IPTables Hacking

CONFidence – 13/05/2006

Jarosław Sajko, PCSS
Jaroslaw.sajko@man.poznan.pl

Zamiast planu

ZB PCSS

- Praca operacyjna w ramach ogólnopolskiej szerokopasmowej sieci PIONIER oraz zasobów Centrum
- Praca badawczo – naukowa w ramach projektów krajowych oraz europejskich (CLUSTERIX, EGEE, UNIZETO, iTVP)
- Usługi komercyjne (audyty, testy penetracyjne, analizy kodów źródłowych i binarnych, szkolenia, etc...)
- Praca „po godzinach” (komunikatory, e-bankowość)

http://security.psnc.pl/

Netfilter (1)

Netfilter to *framework* umożliwiający operowanie na pakietach przesyłanych do systemu, z systemu i poprzez system. Netfilter umożliwia wyniesienie tych operacji poza standardowy kod podsystemu obsługi sieci, jest więc wygodnym narzędziem do implementacji różnego rodzaju filtrów pakietów, bramek NPAT, systemów rozliczania, etc.

http://www.netfilter.org

Netfilter (2)

Koncepcyjnie *Netfilter* opiera się na:

- ***hooks***
  - zdefiniowane w kodzie jądra miejsca gdzie wywoływany jest kod szkieletu *Netfilter* dla pakietów przetwarzanych przez stos sieciowy
  - możliwość rejestrowania w *Netfilter* wywołań, które mają zostać wykonane dla pakietów otrzymanych przez *Netfilter* spełniają kryteria podane przy rejestracji (protokoł oraz miejsce w łańcuchu przetwarzania).
- ***ip_queue*** – sterownik umożliwiający przekazanie pakietu do przetwarzania do przestrzeni użytkownika

IPTables (1)

IPTables to implementacja filtra pakietów z wykorzystaniem *Netfilter*. *IPTables* dostarcza programistom i administratorom wygodniejsze interfejsy, rejestruje się w *Netfilter* w z odpowiednimi parametrami dla filtrowania pakietów, dostarcza standardowych mechanizmów wywoływania reguł oraz zarządzania nimi. W skład pakietu *IPTables* wchodzą również narzędzia przestrzeni użytkownika umożliwiające zarządzanie regułami. *IPTables* umożliwia filtrowanie pakietów dla protokołu IPv4 oraz IPv6.

IPTables (2)

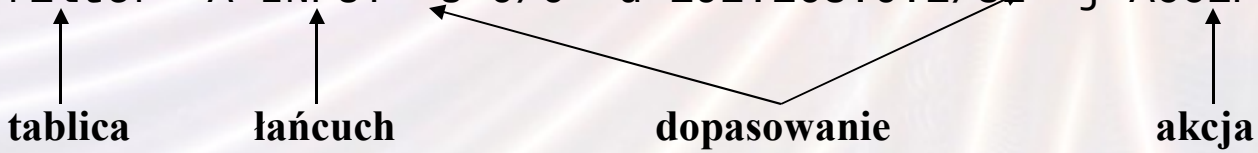
IPTables standardowo umożliwia jedynie proste filtrowanie na podstawie adresów IP oraz wykonywanie prostych akcji jak ACCEPT, DROP, SNAT czy DNAT.

Dzięki temu, że jest rozszerzalne zostało napisane wiele rozszerzeń, które umożliwiają filtrowanie według bardziej wyszukanych kryteriów czy też wykonywanie bardziej zaawansowanych akcji.

IPTables (3)

Przykładowa reguła bez wykorzystania rozszerzeń:

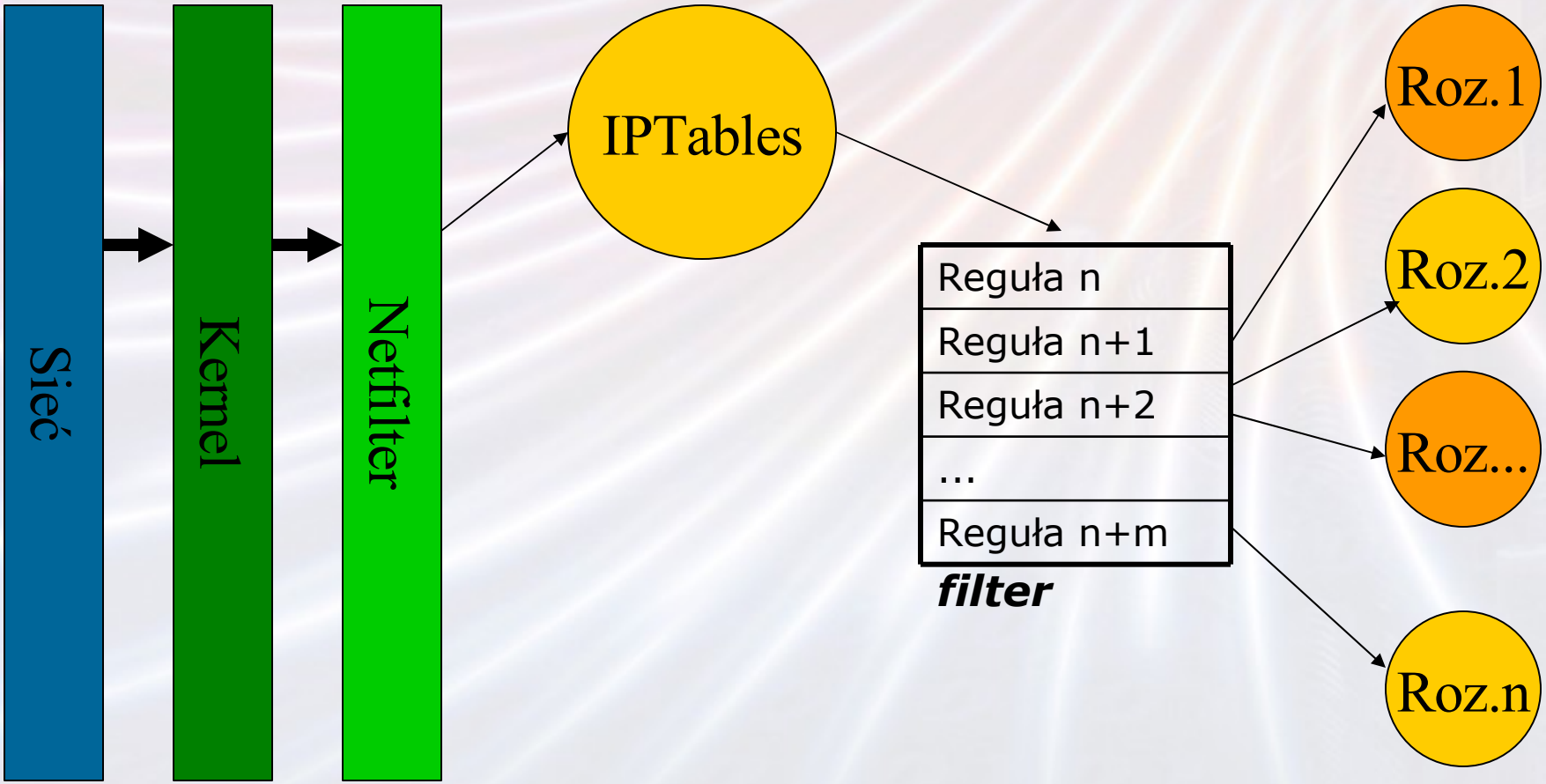
```
iptables –t filter –A INPUT –s 0/0 –d 192.168.0.1/32 –j ACCEPT
```



Przykładowa reguła z wykorzystaniem rozszerzeń:

```
iptables –t filter –A INPUT –s 0/0 –d 192.168.0.1/32 -m ttl --
ttl-eq 123 –j REJECT
```

Netfilter i IPTables



Rozszerzenie (1)

Zasadniczo występują dwa typy rozszerzeń:

- *match* – umożliwia implementacje rozszerzeń dopasowania jak np.: tcp, udp, length, state czy limit
- *target* – umożliwia implementacje rozszerzeń akcji jak np.: REJECT, MASQUERADE, TTL czy LOG

Rozszerzenie (2)

Rozszerzenie, aby było w pełni funkcjonalne powinno składać się z dwóch elementów:

• moduł dla jądra – jest to przede wszystkim standardowym moduł, musi on implementować wymagane przez *IPTables* funkcje, poza tym funkcje te muszą być wielowejściowe.

• moduł dla narzędzia przestrzeni użytkownika – narzędzie, które służy do wprowadzania reguł korzysta z dynamicznie ładowanych rozszerzeń, które obsługują niestandardowe parametry, to jest te parametry, które są specyficzne dla danego rozszerzenia.

Rozszerzenie (3)

Funkcje, które powinny być wywołane przez moduł jądra:

- **ipt_register_match(...)** – jako parametr przyjmuje strukturę **struct ipt_match** i służy do rejestracji rozszerzeń dopasowań,
- **ipt_register_target(...)** – jako parametr przyjmuje strukturę **struct ipt_target** i służy do rejestracji rozszerzeń akcji,
- **ipt_unregister_match(...)** – wyrejestrowuje rozszerzenie dopasowania,
- **ipt_unregister_target(...)** – wyrejestrowuje rozszerzenie akcji.

Rozszerzenie (4)

```
static struct ipt_target nazwa_target = {
        .name = "Nazwa",
        .target = nazwa_target,
        .checkentry = nazwa_checkentry,
        .destroy = nazwa_destroy,
        .me = THIS_MODULE,
};
static struct ipt_match nazwa_match = {
        .name = "Nazwa",
        .match = &nazwa_match,
        .checkentry = &nazwa_checkentry,
        .destroy = &nazwa_destroy,
        .me = THIS_MODULE,
};
```

Rozszerzenie (5)

Analogicznie do rozszerzenia dla jądra implementujemy rozszerzenie dla narzędzia przestrzeni użytkownika. Za pomocą funkcji **_init** rejestrujemy strukturę adekwatną do rozszerzenia (**struct iptables_target** lub **struct iptables_match**). W strukturze tej przekazywane są wskaźniki do funkcji: **help, init, parse, final_check, print, save** oraz wskaźnik do struktury **extra_opts**.

Dwie najważniejsze:

- **parse** – funkcja ta jest wywoływana z parametrami, które nie są standardowe, zwraca „1” w przypadku gdy udało się je przetworzyć
- **final_check** - funkcja ta jest wywoływana na końcu, po przetworzeniu wszystkich parametrów

Prosty przykład

Socket buffer (1)

W pakietach sieciowych poza danymi użytkownika przesyłane są nagłówki protokołów. Każda z warstw poczynając od transportowej i schodząc w dół dodaje swój nagłówek. Za poszczególne warstwy stosu protokołów i poszczególne protokoły odpowiadają różne funkcje.

Zatem aby nie kopiować niepotrzebnie danych została stworzona jedna duża struktura (struct sk_buff), która przechowuje m.in. informacje o nagłówkach wszystkich protokołów.

Dokładny opis w pliku **skbuff.h**

Dane rozszerzeń (1)

Globalne przechowywanie danych:

- Możliwa statyczna inicjalizacja
- Dane powiązane z rozszerzeniem, a nie pojedynczą regułą korzystająca z rozszerzenia

Lokalne przechowywanie danych:

- Dane przechowywane w strukturze powiązanej z regułą, a więc łatwiejsze zarządzanie danymi, które dotyczą pojedynczych reguł

Dane rozszerzeń (2)

Mechanizmy przechowywania danych:

- Przechowywanie danych bezpośrednio w strukturze **userinfo**
- Przechowywanie wskaźników do innych struktur danych
- Wykorzystanie *slab allocator*

Dane rozszerzeń (3)

W systemach SMP dla każdego procesora utrzymywana jest osobna kopia tablicy, a więc pojawia się problem z występowaniem dwóch kopii tych samych danych powiązanych z regułą. Jednym z prostszych rozwiązań tego problemu jest umieszczenie dodatkowego pola w strukturze **targinfo**. Będzie to wskaźnik na kopię główną tej struktury.

(Przykład)

Współbieżność

Funkcje rozszerzeń muszą być wielowejściowe, tj. bezpieczne ze względu na przetwarzanie wielowątkowe. Współbieżny dostęp do danych możemy zrealizować z pomocą:

- operacji atomowych **(przykład)**
- *spin locks* **(przykład)**
- *read-writer spin locks* **(przykład)**

Timers

W wielu przypadkach konieczne jest nieinteraktywne wykonywanie operacji np. „sprzątania”, z pomocą mogą nam przyjść *timery* i API do zarządzania nimi w jądrze.

(przykład)

Kompletny przykład

Narzędzia (1)

Nfsim jest narzędziem umożliwiającym testowanie rozszerzeń w przestrzeni użytkownika, emuluje on środowisko jądra i dostarcza wygodnego interface'u.

```
initialisation done
> iptables -t mangle -A FORWARD -s 192.168.0.2 -d 192.168.1.2 -j IPID
> gen_ip IF=eth0 192.168.0.2 192.168.1.2 0 TCP 1060 80 SYN
rcv:eth0
hook:NF_IP_PRE_ROUTING ip_conntrack NF_ACCEPT {IPv4 192.168.0.2 192.168.1.2 0 6 1060 80 SYN}
hook:NF_IP_PRE_ROUTING iptable_raw NF_ACCEPT {IPv4 192.168.0.2 192.168.1.2 0 6 1060 80 SYN}
hook:NF_IP_PRE_ROUTING ip_conntrack NF_ACCEPT {IPv4 192.168.0.2 192.168.1.2 0 6 1060 80 SYN}
hook:NF_IP_PRE_ROUTING iptable_mangle NF_ACCEPT {IPv4 192.168.0.2 192.168.1.2 0 6 1060 80 SYN}
hook:NF_IP_PRE_ROUTING iptable_nat NF_ACCEPT {IPv4 192.168.0.2 192.168.1.2 0 6 1060 80 SYN}
INFO:IPID: Id changed 0 -> 0
hook:NF_IP_FORWARD iptable_mangle NF_ACCEPT {IPv4 192.168.0.2 192.168.1.2 0 6 1060 80 SYN}
hook:NF_IP_FORWARD iptable_filter NF_ACCEPT {IPv4 192.168.0.2 192.168.1.2 0 6 1060 80 SYN}
hook:NF_IP_POST_ROUTING iptable_mangle NF_ACCEPT {IPv4 192.168.0.2 192.168.1.2 0 6 1060 80 SYN}
hook:NF_IP_POST_ROUTING iptable_nat NF_ACCEPT {IPv4 192.168.0.2 192.168.1.2 0 6 1060 80 SYN}
hook:NF_IP_POST_ROUTING ip_conntrack NF_ACCEPT {IPv4 192.168.0.2 192.168.1.2 0 6 1060 80 SYN}
send:eth1 {IPv4 192.168.0.2 192.168.1.2 0 6 1060 80 SYN}
> gen_ip IF=eth0 192.168.0.2 192.168.1.2 0 TCP 1060 80 SYN
rcv:eth0
hook:NF_IP_PRE_ROUTING ip_conntrack NF_ACCEPT {IPv4 192.168.0.2 192.168.1.2 0 6 1060 80 SYN}
hook:NF_IP_PRE_ROUTING iptable_raw NF_ACCEPT {IPv4 192.168.0.2 192.168.1.2 0 6 1060 80 SYN}
hook:NF_IP_PRE_ROUTING ip_conntrack NF_ACCEPT {IPv4 192.168.0.2 192.168.1.2 0 6 1060 80 SYN}
hook:NF_IP_PRE_ROUTING iptable_mangle NF_ACCEPT {IPv4 192.168.0.2 192.168.1.2 0 6 1060 80 SYN}
hook:NF_IP_PRE_ROUTING iptable_nat NF_ACCEPT {IPv4 192.168.0.2 192.168.1.2 0 6 1060 80 SYN}
INFO:IPID: Id changed 0 -> 1
hook:NF_IP_FORWARD iptable_mangle NF_ACCEPT {IPv4 192.168.0.2 192.168.1.2 0 6 1060 80 SYN}
hook:NF_IP_FORWARD iptable_filter NF_ACCEPT {IPv4 192.168.0.2 192.168.1.2 0 6 1060 80 SYN}
hook:NF_IP_POST_ROUTING iptable_mangle NF_ACCEPT {IPv4 192.168.0.2 192.168.1.2 0 6 1060 80 SYN}
hook:NF_IP_POST_ROUTING iptable_nat NF_ACCEPT {IPv4 192.168.0.2 192.168.1.2 0 6 1060 80 SYN}
hook:NF_IP_POST_ROUTING ip_conntrack NF_ACCEPT {IPv4 192.168.0.2 192.168.1.2 0 6 1060 80 SYN}
send:eth1 {IPv4 192.168.0.2 192.168.1.2 0 6 1060 80 SYN}
>
```

Narzędzia (2)

KDB – Prosty *kernel debugger* wbudowany w jądro, umożliwia wygodny debugging jądra systemu. Projekt prowadzony przez SGI.

```
0xc02fdb00        0        0  1    0   I  0xc02fdcc0 *swapper
EBP        EIP         Function (args)
0xc0373fb8 0xc0100cc1 default_idle+0x41 (0xdffc7f84, 0xc0373f84, 0xc02d6a7f, 0x0
, 0x0)
0xc0373ddc 0xc0113f71 __wake_up_common+0x41 (0xc169e280, 0x1, 0x77, 0x1, 0xc14e1
ed4)
0xc0373e68 0xc0230cfb input_event+0xdb (0xc14e1800, 0x1, 0x77, 0x1, 0xc5)
0xc0373e88 0xc023344e atkbd_report_key+0x3e (0xc14e1800, 0xc0373f84, 0x77, 0x1,
0x55ee7)
0xc0373ec8 0xc023370f atkbd_interrupt+0x23f (0xdf872800, 0x45, 0x0, 0xc0373f84,
0xc030f5a0)
0xc0373eec 0xc020dcbd serio_interrupt+0x3d (0xdf872800, 0x45, 0x0, 0xc0373f84, 0
x9f79b372)
0xc0373f24 0xc020e8e7 i8042_interrupt+0x177 (0x1, 0xc03d1f28, 0xc0373f84, 0x0, 0
x1)
0xc0373f4c 0xc01347e3 handle_IRQ_event+0x33 (0x1, 0x99100, 0xc036c800, 0xc0373f6
c, 0xc0373f84)
0xc0373f74 0xc013487a __do_IRQ+0x5a
0xc0373f7c 0xc01051fc do_IRQ+0x1c (0xc0372000, 0xdffc4000, 0xc0372000, 0x99100,
0xc036c800)
0xc0373fb8 0xc01035e6 common_interrupt+0x1a (0x10808)
           0xc0100d77 cpu_idle+0x67
0xc0373fd8 0xc010024b _stext+0x2b
kdb> _
```

Co powiedziałem

- Pisanie rozszerzeń do *IPTables* jest rzeczą bardzo prostą
- Duża elastyczność *Netfilter/IPTables* daje duże możliwości programistom i administratorom
- Jest kilka rzeczy na które trzeba zwrócić uwagę: sposób przechowywania danych, zarządzanie współbieżnością

Czego nie powiedziałem

- *Conntrack*
- *Helpers*
- *NAT Extensions*
- Własne tablice reguł

Pytania, dyskusja

Dziękuję za uwagę!