Filtrowanie ruchu sieciowego

Mechanizm NetFilter

Konrad Bryłowski
Aleksander Czerwionka
Michał Krause

1. Informacje ogólne

NetFilter to framework w jądrze Linux służący do przechwytywania i zmieniania ruchu sieciowego. Obecnie zalecanym systemem NetFilter jest nftables, który stopniowo wypiera przestarzałe iptables, ip6tables, arptables, itp. Rozwiązanie to jest o wiele bardziej uniwersalne, dostosowywalne i rozbudowane. Stało się ono częścią jądra Linux w 2014 roku.

System nftables pozwala na filtrowanie i kontrolowanie ruchu sieciowego na podstawie zdefiniowanych reguł. Możemy go zastosować w celu stworzenia mechanizmów translacji adresów, firewall, klasyfikacji pakietów oraz do wielu innych.

W odróżnieniu od swojego poprzednika (iptables), w nftables administrator sam tworzy tablice i łańcuchy dopasowując je do chronionej sieci lub urządzenia. Elementami wchodzącymi w składnię nftables są:

- Ruleset (zbiór zasad) - zbiór wszystkich zdefiniowanych reguł, struktur
- Tables (tablice) - służą do organizacji reguł; kontenery na szczególne zastosowania
- Chains (łańcuchy) - są sekwencjami reguł
- Rules (reguły) - definiują akcje dokonywane na ruchu sieciowym oraz warunki wymagane do zrealizowania danej akcji
- Sets (zbiory) - zbiory przechowujące pewne elementy, np. adresy, porty, ciągi znaków, etc.
- Variables (zmienne) - pozwalają na przechowywanie i manipulowanie danymi, które możemy dynamicznie wykorzystywać w regułach, wyrażeniach, itp.

Mechanizm nftables posiada możliwość importowania zbiorów reguł/poleceń z pliku. Służy do tego polecenie:

```
sudo nft -f <file_name>
```

Polecenie to dodaje zawartość pliku lub wykonuje polecenia na obecnym zbiorze reguł (ruleset). Plik może zawierać kolejne polecenia lub strukturę do dodania taką jaką zwraca polecenie `nft list`.

2. Składnia poleceń i reguł nftables

Do wykonania poleceń nftables potrzebujemy uprawnień root. W ich skład wchodzą polecenia dodawania/usuwania elementów, listowania/czyszczenia zawartości struktur. Poniżej znajdują się składnia poleceń nftables:

```
nft {list | flush} {ruleset | table <nazwa> | sets}
nft {add | list | delete | flush} {table | chain} <nazwa>
nft {add | insert} rule <nazwa tablicy/łańcucha> <nazwa> …
nft {add | delete | list | flush} set <tablica> nazwa
nft {add | delete} element <tablica> <zbiór> {element [, …]}
```

Składnia poszczególnych elementów przedstawia się w sposób następujący:

- tablice:
  ```
  table <family> <table_name> {
      …
  }
  np. table inet filters {
      chain forward { … }
      …
  }
  ```
- łańcuchy:
  ```
  chain <table_name> <chain_name> {
      type <chain_type>
      …
  }
  np. chain inet filters forward {
      type filter hook forward priority filter ;
      policy drop ;
  }
  ```
- reguły:
  ```
  <rule_handle> <position> <chain_name> <rule_expression>
  [counter]
  np. inet filters input iifname lo accept
  ```
- wyrażenia:
  ```
  <expression_type> <expression_parameters>
  np. … tcp dport 443 accept
  ```
- zmienne:
  ```
  define <variable_type> <variable_name> = <variable_value>
  np. define ALLOWED_PORTS = { 80, 443 }
  ```

3. Mechanizm SNAT

Source Network Address Translation - translacja źródłowych adresów sieciowych - polega na zmianie adresu źródłowego pakietu na inny określony adres. Najczęściej mechanizm ten jest stosowany na styku sieci o adresacji prywatnej z sieciami publicznymi, aby umożliwić dostęp do Internetu w sieci prywatnej (np. domowe routery).

W przypadku urządzeń o zmiennych adresach publicznych ma miejsce maskarada (ang. masquerade) - zamiast zmiany na konkretny adres zmiana na adres interfejsu, przez który pakiet opuszcza urządzenie.

Pakiet z zewnątrz będzie przekazany do hosta w chronionej sieci tylko jeśli jest odpowiedzią na żądanie tego hosta.

Składnia poleceń pozwalających włączyć SNAT zakładając, że w zmiennej `$PRIVATE_ADDR` jest adres sieci chronionej z maską, `$PUBLIC_ADDR` zawiera adres publiczny urządzenia, a w `$PUBLIC_IF` nazwa interfejsu, przez który urządzenie jest podłączone do sieci publicznej - administrator sam tworzy tablice i łańcuchy:

```
add table nat
add chain nat postrouting { type nat hook postrouting priority
srcnat ; }
add rule nat postrouting ip saddr $PRIVATE_ADDR oif $PUBLIC_IF
snat to $PUBLIC_ADDR
```

Dodanie reguły pozwalającej włączyć SNAT z maskaradą:

```
add rule nat postrouting ip saddr $PRIVATE_ADDR oif $PUBLIC_IF
masquerade
```

Do sprawdzenia nazwy interfesju dostępne są dwa typy dyrektyw - `oif/iif` oraz `oifname/iifname`. Pierwsze są szybsze, ale w przypadku usunięcia a następnie podłączenia interfejsu o danej nazwie reguła może przestać działać, gdyż używa indeksów a nie nazw. Druga opcja porównuje nazwę interfejsu w czasie wykonania, a także umożliwia skorzystanie z wyrażeń typu “enp0s*”.

4. Mechanizm DNAT

Destination Network Address Translation - translacja docelowych adresów sieciowych - polega na zmianie adresu docelowego pakietu na inny należący do chronionej sieci. Najczęściej mechanizm ten jest stosowany do udostępniania usług sieciowych z serwera w sieci prywatnej do Internetu. Pakiety przychodzące z zewnątrz na określone porty są przekierowywane do urządzenia wewnątrz sieci prywatnej. Do jednego portu można przypisać w danym momencie tylko jeden adres docelowy (np. na domyślnym porcie 80 na zewnątrz można udostępnić serwer www

działający tylko pod jednym adresem wewnętrznym, kolejne muszą być dostępne z zewnątrz na innych portach, ale lokalnie mogą działać na domyślnym porcie). Maksymalnie przydzielić można 65535 portów.

Składnia poleceń pozwalających włączyć DNAT zakładając, że w zmiennej `$SERVER_PORTS` jest numer portu albo lista numerów portów do przekierowania (można również podać nazwy usług, do których przydzielone są well-known ports np. `http` zamiast 80, `ssh` zamiast 22), `$SERVER_ADDR` zawiera adres serwera, a w `$PUBLIC_IF` nazwa interfejsu, przez który urządzenie jest podłączone do sieci publicznej - tablica `nat` była utworzona w poprzednim rozdziale:

```
add chain nat prerouting { type nat hook prerouting priority
dstnat ; }
add rule nat prerouting iif $PUBLIC_IF tcp dport $SERVER_PORTS
dnat to $SERVER
```

5. Filtrowanie ruchu sieciowego

Filtrowanie ruchu sieciowego jest realizowane przez ewaluację reguł kolejnych łańcuchów typu `filter` w rosnącej kolejności wartości `priority`.

Reguły mogą decydować o akcji wykonywanej na pakiecie danych w następujące sposoby:

- akceptacja (`accept`) - pakiet jest akceptowany
- odrzucenie (`reject`) - pakiet nie jest akceptowany, informacja zwrotna
- porzucenie (`drop`) - pakiet nie jest akceptowany, brak informacji zwrotnej
- przejście do rozpatrzenia innego łańcucha (`jump` - po zakończeniu powrót do obecnego/`goto` - bez powrotu)
- kontynuowanie (`continue`) - kontynuacja ewaluacji reguł

Decydując o pakiecie mamy dostęp do takich informacji jak dane adresata/nadawcy, port źródłowy, port docelowy, protokół komunikacyjny, dopasowanie do zdefiniowanych przez administratora zasad.

Oprócz wykonywania funkcji filtrowania ruchu sieciowego, możemy również zapisać filtrowany ruch oraz wykonywane akcje do dziennika. Wystarczy definiując reguły zastosować wyrażenie `log` oraz ewentualnie podać `prefix` do dodania przed zapisywaną wartością. Dzienniki są domyślnie zapisywane w katalogu dzienników systemowych.

Istnieje również mechanizm zliczający dopasowań bajtów i pakietów do określonych reguł. Aby skorzystać z takiej funkcji, należy zastosować wyrażenie `counter`.

Przykładowe łańcuchy, które odpowiadają za blokowanie ruchu przychodzącego z wyjątkiem odpowiedzi, pingów i loopbacku, dopuszczanie ruchu wychodzącego z wyjątkiem określonego adresu docelowego wraz ze zliczaniem prób połączenia z nim oraz blokowanie ruchu przekierowywanego z wyjątkiem określonych portów, odpowiedzi na żądania oraz pingów, na dozwolonych portach również blokowane są połączenia z określonym adresem.

```
chain ports {

      tcp dport { 53, 80, 443 } accept

      tcp sport { 53, 80, 443 } accept

      udp sport { 53, 80, 443 } accept

      udp dport { 53, 80, 443 } accept

    }

chain banned_addr {

      ip daddr 212.77.98.9 counter reject with
           icmp port-unreachable

      continue

    }

chain forward {

      type filter hook forward priority filter; policy drop;

      jump banned_addr

      ct state invalid drop

      ct state established,related accept

      meta l4proto icmp accept

      jump ports

    }

chain input {

      type filter hook input priority filter; policy drop;

      iifname "lo" accept

      ct state invalid drop
```

```
        ct state established,related accept
        meta l4proto icmp accept
    }
chain output {
        type filter hook output priority filter; policy accept;
        oifname "lo" accept
        jump banned_addr
        meta l4proto icmp accept
    }
```

6. Port knocking

Port knocking jest techniką wykorzystywaną do zabezpieczenia dostępu do określonych przez nas usług. Jest ona często wykorzystywana do chronienia dostępu zdalnego do urządzenia.

Mechanizm ten opiera się na wprowadzeniu wymogu wykonania pewnej kombinacji prób połączeń do urządzenia. Rozpatrywana jest ilość połączeń, łączenie do odpowiednich portów, czas pomiędzy próbami, kolejność prób. Po wykonaniu wymaganej sekwencji dany adres dostaje się na listę dozwolonych do połączenia na chroniony port (często również przez określony czas).

W nftables port knocking możemy zaimplementować wykorzystując łańcuchy typu `filter` oraz zbiorów lub zmiennych. Nadawca na określony czas dodawany jest do zbioru przy pierwszym połączeniu, a następnie musi wykonać połączenie do kolejnego portu w sekwencji w danym czasie. Przy ostatnim elemencie sekwencji nadawca jest dodawany do kolejnego zbioru zaakceptowanych użytkowników i dopuszczony jest do połączenia się na chroniony port.

Przykładowy łańcuch dopuszczający połączenie ssh po sekwencji portów 123, 234, 345.

```
set clients {
        type ipv4_addr
        flags dynamic,timeout
    }
set candidates {
        type ipv4_addr . inet_service
```

```
        flags dynamic,timeout

    }

chain knock_chain {

        iifname "lo" return

        tcp dport 123 add @candidates { ip saddr . 234 timeout
5s }

        tcp dport 234 ip saddr . tcp dport @candidates add
@candidates { ip saddr . 345 timeout 5s }

        tcp dport 345 ip saddr . tcp dport @candidates add
@clients { ip saddr timeout 20s } log prefix "Succesful knock:
"

        ip saddr .

        tcp dport 22 counter ip saddr @clients accept

        tcp dport 22 ct state established,related accept

        tcp dport 22 counter log prefix "Someone tried to reach
ssh" reject with tcp reset

    }
```

Powyższy przykład jest tylko prostą implementacją tego mechanizmu, trzykrotne skanowanie wszystkich portów otworzyłoby ssh. Aby się przed tym zabezpieczyć należałoby usuwać dany adres ze zbioru `candidates` w przypadku próby połączenia na inny port. Wymagałoby to stworzenia osobnych zbiorów dla każdego kroku w sekwencji i usuwania z nich danego adresu przy każdej próbie połączenia.