Programowanie sieciowe

Filtracja ramek nasłuchiwanych (biblioteka libpcap)

Michał Kalewski

Instytut Informatyki
Politechnika Poznańska
ul. Piotrowo 2, 60–965 Poznań

mkalewski@cs.put.poznan.pl
http://www.cs.put.poznan.pl/mkalewski

Poznań · 19 marca 2019 r.

Plan wykładu

Analizatory komunikacji sieciowej

Analizatory komunikacji sieciowej

Najpopularniejszymi analizatorami sieciowymi dla systemów operacyjnych UNIX oraz GNU/Linux są `tcpdump(1)` (narzędzie konsolowe) oraz `wireshark` (aplikacja z graficznym interfejsem użytkownika) — lub `tshark`, tj. wersja konsolowa tej aplikacji.

Oba rozwiązania wykorzystują bibliotekę `libpcap` do nasłuchiwania i filtracji ramek sieciowych.

Analizator ruchu sieciowego `tcpdump(1)` dostępny jest pod następującym adresem internetowym: http://www.tcpdump.org.

Analizator ruchu sieciowego `wireshark` (oraz `tshark`) dostępny jest pod następującym adresem internetowym: https://www.wireshark.org.

Analizator komunikacji sieciowej `tcpdump(1)` (1/2)

Analizator komunikacji sieciowej `tcpdump(1)` wypisuje na standardowe wyjście informacje o wszystkich nasłuchiwanych ramkach sieciowych, które spełniają podane, jako argument wywołania, *wyrażenie filtracji*. Wyrażenie filtracji może zostać pominięte, jeżeli ramki nasłuchiwane nie mają podlegać filtracji.

Wyrażenie filtracji składa się z jednego lub więcej *wyrażenia podstawowego*, które, z kolei, składa się z *nazwy* lub *numeru* oraz *określenia*.

Istnieją trzy różne określenia:

`type` typ, do którego odnosi się wyrażenie podstawowe (np. `host`, `net` lub `port`); typem domyślnym jest `host`.

`dir` kierunek transmisji danych, do którego odnosi się wyrażenie podstawowe (np. `src`, `dst` lub adres); kierunkiem domyślnym jest `src or dst`.

`proto` protokół, do którego odnosi się wyrażenie podstawowe (np. `ether`, `wlan` lub `ip`); domyślnie wyrażenie podstawowe odnosi się do wszystkich protokołów poprawnych dla wybranego typu.

Analizator komunikacji sieciowej `tcpdump(1)` (2/2)

Przełączniki podstawowe komendy `tcpdump(1)`:

`-i interface` nazwa interfejsu sieciowego, na którym nasłuchiwane mają być ramki sieciowe (interfejsy `pktap` i `all`, przełącznik `-D`).

`-n` włączenie wyświetlania numerycznego adresów IP i numerów portów.

`-v, -vv, -vvv` szczegółowość wypisywanych informacji o nasłuchiwanych ramkach.

`-X, -XX` włączenie wypisywania na standardowe wyjście danych lub nagłówków i danych nasłuchiwanych ramek.

`-w file` włączenie zapisywania nasłuchiwanych ramek do pliku o podanej nazwie.

zob. `tcpdump(1)`

Przykładowe wyrażenia filtracji ramek sieciowych

Przykładowe wyrażenia filtracji ramek sieciowych:

- `host sundown`
- `host helios and \( hot or ace \)`
- `'src helios and (dst port 80 or 443)'`
- `ip host ace and not helios`
- `src net 192.168.0.0/24 and dst net 10.0.0.0/8 or 172.16.0.0/16`
- `not arp and not rarp`

zob. `pcap-filter(7)`

PRZYKŁAD:

```
$ tcpdump -ni eth0 icmp
16:42:02.680728 IP 192.168.10.1 > 172.16.20.2: ICMP echo request,
id 42364, seq 4632, length 12
```


Konstrukcja logiczna analizatora `tcpdump(1)`

Analizator komunikacji sieciowej `tcpdump(1)` składa się z trzech części:

1. Parsera wyrażeń filtracji, który przekształca wyrażenia filtracji do postaci kodu wykonywalnego *filtrów BPF/LSF*.
2. Modułu stosującego kod wykonywalny filtrów BPF/LSF do wskazanego interfejsu sieciowego.
3. Modułu przekształcającego odebrane po filtracji ramki sieciowe do postaci przekazywanej na standardowe wyjście.

Linux Socket Filtering i Berkeley Packet Filter (1/3)

Wyrażenia filtracji ramek sieciowych stosowanie są do gniazd sieciowych PF_PACKET z użyciem mechanizmów Berkeley Packet Filter (BPF) w systemach operacyjnych UNIX oraz Linux Socket Filtering (LSF) w systemach operacyjnych GNU/Linux.

LSF jest rozwiązaniem wywodzącym się z BPF.

Filtry BPF i LSF wykonywane są przez jądro systemu operacyjnego. W systemie operacyjnym mogą one być widoczne jako *specjalne urządzenia znakowe* w katalogu /dev (pliki o nazwach bpfN, gdzie N to numer kolejny urządzenia).

Wyrażenie filtracji ramek sieciowych można przekształcić w kod BPF/LSF przy użyciu komendy tcpdump(1) z przełącznikami -d (pseudokod), -dd (fragment kodu języka C) lub -ddd (liczby dziesiętne).

zob. bpf(2)

zob. http://www.cs.put.poznan.pl/mkalewski/files/ps-bpf-asm.pdf

Linux Socket Filtering i Berkeley Packet Filter (2/3)

PRZYKŁAD:

```
$ tcpdump -d "ip and udp"
(000) ldh  [12]
(001) jeq  #0x800  jt 2  jf 5
(002) ldb  [23]
(003) jeq  #0x11   jt 4  jf 5
(004) ret  #65535
(005) ret  #0
$ tcpdump -dd "ip and udp"
{ 0x28, 0, 0, 0x0000000c },
{ 0x15, 0, 3, 0x00000800 },
{ 0x30, 0, 0, 0x00000017 },
{ 0x15, 0, 1, 0x00000011 },
{ 0x6,  0, 0, 0x0000ffff },
{ 0x6,  0, 0, 0x00000000 },
$ tcpdump -ddd "ip and udp" | tr "\n" ","
6,40 0 0 12,21 0 3 2048,48 0 0 23,21 0 1 17,6 0 0 65535,6 0 0 0
```


</> Linux Socket Filtering i Berkeley Packet Filter (3/3)

```c
struct sock_filter code[] = {
  { 0x28, 0, 0, 0x0000000c },
  { 0x15, 0, 3, 0x00000800 },
  { 0x30, 0, 0, 0x00000017 },
  { 0x15, 0, 1, 0x00000011 },
  { 0x6,  0, 0, 0x0000ffff },
  { 0x6,  0, 0, 0x00000000 }
};
struct sock_fprog bpf = {
  .len = ARRAY_SIZE(code),
  .filter = code
};
sfd = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
setsockopt(sfd, SOL_SOCKET, SO_ATTACH_FILTER,
           &bpf, sizeof(bpf));
```

https://github.com/torvalds/linux/blob/master/include/uapi/linux/filter.h

Biblioteka libpcap

Biblioteka libpcap

Biblioteka libpcap umożliwia nasłuchiwanie oraz filtrowanie ramek sieciowych; jest ona dostępna dla systemów operacyjnych UNIX, GNU/Linux oraz Windows (jako WinPcap).

Implementacja programów z użyciem biblioteki libpcap w systemach operacyjny GNU/Linux wymaga instalacji pakietu libpcap-dev (*development library for libpcap*), a kompilacja takich programów (np. kompilatorem gcc) wymaga jawnego linkowania tej biblioteki (-lpcap w przypadku kompilatora gcc).

Biblioteka libpcap dostępna jest pod następującym adresem internetowym: http://www.tcpdump.org.

Wersja WinPcap (rozwijana przez firmę Riverbed Technology) dla systemów operacyjnych Windows dostępna jest pod następującym adresem internetowym: http://www.winpcap.org.

Wybrane funkcje biblioteki libpcap (1/2)

```
char errbuf[PCAP_ERRBUF_SIZE];

void pcap_perror(pcap_t *p, const char *prefix);

pcap_t *pcap_create(const char *source, char *errbuf);
int pcap_activate(pcap_t *p);

int pcap_set_promisc(pcap_t *p, int promisc);
int pcap_set_snaplen(pcap_t *p, int snaplen);

int pcap_lookupnet(const char *device, bpf_u_int32 *netp,
                   bpf_u_int32 *maskp, char *errbuf);
int pcap_compile(pcap_t *p, struct bpf_program *fp,
                 const char *str, int optimize,
                 bpf_u_int32 netmask);
int pcap_setfilter(pcap_t *p, struct bpf_program *fp);
```


Wybrane funkcje biblioteki libpcap (2/2)

```
struct pcap_pkthdr {
  struct timeval ts;   /* time stamp                      */
  bpf_u_int32 caplen;  /* length of portion present       */
  bpf_u_int32 len;     /* length this packet (off wire) */
};
typedef void (*pcap_handler)(u_char *user,
    const struct pcap_pkthdr *h, const u_char *bytes);

int pcap_loop(pcap_t *p, int cnt,
              pcap_handler callback, u_char *user);

int pcap_next_ex(pcap_t *p, struct pcap_pkthdr **pkt_header,
                 const u_char **pkt_data);

int pcap_inject(pcap_t *p, const void *buf, size_t size);
```

zob. pcap(3pcap)

</> Nasłuchiwanie ramek sieciowych

```
1  char* errbuf;
2  pcap_t* handle;
3
4  errbuf = malloc(PCAP_ERRBUF_SIZE);
5  handle = pcap_create(argv[1], errbuf);
6  pcap_set_promisc(handle, 1);
7  pcap_set_snaplen(handle, 65535);
8  pcap_activate(handle);
9  pcap_loop(handle, -1, trap, NULL);
```

</> Filtracja nasłuchiwanych ramek sieciowych (1/2)

```c
#include <pcap.h>
#include <signal.h>
#include <stdlib.h>
#include <string.h>

char* errbuf;
pcap_t* handle;

void cleanup() { pcap_close(handle); free(errbuf); }
void stop(int signo) { exit(EXIT_SUCCESS); }
void trap(u_char *user, const struct pcap_pkthdr *h,
          const u_char *bytes) {
  printf("[%dB of %dB]\n", h->caplen, h->len);
}
```

cdn.

Filtracja nasłuchiwanych ramek sieciowych (2/2)

ciąg dalszy

```c
int main(int argc, char** argv) {
  bpf_u_int32 netp, maskp;  struct bpf_program fp;
  atexit(cleanup);
  signal(SIGINT, stop);
  errbuf = malloc(PCAP_ERRBUF_SIZE);
  handle = pcap_create(argv[1], errbuf);
  pcap_set_promisc(handle, 1);
  pcap_set_snaplen(handle, 65535);
  pcap_activate(handle);
  pcap_lookupnet(argv[1], &netp, &maskp, errbuf);
  pcap_compile(handle, &fp, argv[2], 0, maskp);
  if (pcap_setfilter(handle, &fp) < 0) {
    pcap_perror(handle, "pcap_setfilter()");  exit(-1);
  }
  pcap_loop(handle, -1, trap, NULL);
}
```


Podsumowanie

Podsumowanie

- Analizator komunikacji sieciowej `tcpdump(1)`.
- Linux Socket Filtering (LSF) oraz Berkeley Packet Filter (BPF).
- Biblioteka `libpcap`:
  - nasłuchiwanie ramek sieciowych;
  - filtracja nasłuchiwanych ramek sieciowych.

Pytania i zadania

Pytania i zadania — praca własna

1. Zapoznaj się z przełącznikami analizatora komunikacji sieciowej `tcpdump(1)`.
2. Zapoznaj się z wyrażeniamu filtracji ramek sieciowych, *zob.* `pcap-filter(7)`.
3. Zapoznaj się z dokumentacją BPF, *zob.* `bpf(2)`.
4. Przeczytaj artykuł pt. „Programming with Libpcap - Sniffing the network from our own application”, Martin Garcia, Hakin9, 2/2008. (http://recursos.aldabaknocking.com/libpcapHakin9LuisMartinGarcia.pdf).
5. Na czym polega, przedstawiony w powyższym artykule, atak typu TCP RST?
6. Sprawdź, jak można wykorzystać filtry BPF/LSF w połączeniu z narzędziem `iptables(8)`.

Literatura dodatkowa:

Linux Socket Filtering aka Berkeley Packet Filter (BPF):
https://www.kernel.org/doc/Documentation/networking/filter.txt

Dziękuję za uwagę!