tcpdump

podstawy sniffowania i analizy

Łukasz Bromirski

lukasz@bromirski.net

Podstawy używania narzędzia `tcpdump`, oraz sposób analizy danych wyjściowych. Artykuł skupia się na zastosowaniu narzędzia w systemie FreeBSD, ale większość informacji dotyczy dowolnego środowiska.

Spis treści

1. Szybki start

tcpdump jest najpopularniejszym i najbardziej przenośnym snifferem. Są oczywiście łatwiejsze w użyciu i efektowniejsze graficznie, ale nie o to nam przecież chodzi ;)

1.1. Co muszę mieć?

Aby móc skorzystać z `tcpdump`a, musisz mieć po pierwsze kernel skompilowany z opcją `BPF` (*Berkeley Packet Filter*) czyli w konfiguracji kernela musi znaleźć się linijka:

```
options         BPF
```

Po drugie, musisz mieć urządzenie `/dev/bpf*` i dostęp do niego. Jeśli pracujesz na koncie **root** po normalnej instalacji systemu, wszystko to jest prawdą. Jeśli dostałeś do administracji po kimś starszy system, mogą być problemy ;).

1.2. Podstawy podstaw

Jednym z pierwszych parametrów, które warto dla porządku podać jest interfejs, z którego `tcpdump` ma zbierać pakiety. Bez tego parametru, `tcpdump` rozpocznie nasłuchiwanie na pierwszym interfejsie sieciowym systemu (przy czym jego wybór determinowany jest przez budowę wewnętrznych struktur systemu). Aby wskazać interfejs **xl0** należy wywołać program w ten sposób:

```
tcpdump -i xl0
```

Drugą istotną opcją jest wskazanie, gdzie skierować wyjście programu. Domyślnie program będzie wyrzucał pakiety pasujące do wzorca (o nim później) na standardowe wyjście (czyli w większości wypadków po prostu konsolę). Standardowo włączony jest mechanizm buforowania i pakiety będą pojawiały się z pewnymi opóźnieniami w stosunku do ich fizycznego odebrania przez `tcpdump` - jeśli chcesz ten mechanizm buforowania wyłączyć, dodaj do linii poleceń parametr '`-l`', tak jak poniżej:

```
tcpdump -l -i xl0
```

Oczywiście wyjście na konsolę nie zawsze jest wygodne - możliwe jest również zapisywanie zbieranych pakietów do pliku. Struktura pliku to właśnie legendarny format *tcpdump*, z którym zgodne są (tzn. potrafią go odczytać a często również zapisać dane w tym formacie) wszystkie liczące się sniffery (np. bardzo popularny obecnie ethereal). Aby zatem zmusić `tcpdump` do logowania informacji do pliku, dodajemy parametr '`-w nazwa_pliku`', gdzie *nazwa_pliku* to jak nietrudno zgadnąć, miejsce gdzie program zapisze dane:

```
tcpdump -i xl0 -w /tmp/zrzut01.dmp
```

Zwykle po zakończeniu podsłuchiwania, interesuje nas również proces odwrotny, tj. obejrzenie tego co zostało zalogowane. Można to wykonać zamieniając parametr '`-w nazwa_pliku`' na '`-r nazwa_pliku`'. Przy okazji, aby dane prezentowane były w dokładniejszej formie, można posłużyć się dodatkowymi parametrami:

'`-v`', '`-vv`' i '`-vvv`'

: służą do coraz dokładniejszego analizowania zawartości pakietu

'`-t`', '`-tt`', '`-ttt`' i '`-tttt`'

: powodują coraz dokładniejsze określenie czasu

'`-x`', '`-tt`', '`-ttt`' i '`-tttt`'

: zawartość pakietu prezentowana jest w postaci heksdecymalnej

'`-X`'

: zawartość pakietu prezentowana jest w postaci ASCII

Interpretację logów zostawmy sobie na koniec.

1.3. Wzorce

Prawdziwą siłą `tcpdump` jest możliwość konstruowania wzorców. Wzorzec to po prostu warunek logiczny sprawdzający jakąś właściwość pakietu (lub zbiór właściwości). Jeśli wynik sprawdzenia wzorca jest logiczną prawdą, pakiet zostaje zalogowany/zapisany/pokazany.

Jeśli chcesz zalogować cały ruch pomiędzy stacją *moja* a komputerami *serwer* lub *stacja1* powinieneś napisać:

```
tcpdump -i xl0 -w /tmp/zrzut02.dmp host moja and \(serwer or stacja1\)
```

> **Notatka:** Znaki backslash ('\') są konieczne, by powłoka nie zinterpretowała nawiasów jako swoich oznaczeń. Jest to prawda dla powłoki, w której przykłady te były testowane - *bash*. Twoja powłoka może mieć inny tzw. *escape-char*, sprawdź to na jej stronie podręcznika.

Częściej jednak chcesz zrzucić pakiety określonego protokołu skierowane od komputera o określonym IP (ew. z sieci) do innego komputera (sieci). Aby zalogować pakiety TCP z hosta *192.168.1.10*, skierowane do portu 80 możesz napisać:

```
tcpdump -i xl0 -w /tmp/zrzut03.dmp src 192.168.1.10 and tcp dst port 80
```

Ponieważ zwykle chodzi nam przy okazji o logowanie pakietów typu SYN (początków połączeń), musimy posłużyć się dodatkowo inną cechą `tcpdump`a: traktowaniem pakietu jak tablicy bajtów. Zaadresowanie *n*-tego bajtu pakietu, polega na odwołaniu się w ten sposób: `[n]`. Zapis ten należy poprzedzić specyfikacją, do nagłówka której warstwy się odwołujemy (np. `ether[x]` dla całej ramki Ethernetowej, `ip[x]` dla pakietów IP, `tcp[x]` dla pakietów TCP, `udp[x]` dla pakietów UDP, `icmp[x]` dla pakietów ICMP czy w końcu `ip6[x]` dla pakietów IPv6), a następnie podać warunek matematyczny i wartość. Najprostsze są warunki, w których po prostu sprawdzamy ustawienie bajtu (bitu) na określoną wartość. Wiedząc, że flaga SYN jest bitem znajdującym się w 13 bajcie nagłówka TCP i dodatkowo wiedząc, że jest to bit o numerze 1, możemy przechwycić wszystkie pakiety z ustawioną flagą SYN w ten sposób:

```
tcpdump -i xl0 -w /tmp/zrzut04.dmp src 192.168.1.10 and dst port 80 \
    and tcp\[13\] == 2
```

Mało przyjazne, prawda? `tcpdump` od wersji *3.7.1* oferuje jeszcze inną możliwość - zastąpienie numeru bajtu słowem kluczowym. Dla flag TCP i typów oraz rodzajów pakietów ICMP możemy posłużyć się zapisem słownym, co jest zdecydowanie korzystniejsze. Ten sam przykład wygląda wtedy tak:

```
tcpdump -i xl0 -w /tmp/zrzut04.dmp src 192.168.1.10 and dst port 80 \
    and tcp\[tcpflags\]\=tcp-syn
```

> **Notatka:** Nie rozumiesz dlaczego akurat 13 bajt i dlaczego ma mieć wartość 2? Pierwszą informację możesz zdobyć przeglądając odpowiednie RFC (dla określonego protokołu), w których dokładnie opisana jest struktura nagłówka. Polecam jednak legendarną już "Biblię Stevensona", czyli "TCP/IP Illustrated", w którym wszystko jest wytłumaczone dokładnie i jednocześnie językiem dużo bardziej przyjaznym niż ten używany w RFC. Książkę tą posiada większość dobrych księgarni, w tym sieć EMPiK - istotny jest przede wszystki pierwszy i trzeci tom, drugi poświęcony jest implementacjom, co nie musi być dla wszystkich ciekawe.
>
> Druga wartość bierze się z reprezentacji dwójkowej ciągu `00000010`, oznaczającego, że drugi bit ustawiony jest na *prawdę*. Tak się składa, że ten właśnie bit przyporządkowany jest fladze SYN. Wartością dwójkowego zapisu `00000010` jest właśnie 2, ponieważ jeśli rozpiszemy kolejne pozycje oktetu a poniżej ich wartość decymalną, otrzymujemy:
>
> ```
>   7     6     5     4     3     2     1     0
> 0*2 + 0*2 + 0*2 + 0*2 + 0*2 + 0*2 + 1*2 + 0*2  =  2
> ```

A jak łatwo wyłapać i zalogować pakiety NetBIOS, rozsyłane ze słabo zabezpieczonych stacji z systemem Microsoft Windows? Pakiety te przenosi protokół TCP i UDP a numery portów to zakres od 137 do 139. Mamy już w zasadzie wszystko co potrzeba do skonstruowania warunku:

```
tcpdump -i xl0 -w /tmp/zrzut05.dmp \(tcp or udp\) and
        \(dst port 137 or dst port 138 or dst port 139\)
```

Łatwo zauważyć, że posiadając takie narzędzie i inne, przeznaczone do zautomatyzowanego uzyskiwania z takich stacji list użytkowników, współdzielonych zasobów itd. łatwo przeprowadzić zautomatyzowany skan ;)

Ostatnia (powiedzmy ;))ciekawa rzecz do zalogowania: pakiety z podejrzanie małym TTL. Istnieje parę technik skanowania, łącznie nazywanych '*firewalking*', które umożliwiają zmapowanie sieci za ścianą ogniową, nie potrafiącą filtrować z utrzymaniem stanów. Jeśli używasz filtrowania z utrzymaniem stanów przy użyciu `ipfw` czy `ipf` nie masz się o co martwić (ściana ogniowa odrzuci losowo generowane pakiety jako nie należące do istniejących połączeń), ale warto wtedy wiedzieć, że takie próby są podejmowane. 'Podejrzanie' małe TTL to kwestia konkretnego skanowania lub narzędzia, ale generalnie wartości od 5 w dół są już podejrzane. Poniżej właśnie taki test:

```
tcpdump -i xl0 -w /tmp/zrzut06.dmp ip\[8\]\<5
```

2. Interpretacja zebranych danych

2.1. Protokół ICMP

Format pakietów ICMP określono w RFC 792.

Poniżej przykład standardowego pinga z komputera o adresie *192.168.0.10* do komputera *192.168.10.10*. Omówimy go z różną szczegółowością.

```
# tcpdump -r /tmp/zrzut01.dmp

23:33:39.280960 192.168.10.10 > 192.168.0.10: icmp: echo request (DF)
23:33:39.283636 192.168.0.10 > 192.168.10.10: icmp: echo reply (DF)
```

Kolejne pozycje oddzielane spacjami w pierwszej linijce to:

`23:33:39.280960`

: godzina w formacie *HH:MM:SS.MS*; format zapisu ściśle związany jest z podanymi opcjami '`-t`', '`-tt`' itd.

`192.168.10.10`

: adres IP hosta-nadawcy pakietu

`>`

: kierunek przepływu pakietu

`192.168.0.10`

: adres IP hosta-adresata pakietu

`icmp:`

: protokół ponad IP - w tym przypadku ICMP

`echo request`

: typ pakietu - w tym przypadku żądanie echa

`(DF)`

Ustawiony bit *Don't Fragment*, czyli nie fragmentować. Jeśli ten bit jest ustawiony, a z jakiegoś powodu router **musi** pakiet sfragmentować, pakiet zostanie odrzucony a kernel wygeneruje komunikat ICMP - *Need Fragmentation* (dotyczy to tylko TCP i UDP, generowanie takiego pakietu dla ICMP mogłoby spowodować pętle i nie jest stosowane)

Druga linijka obrazuje ruch pakietu w przeciwnym kierunku, ale typem pakietu jest już **echo reply** - czyli normalna odpowiedź na ping. Przyjrzyjmy się teraz pakietom dokładniej:

```
# tcpdump -v -r /tmp/zrzut01.dmp

23:33:39.280960 192.168.10.10 > 192.168.0.10: icmp: echo request (DF)
                (ttl 63, id 0, len 84)
23:33:39.283636 192.168.0.10 > 192.168.10.10: icmp: echo reply (DF)
                (ttl 64, id 17568, len 84)
```

Pomijając znane już pola, mamy:

`ttl 63`

TTL pakietu, wynosi 63. Standardowe TTL to zwykłe okrągłe potęgi dwójek, mamy więc systemy ustawiające na starcie TTL równe 32 (zwykle drukarki i bardzo stare systemy), 64 (większość *BSD i Linuksy po tuningu), 128 (Windowsy, część routerów) i 255 (również Windowsy i część routerów).

`id 0`

id pakietu, wynosi 0

`len 84`

długość całkowita pakietu wynosi 84 bajty

Zajrzyjmy w takim razie do środka pakietu - podając parametr `-x` otrzymujemy zrzut heksdecymalny jego zawartości, a dodając do tego `-X` dodatkowo reprezentację ASCII. Poniżej przykład:

```
# tcpdump -xX -r /tmp/zrzut01.dmp

23:33:39.280960 192.168.10.10 > 192.168.0.10: icmp: echo request (DF)
0x0000        4500 0054 0000 4000 3f01 b044 c0a8 0a0a       E..T..@.?..D....
0x0010        c0a8 000a 0800 ff51 4a04 0100 2a65 6b3d       .......QJ...*ek=
0x0020        2504 0800 0809 0a0b 0c0d 0e0f 1011 1213       %...............
0x0030        1415 1617 1819 1a1b 1c1d 1e1f 2021 2223       ............!"#
0x0040        2425 2627 2829 2a2b 2c2d 2e2f 3031 3233       $%&'()*+,-./0123
0x0050        3435                                          45
23:33:39.283636 192.168.0.10 > 192.168.10.10: icmp: echo reply (DF)
0x0000        4500 0054 44a0 4000 4001 6aa4 c0a8 000a       E..TD.@.@.j.....
0x0010        c0a8 0a0a 0000 0752 4a04 0100 2a65 6b3d       .......RJ...*ek=
0x0020        2504 0800 0809 0a0b 0c0d 0e0f 1011 1213       %...............
0x0030        1415 1617 1819 1a1b 1c1d 1e1f 2021 2223       ............!"#
0x0040        2425 2627 2829 2a2b 2c2d 2e2f 3031 3233       $%&'()*+,-./0123
0x0050        3435                                          45
```

2.2. Protokół TCP

ICMP jest mało ciekawe - TCP (odpowiednie RFC: 793 i uzupełnienia w 3168) daje cały ogrom nowych możliwości. Popatrzmy na pakiet TCP SYN do serwera HTTP nasłuchującego pod adresem 192.168.0.10 na porcie 80:

```
# tcpdump -r /tmp/zrzut02.dmp

23:57:41.621201 192.168.10.10.1029 > 192.168.0.10.80: S 2957663764:2957663764(0)
      win 5840 <mss 1460,sackOK,timestamp 190417 0,nop,wscale 0> (DF) [tos 0x10]
```

`S`

Pakiet ma ustawioną flagę SYN. Inne możliwe flagi to (**R**)ST, (**P**)USH, (**F**)IN, oraz kropka '**.**' na oznaczenie braku ustawionych flag. Flaga (**A**)CK oznaczana jest obecnością w dalszej części opisu słowa *ack* - w powyższym zrzucie nie jest ustawiona.

`2957663764:2957663764(0)`

Numer sekwencyjny - oddzielone dwukropkami pierwszy i ostatni numer sekwencyjny dla danych, a w nawiasie znajduje się różnica - jak widać, pakiet nie zawiera danych (`0`).

`win 5840`

Ogłoszenie rozmiaru 'okna' pakietu TCP, tutaj 5840 bajtów. O koncepcji okien (i tzw. *sliding windows*) najlepiej poczytać u Stevensa lub w RFC813.

`mss 1460`

Wartość MSS, czyli *Maximum Segment Size* (dosł. Maksymalny Rozmiar Segmentu). Segmenty danych (wartość nie dotyczy nagłówków ani innych opcji a jedynie dane) o długości większej niż ta wartość będą podlegały fragmentacji - chyba, że ustawiono w pakiecie bit '*Don't Fragment*' (jest ustawiony - `(DF)`). W takim wypadku, router nie mogąc sfragmentować pakietu odeśle do jego źródła pakiet ICMP - *Fragmentation needed*. Wartość MSS powinna być ustawiona według wzoru `MSS=MTU-rozmiar_nagłówka_TCP-rozmiar_nagłówka_IP`. Dla Ethernetu domyślną wartością MTU jest 1500. Jeśli założymy, że pakiet przenosi nagłówki IP i TCP w najkrótszych wersjach (po 20 oktetów) uzyskujemy wynik 1460 dla MSS. Więcej informacji możesz znaleźć w RFC879.

`sackOK`

Opcja informująca odbiorcę, że może wysyłać selektywne potwierdzenia poszczególnych części otrzymywanych danych. Więcej informacji znajduje się w RFC1072

`timestamp 190417 0`

Stempel czasowy z pakietu IP, 32-bitowa liczba modulo północ. Więcej w RFC781

`nop`

Wyrównanie, 'pusta' opcja. Na podstawie ułożenia opcji w pakiecie, a także ilości dodanych opcji 'nop' programy takie jak nmap, p0f v1/v2 czy xprobe2 próbują określić typ stosu TCP/IP - a pośrednio, system operacyjny.

`wscale 0`

Współczynnik skalowania okna. Jest to opcja określona w RFC1323 i w skrócie dotyczy tego, w jaki sposób (i czy w ogóle) hosty mogą uzgodnić skalowanie okna. Opcja ta powinna pojawiać się tylko podczas nawiązywania połączenia.

`(DF)`

Ustawiony bit '*Don't Fragment*'

`[tos 0x10]`

Pole ToS (ang. *Type of Service*, Typ Usługi) pakietu TCP. Ustawienie go jest charakterystyczne m.in. dla hostów pracujących pod Linuksem.

Odpowiedź przychodzi za moment i zawiera:

```
23:57:41.625459 192.168.0.10.80 > 192.168.10.10.1029: S 1472218990:1472218990(0)
   ack 2957663765 win 57344 <mss 1460,nop,wscale 0,nop,nop,timestamp 190382 190417>
```

`S`

Pakiet ma ustawioną flagę SYN

`1472218990:1472218990(0)`

Numer sekwencyjny, ponownie pakiet nie zawiera danych.

`ack 2957663765`

Potwierdzenie od zdalnego hosta, że następny pakiet powinien mieć numer sekwencyjny `2957663765` (pierwszy pakiet miał numer sekwencyjny równy `2957663764` i nie przenosił danych). W TCP potwierdzenia wysyła się wskazując *następny* bajt danych, który odbierający spodziewa się otrzymać.

`win 57344`

Ponownie ogłoszenie rozmiaru okna, tutaj widać rozmiar dużo większy - 57344 bajty.

`timestamp 190382 190417`

Stempel czasowy pakietu IP. Pakiet jest odpowiedzią na pakiet z ustawionym stemplem `190417`, obecny ma `190382` - różnica wynosi jak widać 35 milisekund.

No dobrze, przyjrzeliśmy się pierwszym dwóm pakietom z fazy nawiązywania połączenia. Normalne połączenie TCP powinno mieć swój początek - składający się z tzw. *three-way handshake*:

Pakiet od hosta inicjującego, ma ustawioną tylko flagę SYN

Pakiet od hosta odpowiadającego ma ustawione flagi SYN i ACK (jeśli chce odebrać połączenie)

Pakiet od hosta inicjującego ma ustawioną flagę ACK i zawiera pierwszą porcję danych.

Kolejne pakiety nie stanowią specjalnej zagadki, ponieważ powinny przenosić dane - popatrzmy od początku na zrzut pierwszych paru pakietów, podróżujących od i do hosta inicjującego połączenie:

```
23:57:41.621201 192.168.10.10.1029 > 192.168.0.10.80: S 2957663764:2957663764(0)
  win 5840 <mss 1460,sackOK,timestamp 190417 0,nop,wscale 0> (DF) [tos 0x10]
23:57:41.625459 192.168.0.10.80 > 192.168.10.10.1029: S 1472218990:1472218990(0)
  ack 2957663765 win 57344 <mss 1460,nop,wscale 0,nop,nop,timestamp 190382 190417>
23:57:41.625856 192.168.10.10.1029 > 192.168.0.10.80: . ack 1 win 5840
  <nop,nop,timestamp 190417 190382> (DF) [tos 0x10]
23:57:44.819184 192.168.10.10.1029 > 192.168.0.10.80: P 1:18(17) ack 1 win 5840
  <nop,nop,timestamp 190803 190382> (DF) [tos 0x10]
23:57:45.043438 192.168.0.10.80 > 192.168.10.10.1029: P 1:279(278) ack 18 win 57920
```

```
  <nop,nop,timestamp 190766 190803> (DF)
```

W trzeciej z kolei linijce widać wpis `ack 1`. Normalnie, `tcpdump` po podaniu pełnych numerów sekwencyjnych dla połączenia, kolejne podawać będzie już relatywnie - w kolejnej linicje widzimy pakiet z flagą PUSH (`P`) i numery `1:18(17)`. Oznacza to po prostu, że pakiet zawierał 17 bajtów danych a `relatywne` numery sekwencyjne tych danych to od 1 do 18. Aby zawsze `tcpdump` wyświetlał faktyczne numery sekwencyjne, użyj opcji '`-S`'.

2.2.1. Poprawne i niepoprawne kombinacje flag TCP

'Normalne' kombinacje flag to takie, które opisane w odpowiednich RFC mogą się zdarzyć. 'Nienormalne' kombinacje flag, to takie których nigdzie oficjalnie nie opisano. Są zwykle używane przez programy typu nmap, p0f v1/v2 czy xprobe2 do określenia rodzaju stosu TCP/IP - a pośrednio systemu operacyjnego. Poniżej małe podsumowanie 'normalnych' kombinacji:

`SYN`

: prośba o nawiązanie połączenia

`SYN+ACK`

: potwierdzenie prośby o nawiązanie połączenia

`ACK`, czasami również `PSH+ACK` i ewentualnie `URG`

: Normalny pakiet przenoszący dane, lub pojawiający się jako odpowiedź na pakiet z ustawionymi flagami `FIN+ACK` - opis niżej. Tutaj jest wyjątek - pakiet ACK nigdy nie powinien zawierać potwierdzenia dla bajtu o numerze *0*.

`FIN+ACK` lub `PSH+FIN+ACK`

: Normalne zakończenie połączenia.

`RST` lub `RST+ACK`

: Awaryjne zakończenie połączenia

...i dla kontrastu pakiety które 'nie powinny się zdarzyć':

`SYN+FIN`

: Bardzo często spotykana nieprawidłowa kombinacja flag.

`FIN`

: Flaga FIN nigdy nie powinna znaleźć się w pakiecie jako jedyna ustawiona.

  Czyli pusto :) - to również nieprawidłowa kombinacja. Jakaś flaga **musi** być ustawiona.

2.2.2. Typowe skanowania

Poniżej parę typowych przykładów skanowania. Twój snort powinien wyłapać je bez problemu:

```
00:34:14.656069 192.168.10.10.1030 > 192.168.0.10.1432: S 3805853940:3805853940(0)
                   win 5840 <mss 1460,sackOK,timestamp 270815 0,nop,wscale 0> (DF)
00:34:14.656096 192.168.10.10.1031 > 192.168.0.10.1539: S 3809182350:3809182350(0)
                   win 5840 <mss 1460,sackOK,timestamp 270815 0,nop,wscale 0> (DF)
```

```
00:34:14.656107 192.168.10.10.1032 > 192.168.0.10.946: S 3809888126:3809888126(0)
                   win 5840 <mss 1460,sackOK,timestamp 270815 0,nop,wscale 0> (DF)
00:34:14.656116 192.168.10.10.1033 > 192.168.0.10.138: S 3807752285:3807752285(0)
                   win 5840 <mss 1460,sackOK,timestamp 270815 0,nop,wscale 0> (DF)
00:34:14.656126 192.168.10.10.1034 > 192.168.0.10.32787: S 3817414276:3817414276(0)
                    win 5840 <mss 1460,sackOK,timestamp 270815 0,nop,wscale 0> (DF)
00:34:14.658631 192.168.10.10.1035 > 192.168.0.10.270: S 3813560255:3813560255(0)
                  win 5840 <mss 1460,sackOK,timestamp 270815 0,nop,wscale 0> (DF)
00:34:14.658646 192.168.10.10.1036 > 192.168.0.10.717: S 3811826777:3811826777(0)
                  win 5840 <mss 1460,sackOK,timestamp 270815 0,nop,wscale 0> (DF)
00:34:14.658657 192.168.10.10.1037 > 192.168.0.10.5002: S 3809137986:3809137986(0)
                   win 5840 <mss 1460,sackOK,timestamp 270815 0,nop,wscale 0> (DF)
00:34:14.671598 192.168.0.10.1432 > 192.168.10.10.1030: R 0:0(0) ack 3805853941 win 0
00:34:14.677547 192.168.0.10.1539 > 192.168.10.10.1031: R 0:0(0) ack 3809182351 win 0
00:34:14.684308 192.168.0.10.946 > 192.168.10.10.1032: R 0:0(0) ack 3809888127 win 0
00:34:14.691409 192.168.0.10.138 > 192.168.10.10.1033: R 0:0(0) ack 3807752286 win 0
00:34:14.697630 192.168.0.10.32787 > 192.168.10.10.1034: R 0:0(0) ack 3817414277 win 0
00:34:14.703812 192.168.0.10.270 > 192.168.10.10.1035: R 0:0(0) ack 3813560256 win 0
00:34:14.704959 192.168.0.10.717 > 192.168.10.10.1036: R 0:0(0) ack 3811826778 win 0
00:34:14.706027 192.168.0.10.5002 > 192.168.10.10.1037: R 0:0(0) ack 3809137987 win 0
```

Typowy przykład skanowania SYN (lub *floodingu SYN*) - masa pakietów wysłanych w bardzo krótkim okresie czasu na losowe porty komputera *192.168.0.10*. Po nich widzimy odpowiedź z ustawionymi flagami `RST+ACK` - z jakiegoś powodu host nie chce obsłużyć tych połączeń. Do wygenerowania takiej powodzi pakietów może posłużyć flaga `-sS` programu nmap.

```
00:41:19.543023 192.168.10.10.36957 > 192.168.0.10.357: S \
                 [bad hdr length] (frag 302:16@0+)
00:41:19.543048 192.168.10.10 > 192.168.0.10: (frag 302:4@16)
00:41:19.543058 192.168.10.10.36957 > 192.168.0.10.164: S \
                 [bad hdr length] (frag 37343:16@0+)
00:41:19.543066 192.168.10.10 > 192.168.0.10: (frag 37343:4@16)
00:41:19.543083 192.168.10.10.36957 > 192.168.0.10.683: S \
                 [bad hdr length] (frag 33687:16@0+)
00:41:19.543092 192.168.10.10 > 192.168.0.10: (frag 33687:4@16)
```

Tu widzimy jeszcze jedną właściwość `tcpdump` jeśli chodzi o pakiety - fragmentację. Pierwsza linijka pokazuje pakiet z ustawioną flagą SYN (`S`), ale o złej długości nagłówka (`bad hdr length`) a następnie podaje informacje jak skonstruowany jest fragment - (`frag 302:16@0+`). `302` oznacza identyfikator fragmentu, `16` to rozmiar fragmentu bez nagłówka IP a `0+` oznacza po prostu offset tego fragmentu w całości - widzimy, że jest to pierwszy fragment. Błąd `[bad hdr length]` spowodowany jest tym, że pierwszy fragment zawierał jakieś opcje, które nie zmieściły się w tym fragmencie - jest to niedopuszczalne. Inny błąd, który może pojawić się podczas skanowania to `[bad opt]` - oznaczające, że opcje podane we fragmencie są nieprawidłowe (np. za krótki rozmiar pola lub długość przekraczająca długość całości).

W drugiej linijce mamy jeszcze mniejszy pakiet, ale będący najwyraźniej częścią pierwszego - id jest równe `302`, rozmiar `4` a offset `16`. Ten pakiet, jako kolejny należący do otrzymanego już fragmentu jest odbierany poprawnie.

```
00:37:31.363354 192.168.10.10.61949 > 192.168.0.10.80: . \
   ack 3414377251 win 3072
00:37:31.594794 192.168.0.10.80 > 192.168.10.10.61949: R \
   3414377251:3414377251(0) win 0
```

Tutaj pakiet, który przyszedł z hosta *192.168.10.10* od razu z ustawioną flagą *ACK* (`ack`) na losowo najwyraźniej wybrany numer. W drugiej linijce widzimy odpowiedź systemu, który nie ma w swoich

buforach sieciowych połączenia pasującego do tych parametrów - pakiet ma ustawioną flagę *RESET* (`R`). Podobnie poniżej - system nie wie nic o połączeniu z `192.168.10.10:61929` z dziwnie ustawionymi flagami *FIN* i *PSH* (`FP`) więc odpowiada pakietem *RST* (druga linijka, `R`):

```
00:37:31.594995 192.168.10.10.61929 > 192.168.0.10.957: FP 0:0(0) win 3072 urg 0
00:37:31.605806 192.168.0.10.957 > 192.168.10.10.61929: R 0:0(0) ack 0 win 0
```

2.3. Protokół UDP

Format pakietów UDP określone jest w RFC 768.

Poniżej krótki przykład pakietu UDP - zapytanie do serwera DNS:

```
01:24:47.004874 192.168.0.10.1044 > 192.168.0.1.53:
                    15696+ PTR? 1.0.168.192.in-addr.arpa. (44)
```

Widać, że komputer o adresie *192.168.0.10* wysyła ze swojego portu `1044`, zapytanie do komputera o adresie `192.168.0.1` na port `53`. `tcpdump` jest w stanie zinterpretować proste rekordy DNS i tutaj widzimy taką interpretację: `15696` oznacza identyfikator zapytania wskazujący na zapytania dla klienta; `+` oznacza ustawioną flagę ‘pożądana rekursja’, oznaczająca, że jeśli możliwe będzie odpytanie ‘w górę’ serwerów DNS to należy to zrobić; `PTR?` to zapytanie o adres IP dla wskazanej nazwy a cały pakiet zawierał `44` bajty danych.

2.3.1. Typowe skanowania

Podstawowy typ skanowania - wysyłamy puste (`udp 0`) pakiety UDP na port i czekamy, czy system zdalny odpowie komunikatem ICMP, czy będzie milczał. Poniżej, na wskazanych portach nic nie nasłuchuje i system pod adresem *192.168.0.10* posłusznie zwraca pakiety ICMP:

```
00:36:02.827338 192.168.10.10.42242 > 192.168.0.10.8:  udp 0
00:36:02.827355 192.168.10.10.42242 > 192.168.0.10.904:  udp 0
00:36:02.827365 192.168.10.10.42242 > 192.168.0.10.1415:  udp 0
00:36:02.832504 192.168.0.10 > 192.168.10.10: icmp: 192.168.0.10 udp \
                 port 8 unreachable
00:36:02.834207 192.168.0.10 > 192.168.10.10: icmp: 192.168.0.10 udp \
                 port 904 unreachable
00:36:02.835273 192.168.0.10 > 192.168.10.10: icmp: 192.168.0.10 udp \
                 port 1415 unreachable
```

3. Podsumowanie

`tcpdump` to naprawdę potężne i wydajne narzędzie. Jak to zwykle z takimi bywa - zastosowań są miliony, musisz tylko wiedzieć co chcesz logować. A jak widać po powstaniu czegoś takiego jak Museum of the broken packets lcamtufa, pakietów godnych uwagi jest całkiem sporo :).

4. Materiały

The Art of Port Scanning, Fyodor, Phrack 51

http://www.insecure.org/nmap/nmap_doc.html

Remote OS detection via TCP/IP Stack Fingerprinting, Fyodor

http://www.insecure.org/nmap/nmap-fingerprinting-article.html

Designing and Attacking Port Scan Detection Tools, solar designer

http://www.phrack.org/phrack/53/P53-13

ICMP Usage in Scanning, Ofir Arkin

http://www.sys-security.com/archive/papers/ICMP_Scanning_v3.0.pdf

Identifying ICMP Hackery Tools Used In The Wild Today, Ofir Arkin

http://www.sys-security.com/archive/securityfocus/icmptools.html

Network Scanning Techniques, Ofir Arkin

http://www.sys-security.com/archive/papers/Network_Scanning_Techniques.pdf