

## Zdalne otwieranie/zamykanie portów bez potrzeby logowania

# STUK PUK



KnockD to program umożliwiający zdalne otwieranie i zamykanie portów bez konieczności logowania. Aby to zrobić, wystarczy „zapukać” do wcześniej zdefiniowanych portów. Z KnockD zarówno stawianie demona nasłuchującego, jak i „pukanie” na określone porty staje się bardzo proste.

**DAMIAN ZELEK**

Otwarty port na maszynie podłączonej do sieci, zarówno tej globalnej jak i lokalnej, stanowi zawsze dodatkowe ryzyko dla naszego komputera (kolejna możliwość dla potencjalnego włamywacza). Jeżeli port ten jest niezbędny do poprawnej pracy, np. jak to ma niekiedy miejsce z portem 22, zwykle wykorzystywanym przez ssh, jego wyłączenie staje się niemożliwe. Co robić w takiej sytuacji? Czy istnieje jakieś sensowne wyjście? Tak, jest wiele rozwiązań tego problemu, a jednym z nich jest projekt KnockD [1].

### Jak to działa...

KnockD jest to program, który po uruchomieniu jako demon zaczyna nasłuchiwać ca-

ły ruch na danym interfejsie (standardowo eth0). Co więcej, porty, na których nasz demon będzie nasłuchiwał, nie muszą być włączone, ponieważ jak już wcześniej wspominałem, nasłuchuje on *cały ruch na danym interfejsie*. Jeżeli program wykryje krótkie połączenia na porty (tzw. pukanie), które sami wcześniej zdefiniowaliśmy w pliku konfiguracyjnym, uruchomi odpowiednią procedurę. My chcemy, aby tą procedurą było otwarcie jakiegos portu, np. SSH. Należy pamiętać, iż owo pukanie nie zaloguje nas automatycznie, a jedynie otworzy wcześniej zamknięty port. Dzięki temu zyskujemy wyższy poziom bezpieczeństwa i o jeden otwarty port mniej, a włamywacz kolejne drzwi do

wyważenia. Owo „obejście” drzwi przez włamywacza nam niczym nie grozi (z założenia; o zagrożeniach przeczytasz w osobnej sekcji), a jedynie udostępni port SSH, czyli tak jak gdyby KnockD w ogóle nie był uruchomiony. Klient dołączony do KnockD znacznie ułatwia proces „pukania” automatyzując go. Demon nasłuchuje na przychodzące pakiety typu TCP i/lub UDP, tak więc można również bez specjalnego klienta przeprowadzić ten proces, korzystając z takich narzędzi jak Telnet, NetCat czy sendip.

### Instalacja i konfiguracja

Z głównej strony projektu [1] ściągamy nasz program. Mamy do wyboru źródelka (z tego właśnie skorzystamy), pakiet dla Debiana, RPM (np. Fedora, Mandrake) oraz plik .exe (Windows). Jak wcześniej wspominałem, interesują nas źródła programu. Po ich ściągnięciu rozpakowujemy archiwum tar.gz komendą `tar -zxvf nasz_plik`. Następnie wykonujemy 3 słynne i rutynowe polecenia: `./configure`, `make`, `make install`. Cała konfiguracja,

```

int o_verbose = 0;
int o_no_daemon = 0;
int o_debug = 0;
int o_listen = 0;
char o_iface[32] = "eth0";
char o_cfg[PATH_MAX] = "/etc/knockd.conf";
char o_pidfile[PATH_MAX] = "/var/run/knockd.pid";
char o_logfile[PATH_MAX] = "-";

int main(int argc, char **argv)
{
    struct pcap_pkthdr *h;
    int ret, optidx = 0;
    struct option opts[] =
    {
        { "verbose", no_argument, 0, 'v' },
        { "debug", no_argument, 0, 'd' },
        { "listen", no_argument, 0, 'l' },
        { "iface", required_argument, 0, 'i' },
        { "pidfile", required_argument, 0, 'p' },
        { "logfile", no_argument, 0, 'f' },
        { "no-daemon", no_argument, 0, 'n' },
        { 0, 0, 0, 0 }
    };

    while((ret = getopt_long(argc, argv, "vdl:ip:f:n", opts, &optidx)) != -1)
        break;

    switch(opt) {
        case 'v': o_verbose = 1; break;
        case 'd': o_debug = 1; break;
        case 'l': o_listen = 1; break;
        case 'i': o_iface = optarg; break;
        case 'p': o_pidfile = optarg; break;
        case 'f': o_logfile = optarg; break;
        case 'n': o_no_daemon = 1; break;
    }
}

```

Rysunek 1: Źródła naszego programu...

### Ramka 1: Najważniejsze opcje KnockD

Opcje uaktywniamy z konsoli jako normalny parametr programu KnockD:

- i <inter> // wybieramy interfejs do nasłuchiwania (standardowo eth0)
- c <plik> // ustalamy miejsce pliku konfiguracyjnego (standardowo `/etc/knockd.conf`)
- d // program działa jako demon
- h // dostępna pomoc

kompilacja oraz instalacja przechodzi szybko i nie powinna stanowić problemu nawet dla początkujących użytkowników Linuksa.

Konfiguracja KnockD ogranicza się do dwóch etapów: edycji pliku konfiguracyjnego (standardowo `/etc/knockd.conf`) oraz ustawieniu kilku opcji. Uruchamianie programu z określonymi opcjami odbywa się poprzez wiersz poleceń. Najważniejsze opcje podano w Ramce 1.

Następnym etapem dostrajania KnockD do naszych potrzeb jest edycja pliku konfiguracyjnego (swoistego centrum zarządzania i kontroli). Na Listingu 1 przedstawiono przykładowy wygląd tego pliku (budowa przejrzysta i bardzo prosta). Pokróćce opiszę najważniejsze jego procedury i zasady tworzenia. Oto one:

```
LogFile = xxx
```

plik dziennika (xxx to ścieżka do pliku). Aby przełączyć strumień wyjścia logów na `syslog` wystarczy zamiast `LogFile` wpisać `UseSyslog`.

```
Sequence = port:typ,port:typ
```

Sekwencja, która określa miejsce „pukania” oraz typ pakietów (TCP i/lub UDP). Dla dwóch różnych portów występujących w tej samej sekwencji możliwe jest korzystanie z pakietów różnych od siebie: zarówno TCP jak i UDP.

### Listing 1: Prosty plik konfiguracyjny

```
[options]
UseSyslog
[ssh_open]
sequence = 7001,7002,7003,7004
seq_timeout = 10
tcpflags = syn
command = /usr/sbin/iptables -A INPUT -s %IP% -p tcp --dport 22 -j ACCEPT
[ssh_close]
sequence = 7005,7006,7007,7008
seq_timeout = 10
tcpflags = syn
command = /usr/sbin/iptables -D INPUT -s %IP% -p tcp --dport 22 -j ACCEPT
```



Rysunek 2: Przykładowy plik konfiguracyjny `knockd.conf`.

```
Seq_Timeout = x
```

Procedura ta określa czas, w jakim cały proces „pukania” MUSI zostać zakończony. Czas podajemy w sekundach.

```
TCPFlags = x
```

Podczas wyłapywania połączeń KnockD będzie zwracał uwagę tylko na pakiety oznaczone wypisaną tutaj flagą. Do wyboru mamy: `fin`, `syn`, `rst`, `psh`, `ack`, `urg`. Istnieje także możliwość wpisania paru typów flag, ich ilość nie jest ograniczona. My użyjemy flag `syn`

```
command = x
```

Tutaj za `x` podstawiamy komendę, która ma zostać wykonana (chcemy otworzyć zamknięty port 22 – SSH).

Jeżeli chcemy stworzyć bardziej rozbudowane zasady (inaczej mówiąc, trudniejsze do „przypadkowego” natrafienia), powinniśmy w pełni skorzystać z procedury `Sequence`, np.

```
Sequence = 2
8000:tcp,8500:udp,9000:udp,9001:tcp
```

### Dwie pieczenie na jednym ogniu

W przykładzie z Listingu 1, aby z powrotem zamknąć port, trzeba wystukać kombinację zawartą w `[ssh_close]`. Istnieje jednak możliwość zautomatyzowania zamykania wcześniej otwartego portu. Proszę spojrzeć na Listing 2.

Jak widać, użyliśmy tu trzech nowych procedur: `start_command` (komenda tutaj wpisana zostaje wykonana po wystukaniu odpowiedniej sekwencji), `cmd_timeout` (wartość podawana w sekundach; oznacza ona, ile ma minąć od wykonania `start_command` do `stop_command`), `stop_command` (komenda, która zostanie wykonana po upływie `cmd_timeout` od czasu wykonania `start_command`).

### Listing 2: Automatyzacja zamykania portów

```
sequence = 7001,7002,7003,7004
seq_timeout = 10
tcpflags = syn
start_command = /usr/sbin/iptables -A INPUT -s %IP% -p tcp --syn -j ACCEPT
cmd_timeout = 45
stop_command = /usr/sbin/iptables -D INPUT -s %IP% -p tcp --syn -j ACCEPT
```

Obydwie konfiguracje do zamykania i otwierania portów używają czystego `IPTables`. Komendy otwórz/zamknij różnią się tylko 1 parametrem: `-A` lub `-D` (`A` od *allow* oraz `D` od *deny*). Zmienna `%IP%` oznacza IP maszyny „pukającej”.

### Bezpieczeństwo

Z bezpieczeństwem KnockD może być różnie. Patrząc na to kątem oka, możemy stwierdzić, iż w najgorszej dla administratora sytuacji włamywacz po prostu po wystukaniu odpowiedniej kombinacji otworzy port SSH (lub inny). Jednak może dojść do pewnych *niepożądanych* sytuacji. Włamywacz po wykonaniu np. `BO[2]` (ang. *buffer overflow* – przepełnienie bufora) lub wykorzystaniu złośliwego „format string” [3] albo jeszcze paru innych *nietaktownych* metod może dostać shella z wysokimi uprawnieniami. Ale na pocieszenie można dodać, iż nie powinno to nikogo zniechęcać (zwłaszcza prywatnych użytkowników). Wykonanie ataku typu `BO` lub „format string”, pukając do wielu przypadkowych portów, jest zadaniem praktycznie niemożliwym (nie mówiąc teoretycznie niemożliwym, ponieważ to byłoby już stwierdzenie za daleko idące). Oczywiście, błąd może zostać także znaleziony w samej aplikacji, w kodzie źródłowym, ale na to jesteśmy narażeni ciągle. A więc wybór należy do Was. ■

### INFO

[1] Strona projektu KnockD:

<http://zeroflux.org/knock/>

[2] Opis zagadnienia BO (7 opisów):

<http://t-nas.org/index.php?sekcja=strona&id=354>

[3] Opis zagadnienia „format string”:

<http://cc-team.org/index.php?name=artykuly&show=73>