1 Pakiet SCAPY

Pakiet Scapy stanowi rozszerzenie języka Python, służące do operacji na pakietach i ramkach sieciowych. Poniżej przedstawiono skrótowy opis wybranych jego poleceń, przeznaczony jako pomoc w realizacji laboratorium dotyczącego bezpieczeństwa mechanizmów routingu dynamicznego.
Uwagi ogólne:

- Przed rozpoczęciem pracy należy sprawdzić stan interesujących nas interfejsów sieciowych (patrz 1.1), tak aby były widoczne w pakiecie Scapy
- Pracę rozpoczynamy poleceniem wiersza poleceń: **scapy**
- Rozróżniane są małe i duże litery.
- Pracę poleceń przerywamy kombinacją klawiszy Ctrl+C.
- Pracę z pakietem Scapy kończymy poleceniem **exit()** lub **quit()**.

1.1 Wyświetlenie listy interfejsów sieciowych

Listę dostępnych interfejsów sieciowych można uzyskać poleceniem:
show_interfaces()
Spowoduje ono wyświetlenie wyłącznie interfejsów włączonych (enabled) i posiadających połącznie fizyczne. Nie będą widoczne interfejsy wyłączone (disabled) oraz te z odłączonym kablem lub niepodłączone do żadnej sieci bezprzewodowej.
Jeśli włączymy interfejs w trakcie pracy pakietu Scapy, to aby stały się one dostępne, należy zakończyć pracę pakietu (poleceniem **exit()**) i ponownie go uruchomić.
UWAGA: W pakiecie Scapy zawsze używamy nazw interfejsów sieciowych podanych przez to polecenie, choć są one najczęściej różne od nazw używanych w systemie operacyjnym.

1.2 Definiowanie struktury pakietu

Pakiety definiujemy podając po kolei protokoły w nich występujące, rozdzielając je znakiem „/”. Np. pakiet protokołu UDP przesyłany siecią Ethernet, będzie zapisany jako:
Ether()/IP()/UDP()
jako że kolejne występujące w nim nagłówki to nagłówek sieci Ethernet, poprzedzający nagłówek protokołu IP, po którym następuje nagłówek protokołu UDP.
W powyższym przypadku wartości pól nagłówków przyjmą wartości domyślne (z niewielkimi wyjątkami o których poniżej), co można odczytać opisanym dalej poleceniem **ls(*<pakiet>*)**. Polecenie:
ls(Ether()/IP()/UDP())
spowoduje wyświetlenie następujących informacji:

```
>>> ls(Ether()/IP()/UDP())
dst        : DestMACField         = 'ff:ff:ff:ff:ff:ff' (None)
src        : SourceMACField       = '00:00:00:00:00:00' (None)
type       : XShortEnumField      = 2048            (0)
--
version    : BitField             = 4               (4)
ihl        : BitField             = None            (None)
tos        : XByteField           = 0               (0)
len        : ShortField           = None            (None)
id         : ShortField           = 1               (1)
flags      : FlagsField           = 0               (0)
frag       : BitField             = 0               (0)
ttl        : ByteField            = 64              (64)
proto      : ByteEnumField        = 17              (0)
chksum     : XShortField          = None            (None)
src        : Emph                 = '127.0.0.1'     (None)
dst        : Emph                 = '127.0.0.1'     ('127.0.0.1')
options    : PacketListField      = []              ([])
--
sport      : ShortEnumField       = 53              (53)
dport      : ShortEnumField       = 53              (53)
len        : ShortField           = None            (None)
chksum     : XShortField          = None            (None)
```

Jak widać wyświetlone są tu wszystkie pola występujące w zdefiniowanej przez nas strukturze pakietu. W kolumnie po znaku „=” podane są konkretne wartości tych pól, a w kolumnie ostatniej – domyślne wartości tych pól.
Można, zauważyć, że w niektórych przypadkach wartość pola różni się od domyślnej (np. pole ***proto*** nagłówka IP). Różnice te wynikają z mechanizmów automatyzacji tworzenia pakietu – pole ***proto*** przyjęło wartość 17, gdyż dalej nakazaliśmy umieszczenie protokołu UDP, który w poprawnym nagłówku protokołu IP identyfikowany jest właśnie powyższą wartością wzmiankowanego pola.

Jeśli chcemy stworzyć pakiet w którym zmodyfikowaliśmy wartości niektórych jego składowych, należy podać odpowiednie definicje w nawiasach następujących po nazwie konkretnych protokołów składowych, rozdzielając kolejne przypisania przecinkami. Wartości:

- liczbowe podajemy bez cudzysłowów, np.: version=4
- tekstowe/binarne – w cudzysłowach, np.: dst=’10.10.1.1’

Np. jeśli w powyższym pakiecie zamierzamy ustawić źródłowy adres IP na wartość „192.168.1.1”, docelowy adres IP na „192.168.1.20”, a wartość TTL na 15, to zapisujemy to następująco:
Ether()/IP(dst=’192.168.1.1’,dst=’192.168.1.20’,ttl=15)/UDP(dport=123)
W efekcie otrzymamy:

```
>>> ls(Ether()/IP(src='192.168.1.1',dst='192.168.1.20',ttl=15)/UDP())
dst        : DestMACField         = '00:0c:42:a7:0f:2e' (None)
src        : SourceMACField       = '90:e2:ba:26:fc:f7' (None)
type       : XShortEnumField      = 2048            (0)
--
version    : BitField             = 4               (4)
ihl        : BitField             = None            (None)
tos        : XByteField           = 0               (0)
len        : ShortField           = None            (None)
id         : ShortField           = 1               (1)
flags      : FlagsField           = 0               (0)
frag       : BitField             = 0               (0)
ttl        : ByteField            = 15              (64)
proto      : ByteEnumField        = 17              (0)
chksum     : XShortField          = None            (None)
src        : Emph                 = '192.168.1.1'   (None)
dst        : Emph                 = '192.168.1.20'  ('127.0.0.1')
options    : PacketListField      = []              ([])
--
sport      : ShortEnumField       = 53              (53)
dport      : ShortEnumField       = 123             (53)
len        : ShortField           = None            (None)
chksum     : XShortField          = None            (None)
```

Jak widać odpowiednie pola nagłówka IP przyjęły podane wartości (różne od domyślnych). Warto też zauważyć działanie, wspomnianych wcześniej, mechanizmów automatyzacji, które spowodowały w tym przypadku automatyczne wypełnienie pól ***src*** i ***dst*** w nagłówku ramki Ethernet.

Tworzony w ten sposób pakiet możemy umieścić w zmiennej o wybranej nazwie, w calu dalszego jego wykorzystania, np.:
pakiet=Ether()/IP(dst=’192.168.1.1’,dst=’192.168.1.20’,ttl=15)/UDP()
spowoduje umieszczenie omawianego powyżej pakietu w zmiennej o nazwie ***pakiet***.

W celu wyświetlenia zawartości zmiennej możemy użyć poleceń:

- **ls(<*nazwa zmiennej*>)** – np. **ls(pakiet)** – lista wszystkich pól, ich wartości aktualnych i domyślnych.
- **<*nazwa zmiennej*>.show()** – np. **pakiet.show()** – lista wszystkich pól wraz z wartościami, w postaci „pseudo-graficznej”.

- ***<nazwa zmiennej>*** - np. **pakiet** – uproszczona struktura pakietu, zawierająca wszystkie protokoły składowe oraz pola o wartościach ręcznie zdefiniowanych przez użytkownika. Pola z wartościami domyślnymi lub ustawionymi automatycznie nie są wyświetlane. Jeśli pakiet został otrzymany w wyniku działania polecenia sniff (patrz 1.4), to wszystkie pola są wyświetlane.

Wartości pól pakietu zawartego w zmiennej o podanej nazwie możemy modyfikować. W tym celu podajemy nazwę zmiennej, a następnie, w nawiasach kwadratowych, nazwę protokołu (np. [UDP]) oraz poprzedzoną znakiem „**.**" (kropką), nazwę pola. Np. w przypadku opisywanego powyżej pakietu:

- **pakiet[Ether].src='00:11:22:33:44:55'** – zmiana adresu źródłowego w nagłówku Ethernet,
- **pakiet[IP].src='10.10.1.1'** – zmiana adresu źródłowego w nagłówku IP.

W celu ustalenia nazw pól pakietu, przydatne jest polecenie **ls(*<zmienna>*) lub ls(*<nazwa protokołu>*)**.

1.3 Przechwytywanie pakietów z interfejsu sieciowego

Polecenie **sniff()** umożliwia przechwytywanie pakietów z interfejsu sieciowego. Wywołane bez parametrów spowoduje przechwytywanie pakietów z pierwszego z dostępnych interfejsów sieciowych (wg. listy wyświetlanej poleceniem **show_interfaces()**), kontynuowanego do momentu przerwania go poprzez naciśnięcie klawiszy **Ctrl+C**.
Ewentualne parametry polecenia podajemy w nawiasie, oddzielane przecinkami.
Interesujące nas parametry to:

- **iface='*<nazwa interfejsu>*'** – nakazuje nasłuchiwać na podanym interfejsie sieciowym.
- **filter='*<filtr>*'** – umożliwia ograniczenie przechwytywanych pakietów do zgodnych z podanym filtrem. Filtr definiujemy zgodnie ze standardowym formatem libpcap (patrz 1.5).
- **count=*<liczba pakietów>*** – powoduje przechwycenie podanej liczby pakietów i zakończenie działania polecenia.
- **timeout=*<czas w s.>*** – powoduje zakończenie działania polecenia po upływie określonego czasu.

Przykładowo:
sniff(iface='eth1',filter='udp',count=10)
spowoduje przechwycenie 10 pakietów UDP z interfejsu eth1 i zakończenie działania polecenia sniff.
Przechwycone pakiety można zachować umieszczając je w zmiennej.
pakiety=sniff(iface='eth1',count=10)

Zmienna przybierze w takim przypadku postać listy pakietów, której kolejne pozycje możemy odczytać podając, w nawiasach kwadratowych, ich indeks na liście w postaci kolejnego numeru poczynając od 0 – np.: **ls(pakiety[0])**.

```
>>> pakiety=sniff(filter='udp', count=5)
>>> ls(pakiety[0])
dst        : DestMACField         = 'ff:ff:ff:ff:ff:ff' (None)
src        : SourceMACField       = '00:15:5d:01:0a:22' (None)
type       : XShortEnumField      = 2048            (0)
--
version    : BitField             = 4L              (4)
ihl        : BitField             = 5L              (None)
tos        : XByteField           = 0               (0)
…
```

Należy zwrócić uwagę, że nie wszystkie polecenia przyjmujące jako parametr pakiet, przyjmą jako parametr listę pakietów – dobrym przykładem jest tu polecenie **ls(*<pakiet>*)**: polecenie ls(pakiety) nie jest prawidłowe, podczas gdy polecenie ls(pakiety[2]) – jest.

Opisane wcześniej polecenia służące do wyświetlania zawartości pakietu, mogą zostać również wykorzystane do wyświetlania zawartości listy pakietów (np. **pakiety.show()**, **pakiety**). Wyjątek stanowi polecenie ls, którego parametrem może być wyłącznie pojedynczy pakiet (np. **ls(pakiety[3]**).

1.4 Wysyłanie pakietów

Polecenie **sendp()** służy do wysłania przygotowanych wcześniej pakietów korzystając z procedur warstwy 2 ISO-OSI (łącza danych). Oznacza to, że mamy stosunkowo dużą kontrolę nad procesem wysyłania – możemy np. wskazać interfejs, przez który ma zostać wysłany dany pakiet (co byłoby niemożliwe przy wysyłaniu w pakietu korzystając z procedur warstwy 3, sieciowej, gdzie określone zostało by to automatycznie). Oznacza to jednak, iż wysyłany pakiet musi zawierać protokół warstwy 2 ISO-OSI, np. w przypadku sieci Ethernet – **Ether()**.
Pierwszy parametr z poniższej listy jest wymagany, gdyż określa pakiet/pakiety do wysłania. Pozostałe są opcjonalne i mogą występować w dowolnej kolejności. Np.: **sendp(x,count=2, iface='eth1')**

- x – pakiet lub lista pakietów (patrz 1.3) do wysłania,
- iface='<interfejs>' – określa nazwę interfejsu (wg. polecenia show_interfaces()) przez który wysyłamy pakiet,
- inter=<czas w s.> – określa, w sekundach, okres pomiędzy wysyłaniem kolejnych pakietów,
- count=<liczba> – określa ile razy zostaną wysłane pakiety podane jako parametr x. Jeśli x stanowi listę 5 pakietów, a count=2, to zostanie wysłanych 10 pakietów (5 pakietów z x, a następnie ponownie 5 pakietów z x).

1.5 Filtry libpcap

W przypadku niektórych poleceń pakietu Scapy możemy posłużyć się filtrami pakietów w formacie zgodnym ze zdefiniowanym przez bibliotekę libpcap – dotyczy to głównie poleceń służących przechwytywaniu ruchu sieciowego, jak np. sniff() (patrz 1.3).
Składnia filtrów którymi możemy się posłużyć jest rozbudowana, więc poniżej podano wyłącznie wybrane przykłady.

- Pakiety danego protokołu: **ip**, **ip6**, **icmp**, **icmp6**, **tcp**, **udp**, **ah**, **esp**, **ip proto *<nr protokołu ip>***
- Filtracja wg portów TCP i/lub UPD:
  - **port *<nr>*** - pakiety TCP lub UDP (bo tylko one posiadają porty), o porcie źródłowym lub odcelowym o wartości <nr>
  - **sport *<nr>*** - jak wyżej, lecz o porcie źródłowym o wartości <nr>
  - **dport *<nr>*** - jak wyżej, lecz o porcie docelowym o wartości <nr>
  - **tcp port *<nr>***, **udp port *<nr>***, **tcp sport *<nr>*** itp. – jak wyżej, lecz dla określonego protokołu.
- Filtracja wg adresów IP:
  - **host *<adres IP>*** - pakiety o źródłowym lub docelowym adresie IP równym <adres IP>,
  - **src host *<adres IP>*** - pakiety o źródłowym lub docelowym adresie IP równym <adres IP>,
  - **dst host *<adres IP>*** - pakiety o źródłowym lub docelowym adresie IP równym <adres IP>.

2 Wymagania dotyczące tworzenia pakietów protokołu RIP

Pakiet protokołu RIP przesyłany przez sieć Ethernet składa się z następujących elementów składowych:

1. **Ether()** – ramka sieci Ethernet, przenosząca:

2. **IP()** – pakiet protokołu IP, zwierający:
3. **UDP()** – nagłówek protokołu UDP, w którego polu danych przenoszony jest:
4. **RIP()** – protokół RIP (nagłówek ogólny).

Po nagłówku protokołu RIP, występują elementy:

5. **RIPAuth()** – opcjonalnie, jeśli używany mechanizmów uwierzytelniania RIP,
6. jeden lub więcej element **RIPEntry()**, zawierający konkretne informacje wymieniane przez protokół RIP (np. trasy).
7. RIPAuth() – opcjonalnie, jeśli używamy mechanizmu uwierzytelniania MD5.

Tak więc najprostsza wiadomość protokołu RIP to:
Ether()/IP()/UDP()/RIP()/RIPEntry()
z uwierzytelnianiem prostym hasłem:
Ether()/IP()/UDP()/RIP()/RIPAuth()/RIPEntry()
z uwierzytelnianiem MD5:
Ether()/IP()/UDP()/RIP()/RIPAuth()/RIPEntry()/RIPAuth()

Oczywiście powyższy zapis zawiera domyślne wartości rozmaitych pól i nie jest dla nas, w tej postaci, przydatny w praktyce.
W celu wygenerowania poprawnej wiadomości RIP niewykorzystującej uwierzytelnia MD5, należy ustawić poprawne wartości następujących pól:

1. **Ether**
   - brak, generowane w pełni automatycznie.
2. **IP**
   - **src** – adres źródłowy IP (nadawca wiadomości). Musi należeć do tej samej sieci IP co adres przypisany do tego z interfejsów routera, który odbierze wiadomość.
   - **dst** – adres docelowy IP (odbiorca wiadomości). W przypadku protokołu RIP, powinien być ustawiony na adres multicastowy 224.0.0.9 (na którym nasłuchują routery obsługujące protokół RIP) lub unicastowy adres IP konkretnego routera.
3. **UDP**
   - **sport** – źródłowy port UDP należy ręcznie ustawić na wartość > 1024.
   - **dport** – docelowy port UDP to w przypadku protokołu RIP port 520.
4. **RIP**
   - **version** – wersja protokołu RIP (1 lub 2) używana przez router.
   - **cmd** – określa czy wysyłamy żądanie (request – wartość 1) czy aktualizację (update/response – wartość 2). RIP Request powoduje przesłanie przez router docelowy całej posiadanej informacji routingowej. RIP Update pozwala poinformować router docelowy o nowych trasach lub zaktualizować istniejące.
5. **RIPAuth** (wymagane tylko, jeśli używamy uwierzytelniania z użyciem prostego hasła)
   - **authtype** – rodzaj uwierzytelniania: w tym przypadku proste hasło (wartość 2).
   - **password** – treść hasła.
6. **RIPEntry** (używane tylko w przypadku wiadomości RIP Update)
   - **addr** – adres sieci, której dotyczy wysyłana przez nas wiadomość.
   - **mask** – maska sieci, której dotyczy wysyłana przez nas wiadomość.
   - **metric** – liczba przeskoków, w której (licząc od routera ODBIERAJĄCEGO wiadomość) znajduje się sieć docelowa podana w polach **addr** i **mask**.

3 Śledzenie wydarzeń protokołu RIP

W przypadku routerów Cisco możliwe jest szczegółowe śledzenie wydarzeń dotyczących mechanizmów protokołu RIP.

Mechanizm uruchamiany jest poleceniem **debug ip rip**. Skutkiem jest wypisywanie odpowiednich komunikatów na konsolę szeregową routera.

Wypisywanie komunikatów wyłączamy poleceniem **no debug ip rip**.

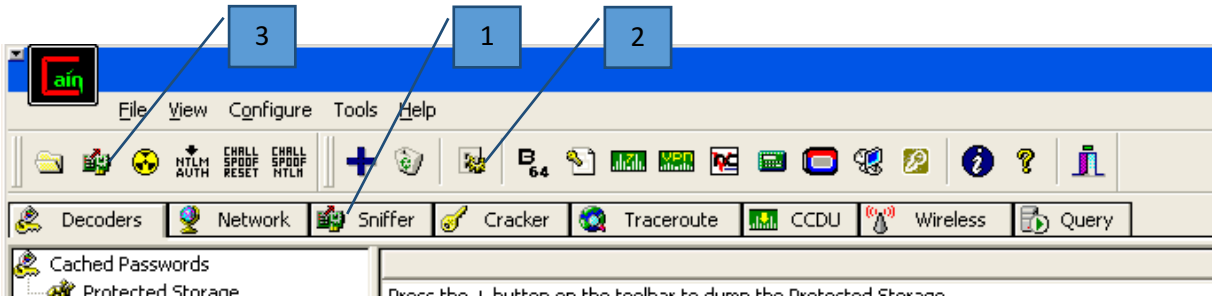
UWAGA: Pojawiające się komunikaty nie przeszkadzają we wpisywaniu nowych poleceń. Ponadto komunikaty pojawiają się wyłącznie na konsoli szeregowej – nie pojawiają się w przypadku połączeń z routerem protokołem telnet.

Przykłady komunikatów to:

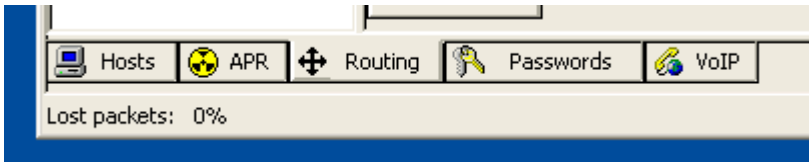
1. **RIP: received v2 request from 10.10.1.17 on FastEthernet0/0/1**
   (bez dalszego ciągu – patrz komunikat 3)
   - Znaczenie: z adresu 10.10.1.17 podłączonego do interfejsu FastEthernet0/0/1 otrzymano widomość RIPv2 zawierającą niepoprawny nextHop.
2. **RIP: received v2 request from 10.10.1.17 on FastEthernet0/0/1**
   RIP: ignore the request received from unlisted network.
   - Znaczenie: z adresu 10.10.1.17 podłączonego do interfejsu FastEthernet0/0/1 otrzymano widomość RIPv2, która zostanie zignorowana, gdyż nadawca wiadomości nie należy do sieci IP, z której adres router posiada na tym interfejsie – tzn. IP nadawcy jest w innej sieci IP niż router.
3. **RIP: received v2 update from 204.0.0.10 on FastEthernet0/0/1**
   210.0.0.0/24 via 204.0.0.123 in 2 hops
   - Znaczenie: w wyniku otrzymania wiadomości RIPv2 od nadawcy pod adresem 204.0.0.10, podłączonego do interfejsu FastEthernet0/0/1 routera, dodano trasę do sieci 210.0.0.0/24 prowadzącą przez router o adresie 203.0.0.123 i odległą od router o 2 przeskoki.

4 Wykorzystanie programu Cain do ustalenia hasła przy uwierzytelnianiu RIP MD5

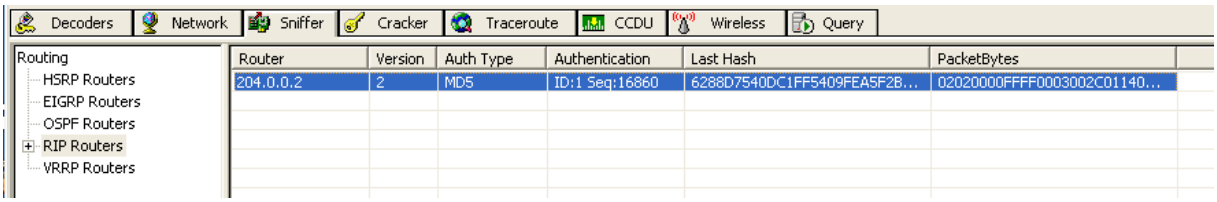
1. Uruchamiamy program Cain. Wybieramy zakładkę „**Sniffer**” (1). Jeśli lista adresów widoczna w zakładce „**Sniffer**” nie jest pusta, to z menu File wybieramy opcję **Remove All**.
2. Klikamy ikonę (2) i wybieramy interfejs, z którego chcemy podsłuchiwać ruch. Interfejsy najłatwiej rozpoznać po przypisanych im adresach IP.
3. Rozpoczynamy przechwytywanie ruchu włączając ikonę (3).



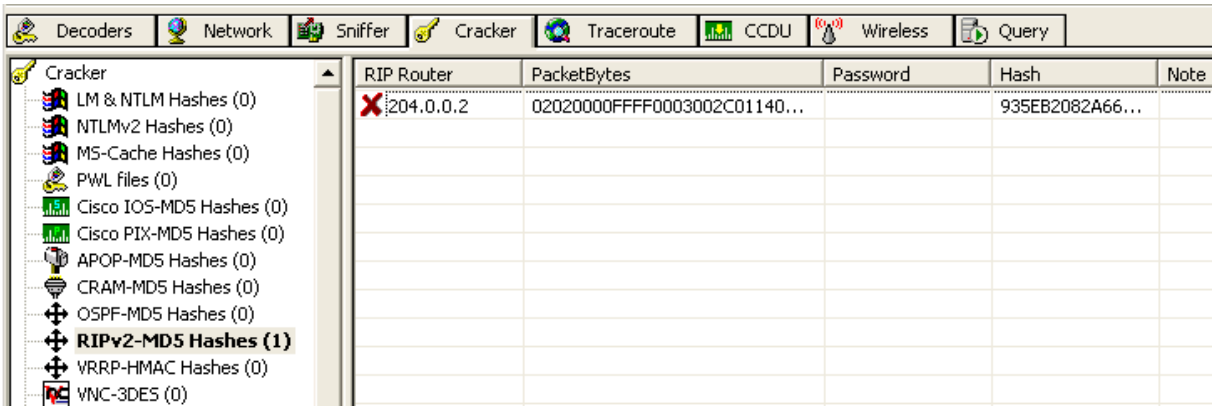
4. Wybieramy zakładkę „**Routing**” zlokalizowaną w dolnej części okna programu. Jeśli nie ma tam tej zakładki, upewniamy się czy spośród zakładek umieszczonych w górnej części okna programu mamy wybraną zakładkę „**Sniffer**”.



5. Z listy protokołów routingu dostępnej w lewej części okna programu, wybieramy „**RIP Routers**”. W głównej części okna programu pojawi się lista wykrytych routerów RIP. Jeśli któryś z nich wykorzystuje proste hasło do uwierzytelniania wiadomości, to zostanie ono podane. Jeśli router wykorzystuje mechanizm MD5, to podany zostanie wartość skrótu MD5 dla ostatnio odebranej wiadomości.



6. Nacisnąć **prawym przyciskiem** myszy na interesującym nas routerze i z pojawiającego się menu wybrać „**Send to Cracker**”.
7. Przejść do zakładki „**Cracker**” zlokalizowanej w górnej części okna programu.



8. Z listy po lewej stronie okna wybrać „**RIPv2-MD5 Hashes**”. Spowoduje to pojawienie się listy znanych programowi skrótów MD5. Jeśli hasło użyte do wygenerowania któregoś z nich jest już znane, to zostanie podane w odpowiedniej kolumnie. Przesłany przez nas (w kroku 6) skrót MD5 również powinien być tu widoczny.
9. Przyciskamy **prawy przycisk** myszy na interesującym nas skrócie MD5 na liście i wybieramy opcję „**Brute Force**”.
10. W pojawiającym się oknie ustawiamy wg. własnego uznania opcje ataku Brute Force i uruchamiamy go.
11. Zwracamy uwagę na informację o szacowanym czasie pozostałym do ustalenia hasła – jeśli nas nie zadowala to przerywamy atak i zmieniamy odpowiednio opcje.
12. Po ustaleniu hasła zamykamy okno ataku – ustalone hasło powinno być widoczne na liście.