

CONNTRACK – monitorowanie połączeń w kernelach ≥ 2.4 .

Bardzo wielu z nas zaczyna poznawanie Linuxa od konieczności rozdzielania Internetu na n-komputerów. Wszędzie w Internecie znajdujemy informacje typu "zrób to na Linuxie, na Linuxie będzie taniej, po co Ci router - postaw Debiana". Instalacje takie zazwyczaj kończą się zainstalowaniem najprostszej wersji Linuxa, konfiguracją NAT i zamknięciem tematu. W tekście tym postaram się przedstawić względnie dokładnie jak to naprawdę działa i jak Linux monitoruje ruch sieciowy za pomocą conntrack.

statefull vs stateless

W starszych wersjach jądra Linuxa pozwalały na filtrowanie pakietów za pomocą ipchains. Było to proste narzędzie, które pozwalało na filtrowanie pakietów na podstawie ich określonych cech, jednak bez możliwości odwoływania się do całych strumieni pakietów. Jeśli jakiś pakiet, np. kończący połączenie, wpadał do naszego serwera, nie byliśmy w stanie określić czy wcześniej takie połączenie w ogóle istniało i czy pakiet ten w ogóle powinien się tu znaleźć. Jest to chyba najważniejsza cecha różniąca ipchains od iptables, które stało się dostępne w kernelach ≥ 2.4 . Stateful inspection, czyli możliwość kontrolowania całych połączeń znacznie poprawiło możliwości wykorzystania Linuxa jako firewalla czy prostego filtru pakietów, jednak pociąga ono za sobą znaczny wzrost skomplikowania.

Przygotowanie kernela

Kontrolą stanu połączeń i śledzeniem pakietów zajmuje się **state machine**, często nazywana **connection tracking machine**. Jest ona dostępna po wkompiłowaniu w jądro następujących opcji:

CONFIG_NETFILTER – włącza obsługę netfilter w Linuxie. To pozwala wykorzystać Linuxa jako firewall czy filtr umożliwiający filtrowanie i zmianę w locie pakietów.

CONFIG_IP_NF_IPTABLES – pozwala na skorzystanie z dobrodziejstw netfilter za pomocą iptables.

CONFIG_IP_NF_CONNTRACK – ta opcja interesuje nas najbardziej: właśnie ona włącza monitorowanie połączeń poprzez **conntrack machine**.

CONFIG_IP_NF_MATCH_STATE: - włączenie tej opcji pozwala na filtrowanie przy pomocy iptables połączeń ze względu na ich stan, który jest oczywiście kontrolowany przez conntrack machine (więcej o tym w dalszej części tekstu); jest ona ograniczoną wersją opcji następnej.

CONFIG_IP_NF_MATCH_CONNTRACK – pozwala na jeszcze bardziej dokładne filtrowanie pakietów niż poprzednia opcja.

Dodatkowo w kernelu znajdują się inne opcje związane z CONNTRACK, które umożliwiają śledzenie protokołów warstw wyższych, która to czynność już nie jest taka prosta. W standardowym kernelu dostępne są: CONFIG_IP_NF_FTP, CONFIG_IP_NF_IRC, CONFIG_IP_NF_TFTP, CONFIG_IP_NF_AMANDA. Dodatkowo w POM dostępne są inne dodatkowe rozszerzenia iptables, które mogą być wykorzystane.

Connection tracking dokonuje się w dwóch łańcuchach iptables: PREROUTING i OUTPUT. Wszystkie połączenia poza lokalnie generowanymi obsługiwane są przez łańcuch PREROUTING, pozostałe przez OUTPUT. Jeśli z naszego hosta wysyłany jest pakiet, łańcuch OUTPUT dba o to żeby odpowiednia informacja znalazła się w conntrack. Powracający pakiet obsługiwany jest już przez łańcuch PREROUTING. Wszelkie pakiety, które nie są generowane lokalnie przechodzą najpierw przez łańcuch PREROUTING, który zapewnia znalezienie się ich w conntrack.

Ważną rzeczą, o której należy wspomnieć jest defragmentacja pakietów – po to żeby conntrack mógł działać poprawnie musi on zdefragmentować wszelkie nadchodzące dane. W

poprzednich kernelach można było włączać i wyłączać defragmentację, kernele z serii powyżej 2.4 zostały jej pozbawione ze względu na conntrack. Jeśli connection tracking ma miejsce, defragmentacja jest dokonywana automatycznie.

Informacja o połączeniach jest przechowywana w strukturze nazwanej conntrack. Co ważne, jest ona także dostępna dla użytkowników przez **/proc/net/ip_conntrack**. Informacja o sesjach przechowywana jest w kernelu wraz z danymi niezbędnymi do zidentyfikowania danego połączenia.

Przykładowa zawartość ip_conntrack:

```
tcp      6 56 SYN_SENT src=195.149.118.7 dst=209.221.29.200 sport=2364 dport=445
[UNREPLIED] src=209.221.29.200 dst=195.149.118.7 sport=445 dport=2364 use=1
mark=0
tcp      6 425809 ESTABLISHED src=195.149.118.164 dst=217.165.9.42 sport=3224
dport=23038 [UNREPLIED] src=217.165.9.42 dst=83.16.37.46 sport=23038 dport=3224
use=1 mark=0
tcp      6 208878 ESTABLISHED src=195.149.118.164 dst=213.234.118.231 sport=1392
dport=6881 src=213.234.118.231 dst=83.16.37.46 sport=6881 dport=1392 [ASSURED]
use=1 mark=0
udp      17 29 src=195.149.118.2 dst=192.215.6.16 sport=34932 dport=53
src=192.215.6.16 dst=195.149.118.2 sport=53 dport=34932 use=1 mark=0
udp      17 47 src=195.149.118.164 dst=83.213.155.220 sport=12078 dport=7738
src=83.213.155.220 dst=83.16.37.46 sport=7738 dport=12078 [ASSURED] use=1 mark=0
tcp      6 354958 ESTABLISHED src=195.149.118.164 dst=83.27.168.72 sport=2812
dport=16589 [UNREPLIED] src=83.27.168.72 dst=83.16.37.46 sport=16589 dport=2812
use=1 mark=0
tcp      6 164820 ESTABLISHED src=195.149.118.164 dst=83.25.233.97 sport=4070
dport=6881 [UNREPLIED] src=83.25.233.97 dst=83.16.37.46 sport=6881 dport=4070
use=1 mark=0
tcp      6 59 TIME_WAIT src=195.149.118.19 dst=66.55.134.226 sport=2532 dport=80
src=66.55.134.226 dst=195.149.118.19 sport=80 dport=2532 [ASSURED] use=1 mark=0
```

Jak widać, poza standardowymi parametrami takimi jak adresy i porty oraz określenie protokołu transportowego, przechowywane są jeszcze dodatkowe dane.

Wszystkie połączenia, które trafiają do kernela mogą zostać zaklasyfikowane do czterech różnych grup:

- NEW - połączenie oznaczone jako NEW jest w zasadzie jedynie informacją, że pojawił się pakiet, który wcześniej był nieodnotowany poprzez conntrack. Niekoniecznie są to pierwsze pakiety zestawiające połączenie, jak pakiet z flagą SYN w TCP. Jeśli pojawi się pakiet, o którym wcześniej kernel nic nie wiedział, zostanie on uznany jako NEW.
- ESTABLISHED - połączenia tego typu oznaczają już potwierdzoną wymianę danych w obydwu kierunkach. Wystarczy, że na wysłany pakiet otrzymana zostanie odpowiedź i dane połączenie otrzymuje status ESTABLISHED. Dotyczy to także pakietów UDP czy ICMP, choć są to protokoły bezpołączeniowe.
- RELATED - połączenia spokrewnione są dokładnie tym na co wskazuje nazwa. Często obok połączeń w stanie ESTABLISHED pojawiają się dodatkowe pakiety, które nie biorą bezpośredniego udziału w komunikacji, ale są jakby dodatkowymi informacjami. Rozpoznanie tych pakietów jest możliwe właśnie przez oznaczenie ich jako RELATED. Dobrym przykładem może być tutaj połączenie ftp, gdzie podstawowa komunikacja odbywa się poprzez port 21 a dane są przesyłane poprzez port 20. To wprowadza dodatkowe problemy ze śledzeniem tych połączeń, które zostaną omówione w dalszej części tekstu.
- INVALID - pakiety, które nie mogą być zidentyfikowane lub powiązane z innymi połączeniami. Powodów takiego zachowania może być wiele, a jeden z nich to problem z

wydajnością systemu czy np. pakiety ICMP, które powracają z informacją na temat połączeń, które nie wychodziły z naszego systemu.

Moduł *state* iptables pozwala na dopasowywanie pakietów według tych właśnie stanów. Dla przykładu, jeśli chcemy wpuszczać w łańcuchu INPUT jedynie połączenia już istniejące lub związane z już istniejącymi:

```
iptables -A INPUT -m state --state RELATED,ESTABLISHED
```

O ile moduł *state* (CONFIG_IP_NF_MATCH_STATE) pozwala na dopasowywanie połączeń jedynie na podstawie ich stanu, to moduł *conntrack* (CONFIG_IP_NF_MATCH_CONNTRACK) ma już znacznie więcej możliwości. Pełną informację o możliwościach dopasowywania obydwu modułów uzyskamy poprzez wykonanie poleceń:

```
iptables -m state -h
iptables -m conntrack -h
```

Conntrack na dywaniku

Zobaczmy jakie wpisy pojawiają się w conntrack przy przechodzeniu przez nasz serwer/router pakietów różnego typu. Dzięki temu będziemy w stanie zrozumieć w jaki sposób działa conntrack, a przy okazji będzie to powtórka z podstaw sieci TCP/IP:)

ICMP

Najprostszym przypadkiem będzie prawdopodobnie sprawdzenie jak zachowuje się conntrack kiedy będą się w nim pojawiać pakiety ICMP.

Po wysłaniu zwyczajnego "pinga" czyli pakietu icmp typu echo-request do adresu, który bądź nie odpowiada celowo, bądź nie istnieje, w tablicy conntrack pokazuje się natychmiast wpis podobny do poniższego:

```
icmp      1 18 src=192.168.1.11      dst=195.149.118.255 type=8 code=0 id=64841 [UNREPLIED]
          src=195.149.118.255 dst=192.168.1.11      type=0 code=0 id=64841 use=1
```

Widzimy kolejno typ protokołu¹, jego numer według oznaczeń IANA (można je sprawdzić w /etc/protocols lub bezpośrednio u źródła: <http://www.iana.org/assignments/protocol-numbers>). Następnie pojawia się czas po jakim dany wpis zniknie z conntrack. Wartości te są określone przy kompilacji kernela w pliku:

```
/usr/src/linux/net/ipv4/netfilter/ip_conntrack_proto_icmp.c
unsigned long ip_ct_icmp_timeout = 30*HZ;
```

Daje to domyślnie 30 sekund, po których wpis z conntrack jest usuwany.

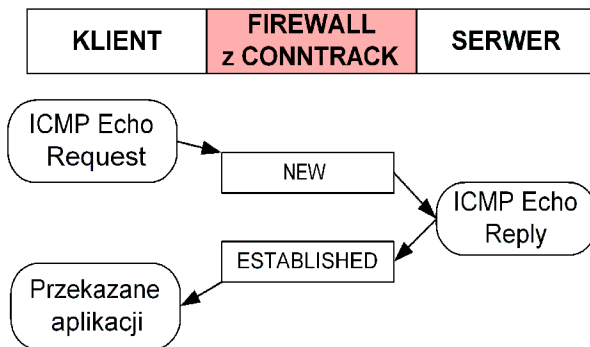
Wpis w tablicy conntrack pokazuje adres źródłowy (src) i docelowy (dst) połączenia oraz informację o typie pakietu ICMP (8 – echo request). Zaznaczona została także informacja, że do tego pakietu nie przyszła żadna odpowiedź ([UNREPLIED]), dopasowana do danych opisanych dalej. Pole "code" w przypadku pakietu icmp typu echo-request i echo replay jest zawsze równe zero. Pole "id" pozwala na jednoznaczne zidentyfikowanie wysyłanych i otrzymywanych pakietów icmp, aby można było rozpoznawać dane "połączenia" icmp. Pole "use" jest potrzebne jądro systemu, aby oznaczać ile innych obiektów zawartych w jądrze odwołuje się dodatkowo do danego połączenia. Najczęściej pole "use" ma wartość 1.

1 Typy pakietów icmp oraz ich kody: <http://iptables-tutorial.frozentux.net/iptables-tutorial.html#ICMPTYPES>

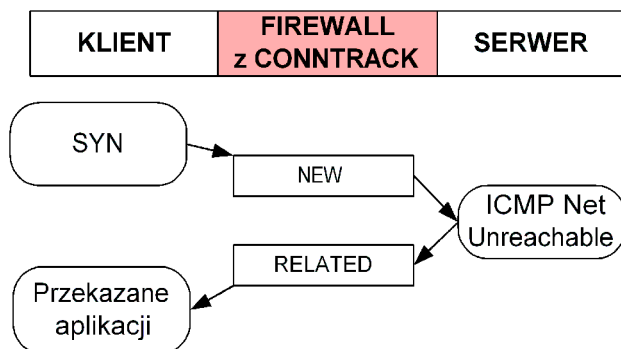
Przez conntrack rozpoznawany jest określony podzbiór typów pakietów icmp (określone również w /usr/src/linux/net/ipv4/netfilter/ip_conntrack_proto_icmp.c):

0	Echo Reply	[RFC792]
8	Echo	[RFC792]
13	Timestamp	[RFC792]
14	Timestamp Reply	[RFC792]
15	Information Request	[RFC792]
16	Information Reply	[RFC792]
17	Address Mask Request	[RFC950]
18	Address Mask Reply	[RFC950]

I tylko te typy pakietów ICMP będzie można później klasyfikować za pomocą iptables jako NEW i ESTABLISHED. Ponieważ działają one na zasadzie zapytania i odpowiedzi, są klasyfikowane przez conntrack jako "połączenia".



Pozostałe typy pakietów icmp mogą być wyłapywane jako pakiety RELATED, czyli powiązane z innymi połączeniami.



Poniżej widać wpis conntrack, w którym brakuje oznaczenia [UNREPLIED], czyli przyszła odpowiedź na naszego pinga.

```
icmp 1 29 src=192.168.1.11 dst=195.149.118.2 type=8 code=0 id=6660
      src=195.149.118.2 dst=192.168.1.11 type=0 code=0 id=6660 use=1
```

UDP

UDP jest protokołem bezpołączeniowym, przy którym nie można mówić jednoznacznie o ustanowionej sesji, jak w przypadku TCP, gdzie cała wymiana jest zapoczątkowana stworzeniem połączenia poprzez wymianę sławnej trójki (SYN, SYN-ACK, ACK). UDP ma raczej charakter wysyłania pojedynczych informacji w określonym kierunku. Zobaczmy jak sobie z tym radzi conntrack.

```
udp 17 28 src=192.168.1.11 dst=195.149.118.2 sport=44765 dport=33444 [UNREPLIED]
      src=195.149.118.2 dst=192.168.1.11 sport=33444 dport=44765 use=1
```

Powyższy wpis to wynik działania traceroute, który korzysta z UDP i portu docelowego 33444. Jak widać informacja jest bardzo zbliżona do tej, która jest umieszczona w przypadku ICMP.

Domyślny czas wymazania informacji z conntrack, podobnie jak w ICMP, określony jest w pliku źródeł jądra (/usr/src/linux/net/netfilter/ip_conntrack_proto_udp.c):

```
unsigned long ip_ct_udp_timeout = 30*HZ;
```

Dodatkowo pojawia się czas określający wygasanie informacji o strumieniu UDP wynoszące 180 sekund.

```
unsigned long ip_ct_udp_timeout_stream = 180*HZ;
```

Strumień jest rozumiany jako ciąg pakietów wymienionych pomiędzy dwoma adresami i portami. Widać to doskonale w poniższym fragmencie wyżej wspomnianego pliku:

```
#ip_conntrack_proto_udp.c

/* If we've seen traffic both ways, this is some kind of UDP
   stream. Extend timeout. */
if (test_bit(IPS_SEEN_REPLY_BIT, &conntrack->status)) {
    ip_ct_refresh(conntrack, ip_ct_udp_timeout_stream);
    /* Also, more likely to be important, and not a probe */
    set_bit(IPS_ASSURED_BIT, &conntrack->status);
}
...
```

Poniżej conntrack odnotował wysłanie pakietu UDP na port 53, czyli zapytanie do DNS. Czas wygaśnięcia wpisu ustawiony został na wartość 29, jako jednorazowa wymiana.

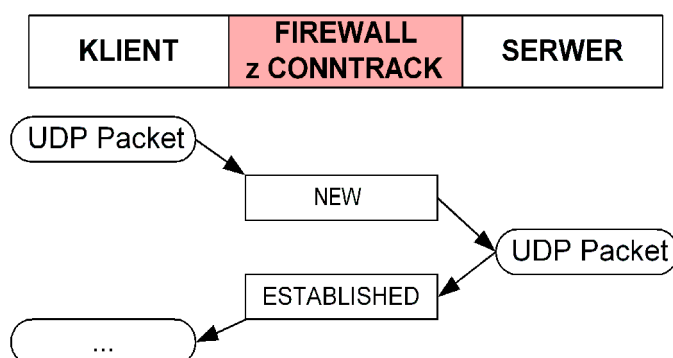
```
udp      29 7  src=192.168.1.11 dst=157.25.5.18 sport=33995 dport=53
         src=157.25.5.18 dst=192.168.1.11 sport=53 dport=33995 use=1
```

Kiedy pojawiają się kolejne pakiety pomiędzy tymi dwoma lokalizacjami, połączenie zamieniane jest w strumień ([ASSURED]) i czas wygaśnięcia tej informacji wydłużany jest do 180 sekund.

```
udp      17 170 src=192.168.1.11 dst=157.25.5.18 sport=33995 dport=53
         src=157.25.5.18 dst=192.168.1.11 sport=53 dport=33995 [ASSURED] use=1
```

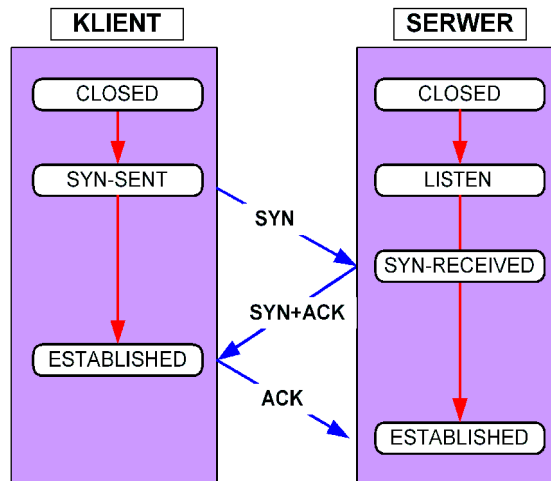
Warto przy tym odnotować, że system będzie preferował połączenia oznaczone jako ASSURED od tych, które są oznaczone jako UNREPLIED w przypadku przeciążenia systemu.

Przy dopasowywaniu połączeń UDP przez iptables można korzystać jedynie ze stanów NEW i ESTABLISHED.



TCP

Ponieważ TCP jest protokołem połączeniowym, który inicjowany jest za pomocą znanej wszystkim kombinacji SYN, SYN+ACK, ACK, contrack ma więcej pracy przy monitorowaniu tych połączeń.



W contrack TCP może przybierać znacznie więcej stanów niż ICMP czy UDP:

```
static const char *tcp_contrack_names[] = {
    "NONE",
    "ESTABLISHED",
    "SYN_SENT",
    "SYN_RECV",
    "FIN_WAIT",
    "TIME_WAIT",
    "CLOSE",
    "CLOSE_WAIT",
    "LAST_ACK",
    "LISTEN"
};
```

Część z tych stanów jest wykorzystywana przez contrack jedynie wewnętrznie. Ponieważ połączenia typu TCP są bardziej złożone niż ICMP i UDP, znacznie więcej jest także timeoutów związanych z różnymi stanami.

```
unsigned long ip_ct_tcp_timeout_syn_sent = 2 MINS;
unsigned long ip_ct_tcp_timeout_syn_recv = 60 SECS;
unsigned long ip_ct_tcp_timeout_established = 5 DAYS; <- !!!
unsigned long ip_ct_tcp_timeout_fin_wait = 2 MINS;
unsigned long ip_ct_tcp_timeout_close_wait = 60 SECS;
unsigned long ip_ct_tcp_timeout_last_ack = 30 SECS;
unsigned long ip_ct_tcp_timeout_time_wait = 2 MINS;
unsigned long ip_ct_tcp_timeout_close = 10 SECS;
```

Podczas ustanawiania połączenia² wysyłany jest pakiet TCP z ustawioną flagą SYN. Contrack automatycznie wpisuje do swoich tablic wpis oznaczony SYN_SENT z dodatkowym opisem UNREPLIED.

```
tcp      6 119 SYN_SENT src=192.168.1.11 dst=192.168.1.111 sport=46122 dport=25 [UNREPLIED]
          src=192.168.1.111 dst=192.168.1.11 sport=25 dport=46122 use=1
```

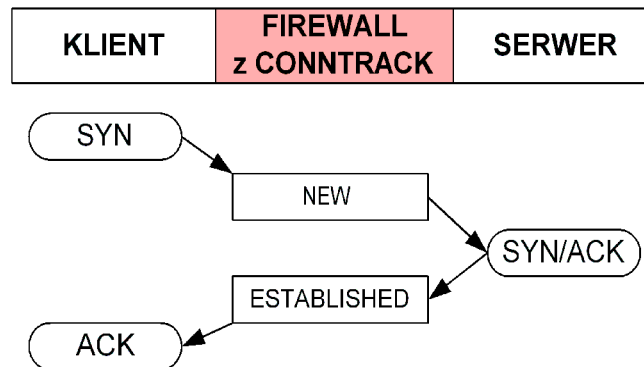
Jeśli połączenie zostanie zaakceptowane, odesłany zostanie pakiet TCP z flagami SYN i ACK, co w naszej tabeli zostanie odzwierciedlone jako SYN_RECV.

² Dokładny opis procedury ustanawiania połączenia TCP jest opisany w RFC793, rozdział 3.4.

```
tcp      6 101 SYN_RECV src=192.168.1.11 dst=192.168.1.111 sport=46122 dport=25
          src=192.168.1.111 dst=192.168.1.11 sport=25    dport=46122 use=1
```

Do pełnego ustanowienia połączenia potrzebne jest uzyskanie potwierdzenia przez pakiet z flagą ACK. Jeśli pozostałe dane takie jak porty i adresy połączenia się zgadzają, wpis w conntrack zamieniany jest na ESTABLISHED i ASSURED (co podobnie jak w przypadku UDP powoduje, że połączenia te są utrzymywane przy zbyt wysokim load systemu). Zmienia się także czas wygasania takiego połączenia. Domyślnie jest to 5 dni (432000 sekund). Skąd tak duża wartość, trudno powiedzieć. Warto poeksperymentować z tą wartością i próbować ją skrócić, co może pomóc przy wypełniającej się tablicy conntrack kiedy nie można zwiększyć samej tablicy z powodu braku zasobów.

```
tcp      6 431900 ESTABLISHED src=192.168.1.11 dst=192.168.1.111 sport=46122 dport=25
          src=192.168.1.111 dst=192.168.1.11 sport=25    dport=46122 [ASSURED] use=1
```



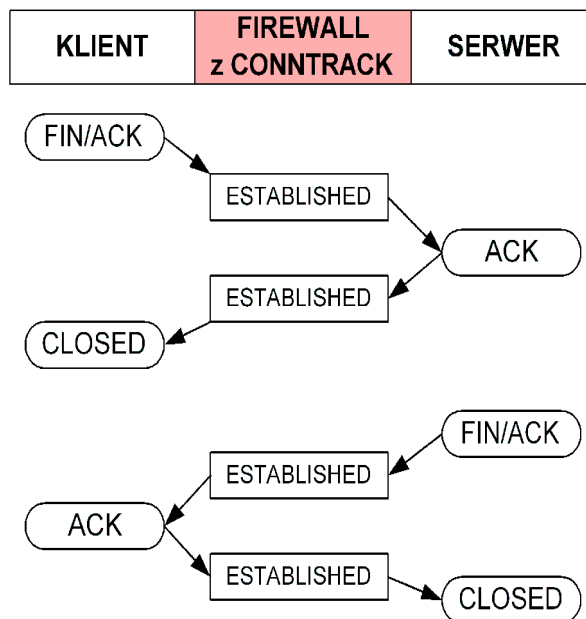
Należy odnotować, że fazy połączenia TCP nie do końca pokrywają się z tym co conntrack uznaje jako połączenie. Przy TCP mamy trzy fazy tworzenia połączenia: pakiety z ustawionymi flagami SYN, SYN+ACK i ACK. Conntrack uznaje połączenie za zestawione zaraz po otrzymaniu pierwszej odpowiedzi na pakiet z flagą SYN, czyli pakiet z SYN+ACK. Pozwala to na "wypuszczanie" przez firewall w kierunku świata jedynie pakietów ze stanem ESTABLISHED. Mimo, że połączenie nie jest jeszcze zestawione pozwala to na wychodzenie pakietów z sieci wewnętrznej zza firewalla.

Tak wygląda po krótko opisany proces umieszczania połączeń TCP w conntrack.

Po zamknięciu połączenia za pomocą pakietów z flagami FIN, przechodzi ono w stan TIME_WAIT i po 120 sekundach jest usuwane z tablicy.

```
tcp      6 118 TIME_WAIT src=192.168.1.11 dst=157.25.86.86 sport=32780 dport=22
          src=157.25.86.86 dst=192.168.1.11 sport=22    dport=32780 [ASSURED] use=1.
```

Jeśli połączenie jest zamykane za pomocą pakietu RST, wpis w conntrack jest usuwany po 10 sekundach a stan połączenia przechodzi w stan CLOSE.



Śledzenie bardziej złożonych protokołów

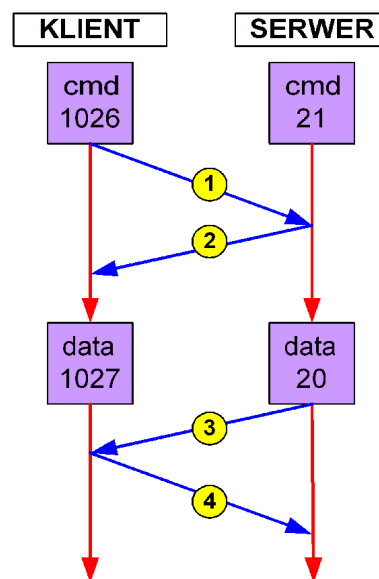
Trudno jest śledzić niektóre protokoły za pomocą zwykłych dopasowań na podstawie portów czy adresów źródłowych i docelowych. Protokoły, które mogą sprawiać problemy to np. ftp, h323, ptp i wiele innych. Ftp jest jednym z tych protokołów, których używanie jest nieco bardziej skomplikowane i któremu przyjrzymy się dokładniej. Ftp korzysta z dwóch połączeń do wykonywania swojej funkcji. Łącząc się z portem 21 klient wymienia polecenia i wysyła serwerowi ftp numer portu, na którym oczekuje połączenia z portu 20 serwera. Tak po krótkce wygląda połączenie aktywne. Problem z tego typu połączeniami jest taki, że pojawia się dodatkowa sesja, której conntrack i firewall normalnie się nie spodziewa. W przypadkach takich protokołów wykorzystuje się dodatki nazywane pomocnikami – "conntrack helpers". Obecnie w jądrach dostępne są trzy takie pomocniki, dla ftp, irc, tftp i amanda. W kernelu odpowiadają za nie kolejno opcje: CONFIG_IP_NF_FTP, CONFIG_IP_NF_IRC, CONFIG_IP_NF_TFTP, CONFIG_IP_NF_AMANDA. Dodatkowych rozszerzeń conntrack należy szukać na stronach <http://www.netfilter.org/patch-o-matic/>, gdzie znajdują się takie rozszerzenia.

AKTYWNE FTP

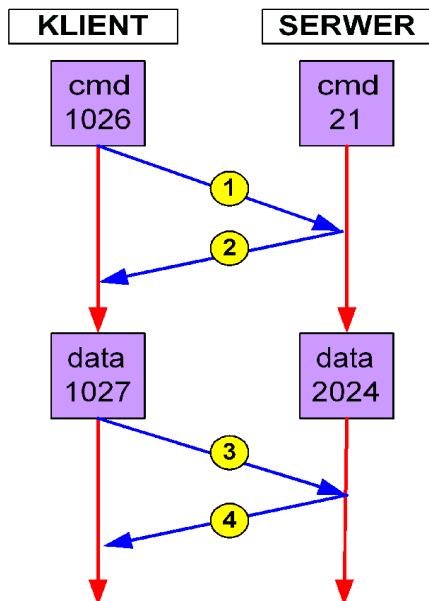
Aktywne ftp działa w następujący sposób:

1. klient nawiązuje połączenie z serwerem i za pomocą komendy PORT informuje go, że dane będą przesyłane za pomocą portu 1027.
2. wysyłając ACK serwer potwierdza połączenie.
3. serwer łączy się z portem 1027 po stronie klienta, wykorzystując port 20 jako źródłowy.
4. połączenie jest potwierdzone i następuje przesyłanie danych.

Firewall stojący pomiędzy klientem a serwerem często nie spodziewa się połączeń przychodzących do wnętrza sieci.



PASYWNE FTP



Pasywne ftp zostało rozwinięte jako odpowiedź na problemy związane z aktywnymi połączeniami. Pojawiło się polecenie PASV, które informowało serwer, że dalsze połączenia będą w trybie pasywnym.

Pasywne ftp działa w sposób następujący:

1.klient nawiązuje połączenie z serwerem i posyła komendę PASV.

2.serwer odpowiada poleceniem PORT podając port 2024, na którym serwer będzie nasłuchiwał połączeń od klienta.

3.klient otwiera połączenie z serwerem, łącząc się z portem 2024.

4.serwer potwierdza połączenie za pomocą ACK.

Tuning conntracka

Część parametrów conntrack można dopasować do swoich potrzeb. Jeśli mamy bardzo dużo połączeń i tablica conntrack wydaje się zbyt mała, możemy wpływać na parametry ustalone podczas kompilacji jądra systemu³.

```
#sysctl -a | grep conntrack
net.ipv4.ip_conntrack_max = 16376
net.ipv4.netfilter.ip_conntrack_generic_timeout = 600
net.ipv4.netfilter.ip_conntrack_icmp_timeout = 30
net.ipv4.netfilter.ip_conntrack_udp_timeout_stream = 180
...
net.ipv4.netfilter.ip_conntrack_buckets = 2047
net.ipv4.netfilter.ip_conntrack_max = 16376
```

Zmienne te można zmieniać poprzez wpisywanie wartości w pliki znajdujące się w katalogu /proc/sys/net/ipv4/netfilter bądź też poleceniem sysctl:

```
sysctl -w net.ipv4.ip_conntrack_max=32000
```

lub co daje identyczny wynik:

```
echo "32000" > /proc/sys/net/ipv4/netfilter/ip_conntrack_max
```

ip_conntrack_max definiuje ilość wpisów conntrack, które system może maksymalnie przechowywać. Każdy taki wpis to około 300b pamięci RAM. W systemach i386 wartość conntrack_max wyliczana jest według wzoru:

```
CONNTRACK_MAX = RAMSIZE (w bajtach) / 16384 = RAMSIZE (w MegaBajtach) * 64.
```

Wzorem, który sprawdzi się także dla platform 64 bitowych jest:

```
CONNTRACK_MAX = RAMSIZE (w bajtach) / 16384 / (x / 32) - gdzie x=32, 64bit
```

3 Opis tunowania parametrów conntrack_max i conntrack_buckets znajduje się pod adresem: http://www.wallfire.org/misc/netfilter_conntrack_perf.txt

Z tego wynika, że `conntrack_max` nie może być mniejszy od 128; dla systemów z ilością pamięci większą i równą 1GB `conntrack_max` będzie równy 65536, chyba że wartość ta zostanie ręcznie zmieniona.

Zmienna `ip_conntrack_buckets` określa wielkość tablicy hash, która pomaga znaleźć odpowiedni wpis `conntrack`. Tablica ta zawiera listę `conntrack_buckets`, których elementami są wpisy w `conntrack`. Czyli każda z tych list może zawierać maksymalnie `conntrack_max/conntrack_buckets` pozycji. Domyślnie w systemach i386 rozmiar `conntrack_buckets` określony jest prostym wzorem:

```
HASHSIZE = CONNTRACK_MAX / 8 = RAMSIZE (w bajtach) / 131072 = RAMSIZE (w MegaBajtach) * 8.
```

dla innych platform:

```
HASHSIZE = CONNTRACK_MAX / 8 = RAMSIZE (w bajtach) / 131072 / (x / 32) - gdzie x=32, 64 bity
```

Hashsize, czyli `conntrack_buckets` nie może być mniejszy niż 16, dla systemów z ilością pamięci większą i równą 1GB `conntrack_buckets` będzie równy 8192, chyba że wartość ta zostanie ręcznie zmieniona.

Zrozumienie `conntracka` może znacznie ułatwić pracę poprzez umożliwienie zlokalizowania problemów w sieci, czy choćby lepsze poznanie specyfiki ruchu w niej istniejącego. Znajomość `conntracka` ułatwia budowę firewalli opartych o statefull inspection, co stanowi temat na odrębny tekst:)

Linki

[1]http://www.sns.ias.edu/~jns/security/iptables/iptables_conntrack.html

[2]<http://iptables-tutorial.frozentux.net/chunkyhtml/statemachine.html>

[3]<http://www.tcpipguide.com>

[4]<http://tweegy.demon.nl/projects/netstat-nat/>

[5]<http://www.phildev.net/iptstate/>

[6]<http://cv.intellos.net/>