingu 1.



**Rysunek 4**. *Start honeypota*

## Podsumowanie

Czytelnikowi pozostawiam zebranie czynności aktywacyjnych w jeden wygodny w użytkowaniu skrypt. W dalszej kolejności można pokusić się o uruchomienie kilku wirtualnych honeypotów, tworząc wirtualny *honeynet*. Wymagałoby to drobnych modyfikacji w skrypcie *rc.firewall*, które uwzględniłyby dodatkowe interfejsy *tapn*. ■