Paweł Pokrywka

Testowanie zabezpieczeń sieci lokalnych pod względem nieupoważnionego dostępu na poziomie warstw 2-3 OSI.

Pokrywka, Paweł:

Testowanie zabezpieczeń sieci lokalnych pod względem nieupoważnionego dostępu na poziomie warstw 2-3 OSI.

Data wydruku: 20 lipca 2005

Spis treści

Wstęp

Podstawę funkcjonowania każdego społeczeństwa stanowi komunikacja we wszelkich odmianach. Najważniejszą i najpopularniejszą jej formą jest mowa i związany z nią język ludzki, będący jedną z pierwszych umiejętności, które zdobywa jednostka.

Wraz z rozwojem cywilizacyjnym komunikacja ewoluowała, co objawiało się w szczególności rozwojem metod przekazywania informacji na odległość. Plemiona afrykańskie używały w tym celu sygnałów dźwiękowych wydawanych przez potężne bębny, Indianie korzystali z systemu znaków dymnych, zaś Rzymianie za pomocą umieszczonych na wzgórzach luster przesyłali impulsy świetlne. W 1838 roku został zbudowany telegraf elektryczny wykorzystujący alfabet Morse'a[1], co stało się podwaliną współczesnej telekomunikacji.

Obecnie informacje przesyłane są przez nowoczesne trakty cyfrowe, które pozwalają na szybką transmisję danych na duże odległości. Systemy komputerowe, które jeszcze kilkadziesiąt lat temu z reguły pracowały w izolacji, dziś łączone są w sieci umożliwiające współdzielenie zasobów. Telekomunikacja i informatyka stały się wzajemnie zależne, co zaowocowało powstaniem *teleinformatyki*, czyli nauki łączącej te dziedziny wiedzy.

Tematem niniejszej pracy są lokalne sieci komputerowe oraz ich bezpieczeństwo, ze szczególnym naciskiem na problem uwierzytelniania. Program komputerowy, który powstał w jej ramach, pozwala na demonstrację słabości najpopularniejszej metody zabezpieczania tego typu sieci przed nieupoważnionym dostępem.

Praca została podzielona na rozdziały. Podział ten przedstawia się następująco:

Rozdział 1 ma za zadanie wprowadzić czytelnika w tematykę tzw. sieci osiedlowych oraz problemy bezpieczeństwa w nich występujące.

Rozdział 2 zawiera przegląd technologii sieciowych wykorzystywanych w aplikacji *multispoof*, która powstała w ramach pracy.

Rozdział 3 przedstawia koncepcję aplikacji *multispoof*, a następnie koncentruje się na architekturze programu oraz problemach implementacyjnych.

Rozdział 4 zawiera analizę zachowania aplikacji podczas działania, wraz z wynikami przeprowadzonych testów.

Rozdział 5 stanowi opis metod wykrywania nieupoważnionego dostępu do sieci oraz środków, które pozwalają ją przed tym zabezpieczyć.

Rozdział 6 przedstawia kierunki dalszego rozwoju aplikacji, które dotyczą utrudniania detekcji, przełamywania innych typów zabezpieczeń oraz zwiększania wydajności i wygody użytkowania.

Dodatek A zawiera dwa listingi: skryptu *multispoof* oraz skryptu używanego w testach.

Dodatek B stanowi zbiór informacji niezbędnych w procesie kompilacji oraz instalacji programu *multispoof*. W dodatku zawarta jest także instrukcja obsługi, oraz analiza przykładowej sesji aplikacji.

[1] Pierwszy telegraf elektryczny wymagający pięciu przewodów skonstruował anglik Sir Charles Wheatstone rok wcześniej. Jednak dopiero urządzenie Samuela Morse'a, wykorzystujące opracowany przez niego alfabet, zyskało popularność, ponieważ do komunikacji wymagany był tylko jeden przewód.

Konwencje i oznaczenia

W celu ułatwienia czytania zostały wprowadzone konwencje typograficzne. Narzędzia, biblioteki i pakiety oprogramowania pisane są `czcionką o stałej szerokości`, zaś komponenty aplikacji *multispoof* są dodatkowo **pogrubione**. Czcionka o `stałej szerokości` jest używana także w listingach oraz sesjach powłoki systemowej. Te fragmenty, na które należy zwrócić szczególną uwagę lub stanowią komendy wpisywane przez użytkownika, są **pogrubione**. Zbiór oznaczeń graficznych, używany jest konsekwentnie na większości rysunków. Adresy URL, z których można pobrać wykorzystywane narzędzia podawane są w przypisach, najczęściej z krótką informacją o funkcjonalności danego programu.

Podziękowania

Praca niniejsza powstała w dużej mierze dzięki pomocy wielu życzliwych osób, którym należą się słowa podziękowania.

W pierwszej kolejności chcę wyrazić wdzięczność dr Tomaszowi Surmaczowi za owocną współpracę.

Pragnę serdecznie podziękować moim rodzicom, Krystynie i Janowi Pokrywkom za wszelką pomoc, której nie jestem w stanie opisać, ani wyliczyć.

Za pomoc przy rozwiązywaniu problemów podczas pisania aplikacji, dziękuję Michałowi Pokrywce. Za cierpliwość, cenne dyskusje, wskazówki i pożyteczne uwagi chcę podziękować Krzysztofowi Noskowi, Przemysławowi Plata oraz moim braciom Piotrowi i Przemysławowi Pokrywkom. Chcę także wyrazić wdzięczność wszystkim osobom, tworzącym oprogramowanie, które wykorzystywałem podczas pisania niniejszej pracy oraz aplikacji *multispoof*.

I would like to thank especially Guy Harris from tcpdump project for help with libpcap patch inclusion, and John Bothe for initial patch code.

"The enemy knows the system being used."

– Claude Shannon

Rozdział 1

Problem uwierzytelniania w sieciach LAN

W tym rozdziale zostanie przedstawiona koncepcja sieci LAN, zagrożenia jakie w nich występują oraz praktyczne problemy, związane z ich wykorzystaniem przez intruzów. Główny nacisk zostanie położony na omówienie popularnego sposobu uwierzytelniania użytkowników sieci, którego słabości będą zademonstrowane praktycznie w następnych rozdziałach.

1.1 Obszar zainteresowania

Ze względu na fakt, że tematyka sieci lokalnych to szeroki zakres dziedziny IT, jaką są sieci komputerowe, należy uściślić obszar zainteresowania tej pracy. Są nim lokalne sieci komputerowe, zbudowane z wykorzystaniem technologii Ethernet [Net99] lub pochodnych i wykorzystywane do płatnego świadczenia usługi dostępu do Internetu. Dotyczy to tzw. sieci osiedlowych, zarówno amatorskich (non-profit) jak i profesjonalnych. Te pierwsze są zarządzane przez osoby, które nie są zainteresowane zyskiem, zaś opłaty pobierane od korzystających z sieci są w całości przeznaczane na zwrot kosztów jej utrzymania. W przeciwieństwie do nich, sieci profesjonalne są własnością firm (nazywanych operatorami lub dostawcami usługi dostępu do Internetu), które opierają swoją działalność na przychodach uzyskiwanych z abonamentu.

Głównym powodem tego, że sieci osiedlowe istnieją i rozwijają się, jest niska cena infrastruktury ethernetowej w zakresie instalacji i utrzymania. Urządzenia wchodzące w skład takiej sieci są łatwo dostępne i tanie, w przeciwieństwie do sprzętu niezbędnego w innych rozwiązaniach szerokopasmowych, jak technologia DSL[1] czy telewizja kablowa[2]. Na infrastrukturę osiedlowego Ethernetu składają się najczęściej tylko trzy rodzaje elementów: przełączniki sieciowe, kable miedziane lub światłowody oraz karty sieciowe w komputerach. Dostęp do Internetu zapewnia router, który w mniejszych sieciach bywa implementowany na zwykłym, niedrogim komputerze klasy PC.

1.2 Uwierzytelnianie użytkowników

Dostęp do Internetu jest udzielany użytkownikom, którzy uregulowali należności finansowe. Natomiast klienci, którzy zalegają z opłatami powinni tracić dostęp do usługi, najlepiej w dniu przekroczenia terminu płatności. Najpewniejszą metodą odcięcia użytkownika od sieci jest wyłączenie z najbliższego przełącznika kabla, który do niego prowadzi. Jest to jednak kłopotliwe, ponieważ wymaga wykonania pracy fizycznej, zaś przełączniki sieciowe często instalowane są w miejscach trudno dostępnych, jak piwnice, strychy, słupy lub mieszkania osób prywatnych. Dodatkowo, niektórzy dostawcy sieci osiedlowych zezwalają także na

[1] DSL (ang. *Digital Subscriber Line*) to sposób transmisji danych z wykorzystaniem jednej pary miedzianej. W zależności od wariantu, technologia pozwala na osiągnięcie nawet dziesiątek megabitów na sekundę [Net99].

[2] Transmisję danych cyfrowych z wykorzystaniem telewizji kablowej określają standardy DOCSIS (ang. *Data Over Cable Service Interface Specification*).

bezpłatny lub obciążony mniejszą opłatą dostęp do zasobów sieci lokalnej, pod warunkiem niekorzystania z Internetu.

Dlatego najczęstszą praktyką jest blokowanie komputera klienta na poziomie logicznym w tej sposób, że nie traci on dostępu do sieci LAN, natomiast łączność z Internetem jest zawieszana. W praktyce polega to na odfiltrowywaniu danych, płynących z komputera użytkownika w kierunku globalnej sieci i *vice versa*. Wymaga to jednak odpowiedzi na pytanie:

> Jak powiązać transmisje przepływające przez router operatora z człowiekiem, którego komputer je wygenerował[3]?

Odpowiedzią jest proces uwierzytelniania. Aby uzyskać dostęp do usługi, komputer użytkownika musi *przedstawić się*, wykorzystując do tego informacje uzgodnione wcześniej między jego właścicielem i operatorem sieci.

Jeśli hosty w sieci są uwierzytelnione, oznacza to, że jest możliwe ich rozróżnienie, co w efekcie pozwala na oferowanie spersonalizowanych usług przez dostawcę. Takimi usługami są na przykład możliwość wyboru przepustowości dostępu do Internetu przez klienta, blokowanie niepożądanych treści, dostęp do informacji billingowych, ogłoszenia i reklamy.

Najpopularniejsza metoda uwierzytelniania w sieciach osiedlowych wykorzystuje to, że każdy komputer posiada kartę sieciową, zaś każda karta ma przypisany przez producenta unikalny adres sprzętowy MAC (ang. *Medium Access Control*). Na jego podstawie serwer DHCP[4], stanowiący część infrastruktury sieciowej, przydziela mu adres IP, który jest wymagany do komunikacji. Ponieważ serwery DHCP najczęściej konfigurowane są w ten sposób, że dla danego adresu MAC przydzielają zawsze ten sam adres IP, zaś użytkownicy rzadko wymieniają karty sieciowe, adres ten może być wykorzystywany do uwierzytelniania.

Najczęściej router, który zapewnia dostęp do Internetu, jest konfigurowany tak, aby w procesie uwierzytelniania brały udział oba adresy: sprzętowy oraz IP. Dzieje się tak, ponieważ ten drugi adres użytkownik może w trywialny sposób zmienić, zaś weryfikacja zgodności obu adresów pozwala na wykrycie tego typu oszustwa. Mimo tego, autorowi znane są sieci osiedlowe, które nie stosują nawet tego, podstawowego zabezpieczenia.

1.3 Metody nadużyć w sieci LAN

Niski koszt sieci Ethernet, prostota jej instalacji oraz wysoka elastyczność użytkowania są niestety okupione podatnością na wiele ataków, obniżających jej bezpieczeństwo zarówno w sensie poufności i integralności transmisji, jak i dostępności. Wydaje się, że głównymi źródłami tych problemów są protokół ARP oraz technika dynamicznego przełączania ramek.

1.3.1 Podszywanie się

W sieciach Ethernet opartych na popularnych, tanich przełącznikach lub koncentratorach istnieje niebezpieczeństwo podszycia się poprzez zmianę adresu. Możliwe jest to zarówno w warstwie trzeciej modelu OSI, poprzez zmianę numeru IP[5], jak i w warstwie drugiej poprzez zmianę adresu MAC. W popularnych systemach operacyjnych modyfikacja tego drugiego jest równie prosta jak pierwszego. Jednak zmiana adresu MAC stanowi z pewnych względów wiedzę tajemną, dostępną tylko *wybranym*[6]. Rozpowszechniony jest natomiast pogląd, że adres MAC, tzw. adres sprzętowy jest zapisany na stałe w urządzeniu i nie ma możliwości wpływu na jego wartość poza wymianą karty sieciowej.

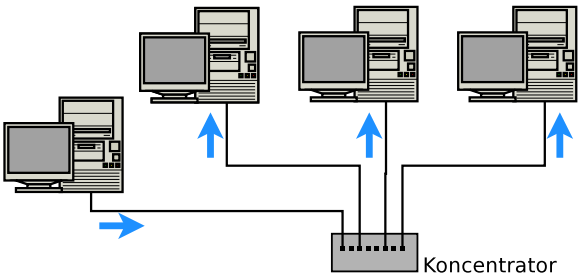
---

[3] Osobne zagadnienie stanowi fakt, że dane transmitowane i odbierane przez komputer nie zawsze są wynikiem woli użytkownika, który może np. nie być tego świadomym. Jednak na potrzeby dalszych rozważań przyjęte zostanie założenie, że jest on w pełni odpowiedzialny za aktywność sieciową posiadanego komputera.

[4] DHCP (ang. *Dynamic Host Configuration Protocol*) to protokół służący do automatycznego konfigurowania ustawień sieciowych hostów. Pozwala na scentralizowane zarządzanie adresami IP oraz innymi parametrami sieciowymi komputerów.

[5] Zmiana adresu IP określana jest jako IP Spoofing [Bellovin, 1989]. Atak ten jest stosowany także w sieciach rozległych [Shimomura, 1995], jednak ze względu na proces routowania, choć pakiety ze sfałszowanym adresem źródłowym docierają do celu, to atakujący nie otrzymuje na nie odpowiedzi. W sieciach ethernetowych ta niedogodność nie występuje, dlatego atak daje dużo większe możliwości.

[6] Najczęściej tymi *wybrańcami* są administratorzy, których sieci zazwyczaj nie są zabezpieczone przed kradzieżą usługi z wykorzystaniem zmiany adresu MAC. Nie należy się zatem dziwić, że nie są oni zainteresowani dzieleniem się tą wiedzą.



Rysunek 1.1: Przepływ ramek przez koncentrator sieciowy.

Adres MAC jest faktycznie zapisany w pamięci karty, ale zazwyczaj ta pamięć pozwala na zmianę jej zawartości. Prostszą metodą jest jednak programowa zmiana adresu na poziomie sterownika karty sieciowej w systemie operacyjnym. Wreszcie wysyłanie ramek ethernetowych ze sfałszowanym adresem źródłowym jest możliwe także w przestrzeni użytkownika za pomocą odpowiednich bibliotek i interfejsów udostępnianych przez jądro. Więcej informacji na temat generowania ramek z poziomu aplikacji zostało zamieszczonych w podrozdziale 2.2.

Zmiana adresu MAC na adres komputera innego użytkownika, który jest w danej chwili nieaktywny powoduje, że serwer DHCP przydziela jego adres IP i dostęp do Internetu odbywa się w cudzym imieniu. Jest to kradzież usługi, przed którą operator sieci osiedlowej powinien się zabezpieczyć.

1.3.2 Bierne podsłuchiwanie

Sieci oparte na koncentratorach są podatne na podsłuchiwanie, ponieważ urządzenia te przesyłają otrzymane ramki na wszystkie porty (rysunek 1.1). Pozwala to atakującemu na dostęp do informacji przesyłanych w obrębie segmentu sieci, do którego on należy. Narzędzia do podsłuchiwania popularnie nazywane są *snifferami*. Najbardziej znanym programem tego typu jest `tcpdump`[7]. Podsłuchiwanie jest techniką, która obecnie ma coraz mniejsze znaczenie, ze względu na wypieranie koncentratorów przez przełączniki. W sieciach przełączanych efekty biernego podsłuchiwania są znikome. Jednak podsłuchiwanie nadal jest wykorzystywane przez intruzów w połączeniu z innymi metodami. Więcej o tym można przeczytać w podrozdziale 2.1.

1.3.3 ARP Spoofing

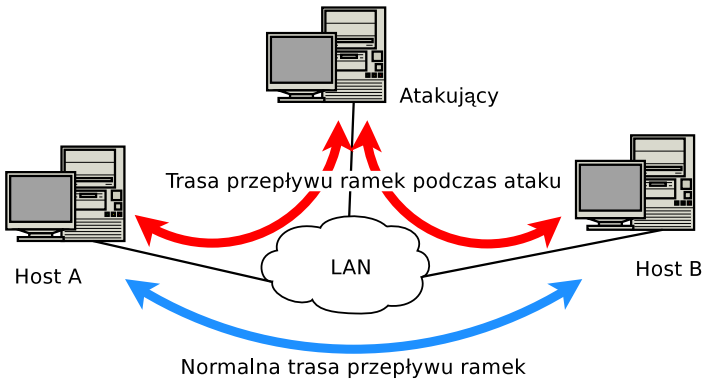
Jest to najbardziej znany typ ataku na sieci lokalne, wykorzystujący słabości protokołu ARP [Gusta i Szmit, 2003]. Protokół ten jest używany do mapowania numerów IP na adresy MAC, czyli łączy warstwę trzecią z warstwą drugą, umożliwiając tym samym komunikację w sieci fizycznej. Każdy host w sieci posiada lokalną tablicę ARP. Jeśli wystąpi potrzeba wysłania danych do innego hosta o znanym adresie IP w tym samym segmencie sieci lokalnej, tablica ta jest przeglądana w celu odnalezienia odpowiadającego mu adresu MAC. W przypadku, gdy tablica nie zawiera wpisu dla poszukiwanego adresu IP, system operacyjny wysyła do wszystkich komputerów w sieci lokalnej komunikat ARP Request, z zapytaniem o adres MAC tego hosta. Jeśli w danej chwili komputer o poszukiwanym adresie IP jest obecny w sieci, to wysyła komunikat ARP Reply do hosta, który wysłał zapytanie. Odpowiedź zawiera jego adres MAC, co pozwala na uaktualnienie tablicy ARP i dalszą transmisję.

Atak ARP Spoofing[8] polega na wprowadzaniu modyfikacji do tablic zdalnych hostów, wykorzystując do tego sfałszowane komunikaty ARP[9]. Na przykład atakujący wprowadza swój adres MAC w miejsce adresu hosta B w tablicy komputera A, oraz w miejsce adresu hosta A w tablicy komputera B. Dodatkowo komputer oszusta zmienia adres źródłowy ramek pochodzących od A na swój i przesyła je do B. Tak samo postępuje w przypadku ramek otrzymywanych od B. Komputery A i B mają wrażenie, że komunikują się bezpośrednio,

[7] Program `tcpdump` dostępny jest pod adresem `http://www.tcpdump.org/`.

[8] Metoda ta znana jest też pod nazwą ARP Poisoning.

[9] Istnieje bardzo wiele programów implementujących ten typ ataku, na przykład `arpspoof` z pakietu `dsniff`. Pakiet ten jest dostępny pod adresem `http://www.monkey.org/~dugsong/dsniff/`.



Rysunek 1.2: Koncepcja ataku ARP Spoofing.

ale w rzeczywistości cały ruch przechodzi przez komputer atakującego (rysunek 1.2). Daje to intruzowi następujące możliwości:

Podsłuchiwanie aktywne. Atak ARP Spoofing pozwala na przeprowadzanie podsłuchiwania w sieciach przełączanych.

Ingerencja w transmisje. Atakujący może modyfikować wybrane elementy ruchu, co nazywane jest *atakiem z wnętrza systemu* (ang. *Man in the Middle*).

Uniemożliwianie transmisji. Efekt ten wystąpi, jeśli atakujący zaprzestanie przesyłania ramek pomiędzy obydwoma hostami. Jeśli atak ten będzie dotyczył routera, to spowoduje on odmowę usługi dostępu do Internetu (ang. *Denial of Service*).

1.3.4 MAC Flooding

Sieć ethernetowa może zostać *zalana* (ang. *flooding*) ramkami generowanymi przez komputer atakującego[10]. W zależności od metody nadawania adresów źródłowych ramkom, można uzyskać dwa poniższe efekty:

- Jeśli ramki mają losowe adresy źródłowe, to tablice MAC przełączników po krótkim czasie ulegają przepełnieniu, co prowadzi do tego, że przechodzą one w tryb koncentratora. Efektem jest zdegradowanie sieci przełączanej do poziomu współdzielonego medium, co pozwala np. na bierne podsłuchiwanie.
- Jeżeli ramki generowane przez atakującego mają adres źródłowy równy adresowi hosta ofiary, to pamięć przełączników ulegnie przeprogramowaniu. Ruch, który płynął do tego hosta, zmieni kierunek, i popłynie do komputera intruza. Spowoduje to, że ofiara nie otrzyma danych, na które oczekiwała, natomiast otrzyma je atakujący.

Takie zachowanie sieci wynika ze sposobu działania składających się na nią przełączników. Zapisują one w swojej pamięci numer portu fizycznego i adres źródłowy ramek, które na nim się pojawiają. Te asocjacje są następnie używane w celu wyboru portu, przez jaki mają być wysyłane dalsze ramki.

1.3.5 Fałszywy serwer DHCP

Jeśli hosty w sieci są skonfigurowane w ten sposób, że pobierają ustawienia sieciowe z serwera DHCP, to atakujący może uruchomić drugi, konkurencyjny serwer, który będzie przydzielał komputerom użytkowników inne ustawienia. Atakujący może w ten sposób wymusić na komputerach, aby przesyłały ruch internetowy do jego maszyny, zamiast do routera. Manipulując wartością maski sieciowej, może także spowodować, że nawet transmisje lokalne będą kierowane do niego. Atakujący uzyskuje wtedy pełną kontrolę nad ruchem, tak jak w przypadku ataku ARP Spoofing.

[10] Atak MAC Flooding jest implementowany m.in. przez narzędzie `macof` z pakietu `dsniff`.

Inną możliwością jest zmiana adresu serwera DNS na adres maszyny kontrolowanej przez atakującego. Fałszywy serwer DNS natomiast będzie przekierowywał wybrany ruch na host oszusta, który znów przejmuje nad nim pełną kontrolę.

Atakujący może też selektywnie blokować hosty w sieci przez przydzielanie nieprawidłowych adresów, co skutkuje efektem odmowy usługi.

1.3.6 Przekierowania ICMP

Jeśli host próbuje wysyłać pakiety IP przez router, zaś ten posiada informacje o lepszej trasie, prowadzącej przez bramę znajdującą się w tej samej sieci, to router wysyła do hosta komunikat ICMP Redirect. W komunikacie zawarta jest informacja o adresie routera, którego host powinien użyć aby wysyłane przez niego pakiety trafiły do miejsca przeznaczenia.

Komunikaty te mogą być wysyłane także przez atakującego w celu przekierowania ruchu hosta do kontrolowanej przez niego maszyny[11]. Maszyna ta uzyskuje pełną kontrolę nad ruchem hosta, zaś efekty są podobne jak w przypadku ataku ARP Spoofing.

1.3.7 Resetowanie połączeń

Jeśli ataki zapewniające kontrolę nad transmisją nie są wykonalne, zaś celem atakującego jest uniemożliwienie komunikacji w sieci to może posunąć się do prób resetowania połączeń. Polega to na wstrzykiwaniu do sieci pakietów, które służą do awaryjnego zawieszania połączeń TCP i transmisji UDP[12]. W przypadku protokołu TCP są to segmenty z ustawioną flagą RST, zaś przerywanie połączeń UDP odbywa się z wykorzystaniem komunikatów ICMP Unreachable. Pakiety resetujące wymagają podstawowych informacji o połączeniu jak adresy i porty źródłowe oraz przeznaczenia, a także numery sekwencyjne dla połączeń TCP. Dlatego ten atak jest najbardziej efektywny w połączeniu z podsłuchiwaniem ruchu sieciowego. Z przechwyconego ruchu atakujący wybiera te połączenia, które chce zakończyć.

Inną metodą jest *zgadywanie* części danych opisujących połączenie, o którym pozostałe informacje są znane. Na przykład jeśli atakujący wie, że dany host ustanowił długotrwałą sesję z innym hostem wykorzystując protokół FTP, to generuje pakiety resetujące z wszystkimi możliwymi kombinacjami portów źródłowych i numerów sekwencyjnych[13]. Pozostałe dane jak adresy IP i numer portu docelowego, są znane.

[11] Narzędziem, które można do tego użyć jest na przykład `ICMP redirect flooder`, dostępny na `http://www.phenoelit.de/irpas/`.

[12] Przykładem programu do resetowania połączeń jest `Couic`, dostępny pod adresem `http://michel.arboi.free.fr/UKUSA/couic.html`.

[13] Przegląd zupełny przestrzeni numerów sekwencyjnych nie jest konieczny, ponieważ implementacje TCP akceptują pakiety z numerami znajdującymi się w obrębie tzw. *okna* [Watson, 2003].

“There are only 10 types of people in this world. Those who know binary, and those who don’t.”

– autor nieznany

Rozdział 2

Wykorzystywane technologie

Aby czytelnik mógł świadomie korzystać z programu *multispoof*, a także aby zrozumieć jego budowę, musi znać podstawowe technologie, wykorzystywane przez tę aplikację. W tym rozdziale zostaną przybliżone cztery najważniejsze. Zakłada się, że czytelnik posiada ogólną wiedzę na temat systemu GNU/Linux, jego administracji oraz zasad funkcjonowania lokalnych sieci komputerowych wykorzystujących protokół IP [Hunt, 1997] [Bautts *et al.*, 2005] [Russell, 2001].

2.1 Podsłuchiwanie sieci

Podsłuchiwanie sieci LAN to proces polegający na odbieraniu wszystkich danych, które pojawiają się na interfejsie sieciowym, w formie ramek ethernetowych. Ramki odebrane przez sterownik karty sieciowej są przesyłane równolegle do programu lub programów nasłuchujących oraz do stosu sieciowego w systemie operacyjnym, dlatego proces nie zaburza normalnego funkcjonowania aplikacji korzystających z sieci. Dodatkowo, jeśli karta sieciowa znajduje się w trybie *promiscuous* odbierane ramki nie podlegają filtrowaniu, które standardowo odbywa się w karcie. Domyślnie bowiem, karta sieciowa pozwala tylko na odbiór ramek o adresie docelowym zgodnym z adresem MAC karty oraz ramek rozgłoszeniowych i *multicastowych*[1]. Aplikacja nasłuchująca ma dostęp do warstwy drugiej i wyższych modelu OSI, co pozwala na odbiór informacji przenoszonych nawet przez protokoły, które nie są obsługiwane przez system operacyjny.

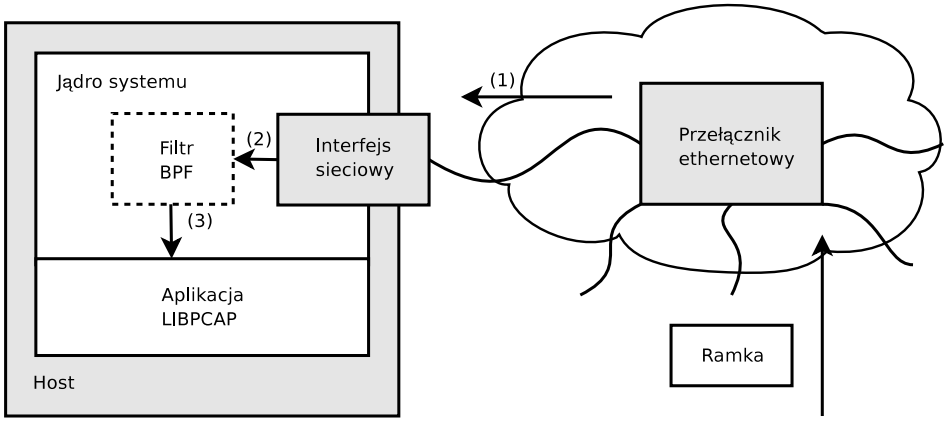
Podsłuchiwanie sieci wykorzystywane jest głównie w analizatorach pakietów, nazywanych *snifferami*. Znajdują one zastosowanie przy diagnozowaniu problemów oraz pomagają poznawać protokoły sieciowe. Funkcje przechwytywania informacji wysyłanych przez inne hosty są także wykorzystywane przez komputerowych włamywaczy np. w celu poznawania haseł.

Podsłuchiwanie w trybie *promiscuous* umożliwia analizę ruchu całej sieci, pod warunkiem, że jest ona oparta na koncentratorach. Sieci budowane z wykorzystaniem przełączników są bardziej odporne na bierne podsłuchiwanie[2], jednak zastosowanie metod aktywnych w dalszym ciągu pozwala na przechwytywanie transmisji (zostało to wyjaśnione w podrozdziale 1.3.3). Niemniej aplikacja, która powstała w ramach pracy dyplomowej, nie wymaga przechwytywania transmisji innych hostów.

Programy, które wykorzystują podsłuchiwanie sieci, często nie potrzebują wszystkich ramek, które odbiera karta. Zazwyczaj wybierają z nich tylko te, które spełniają określone kryteria. W tym celu został opracowany filtr BPF (ang. *Berkeley Packet Filter*). Pozwala on na wstępną selekcję danych z wykorzysta-

[1] Dzięki temu system operacyjny hosta nie musi analizować (i odrzucać) ramek, które nie są przeznaczone dla tego komputera. Efekt tej optymalizacji jest szczególnie widoczny w sieciach opartych na koncentratorach i charakteryzujących się dużym ruchem, ponieważ wtedy filtrowanie ramek na poziomie karty sieciowej istotnie odciąża procesor hosta.

[2] W sieciach przełączanych do hosta docierają jedynie ramki rozgłoszeniowe, ramki o adresach zarejestrowanych w pamięci przełącznika i skojarzonych z portem, do którego jest podłączony host, oraz ramki z adresami, których przełącznik nie ma w pamięci. Host nie otrzymuje większości ramek, które nie są do niego przeznaczone. Nie oznacza to jednak, że jedynymi ramkami, które do niego docierają są te, które powinien otrzymać.



Rysunek 2.1: Droga ramki od momentu nadania do otrzymania przez aplikację podsłuchującą ruch sieciowy.

niem prostego języka[3]. Program w tym języku jest optymalizowany, kompilowany i ładowany do jądra[4]. Droga ramki sieciowej od momentu nadania, do otrzymania przez aplikację podsłuchującą jest zobrazowana na rysunku 2.1. Zanim ramka dotrze do programu, może zostać odfiltrowana na przełączniku sieciowym (1), ze względu na to, że jej adres docelowy znajduje się już w pamięci przełącznika i jest skojarzony z portem innego hosta. Jeśli ramka dotrze do karty sieciowej hosta (2), karta nie znajduje się w trybie *promiscuous*, zaś adres docelowy MAC ramki nie pokrywa się z adresem karty ani nie jest adresem rozgłoszeniowym, wtedy karta odrzuci ramkę. W przeciwnym razie ramka jest analizowana przez program filtra BPF (3) i tylko jeśli spełni jego warunki zostanie dopuszczona do aplikacji.

2.2 Wstrzykiwanie danych do sieci

Wstrzykiwaniem określa się wysyłanie ramek ethernetowych bezpośrednio do sterownika karty sieciowej w systemie operacyjnym, czyli z pominięciem większości stosu sieciowego. Aplikacja w całości decyduje o zawartości ramki i może dowolnie określać wszystkie jej pola i dane protokołów wyższych warstw. Tak więc operacje, które nie są udostępniane przez system operacyjny, jak zmiana adresu źródłowego, wybranie docelowego adresu MAC i inne – są możliwe w trybie wstrzykiwania.

Podobną funkcjonalność oferują tzw. surowe gniazda sieciowe (ang. *RAW sockets*). Pozwalają one na generowanie z poziomu aplikacji dowolnych pakietów IP, które następnie są opakowywane przez jądro systemu w ramki protokołu warstwy drugiej modelu OSI [Al-Herbish, 1999]. Zaletą surowych gniazd jest niezależność od medium transmisyjnego, natomiast wadą brak kontroli nad ramką.

Obie metody wysyłania danych są wykorzystywane głównie w aplikacjach do diagnozowania sieci oraz testowania i przełamywania zabezpieczeń.

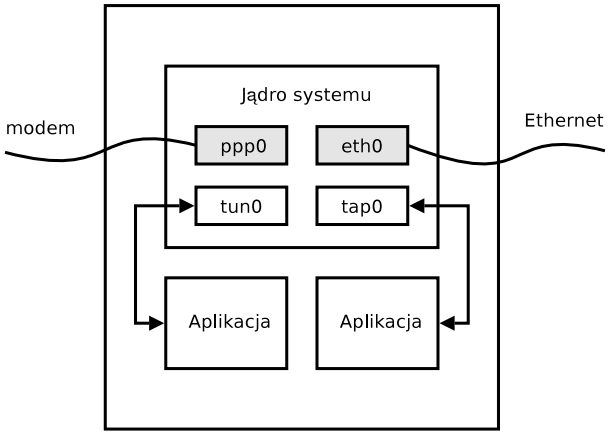
2.3 Koncepcja wirtualnych interfejsów *tun/tap*

Interfejs *tun* jest zapewnianym przez jądro wirtualnym urządzeniem sieciowym typu punkt-punkt. Podobnie interfejs *tap* z tą różnicą, że oferuje on wirtualną kartę ethernetową. Oba typy interfejsów są dynamiczne, tzn. mogą być tworzone w dowolnej ilości i usuwane [Krasnyansky i Thiel, 2002]. Utworzenie i komunikacja z interfejsem odbywa się odpowiednio przez otworzenie pliku urządzenia[5] oraz odczyt i zapis do deskryptora z poziomu aplikacji. Dane które zostaną zapisane pojawią się na interfejsie *tun* jako pakiet IP, zaś na interfejsie

[3] Przykładowo, program który akceptuje jedynie pakiety o adresie źródłowym `156.17.40.10` niosące protokół TCP będzie miał postać: `tcp and host 156.17.40.10`. Dokładny opis BPF jest dostępny w [McCanne i Jacobson, 1992].

[4] W przypadku większości systemów operacyjnych wykonywanie programu filtra jest realizowane w jądrze systemu, co zwiększa wydajność przetwarzania ramek.

[5] Jest to urządzenie znakowe, najczęściej dostępne jako `/dev/net/tun` lub `/dev/tunX`, gdzie X jest numerem interfejsu. Typ interfejsu (*tun* czy *tap*) jest wybierany za pomocą wywołania ioctl.



Rysunek 2.2: Porównanie interfejsów wirtualnych i fizycznych.

tap jako ramka ethernetowa. Pakiet bądź ramka, która zostanie wysłana przez interfejs może być odczytana z deskryptora jako ciąg bajtów.

Interfejsy *tun/tap* są wykorzystywane zazwyczaj w tunelowaniu. Tunelowanie polega na przesyłaniu pakietów lub ramek, po wcześniejszej enkapsulacji w pakiety specjalnych protokołów komunikacyjnych. Po obu stronach tunelu znajdują się interfejsy *tun/tap*, dzięki czemu użytkownik odnosi wrażenie, że komunikuje się z drugą stroną przez dedykowane połączenie fizyczne. Tunelowanie jest z reguły używane w wirtualnych sieciach prywatnych (VPN) oraz zastosowaniach eksperymentalnych[6].

Rysunek 2.2 przedstawia porównanie interfejsów wirtualnych *tun/tap* oraz interfejsów fizycznych. Interfejsy fizyczne nadają i odbierają dane do/z odpowiedniego medium transmisyjnego za pośrednictwem właściwego sprzętu sieciowego. Natomiast interfejsy wirtualne komunikują się z aplikacjami, znajdującymi się w przestrzeni użytkownika.

2.4 Translacja adresów NAT

NAT (ang. *Network Address Translation*) to proces, który odbywa się zwykle na routerze sieciowym. Polega on na dynamicznym tłumaczeniu adresów pakietów IP. Najczęściej używany jest w celu zapewnienia dostępu do Internetu sieci, która używa prywatnego zakresu adresów IP[7]. W tym przypadku router natujący zmienia źródłowy adres IP każdego pakietu wychodzącego do Internetu na adres publiczny routera. Każdy pakiet powracający do sieci otrzymuje adres odpowiadający komputerowi, który zainicjował transmisję. Dzięki temu cała sieć może wykorzystywać jeden publiczny adres IP[8].

NAT jest wykorzystywany także podczas równoważenia obciążenia pomiędzy kilka serwerów dysponujących pojedynczym adresem IP. Router równoważący obciążenie korzystając z translacji adresów może przypisywać kolejne połączenia do różnych serwerów, np. w zależności od ich obciążenia, podmieniając docelowy adres IP połączeń na adres konkretnego serwera, który znajduje się w sieci wewnętrznej. Dzięki temu użytkownicy korzystający z usług serwerów odnoszą wrażenie, że komunikują się z jednym hostem, zaś w rzeczywistości jest ich kilka.

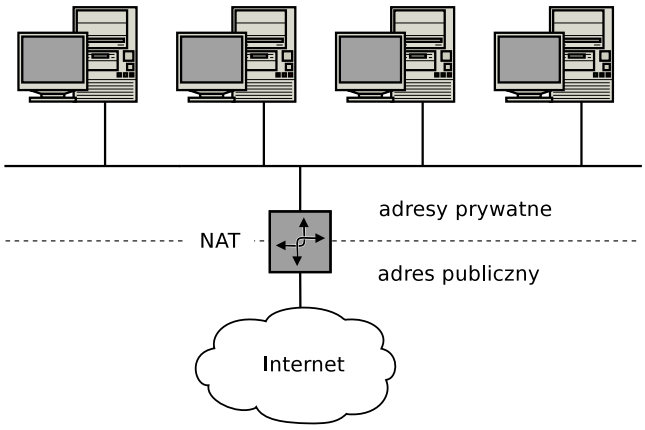
Zasada działania NAT została zobrazowana na rysunku 2.3. Zaawansowane implementacje NAT[9] umożliwiają tworzenie bardziej złożonych translacji niż wymienionego powyżej schematu *jeden do wielu*. Mogą to być *wiele do wielu* (adresy są odwzorowywane w liczniejszą lub mniej liczną pulę), *jeden do jednego* (pule adresów zewnętrznych i wewnętrznych są jednakowo liczne, odwzorowania są stałe). Możliwe jest

[6] Istnieją programy wykorzystujące do tunelowania protokoły, które nie były w tym celu projektowane jak np. ICMP (`http://neworder.box.sk/newsread_print.php?newsid=13688`) lub HTTP (projekt `HTUN`, `http://htun.runslinux.net/`).

[7] Zakresy adresowe przeznaczone do prywatnego użytku są określone w RFC 1918.

[8] Cechą tego typu translacji jest brak możliwości zainicjowania połączenia z Internetu do sieci wewnętrznej. Uniemożliwia to pracę programom, które oczekują na takie połączenia – np. aplikacjom serwerowym. Z drugiej strony, takie rozwiązanie zwiększa bezpieczeństwo hostów w sieci wewnętrznej, ponieważ są one w ten sposób separowane od Internetu i płynących z niego zagrożeń.

[9] Dostępne na przykład w otwartych systemach BSD lub w Linuksie [Russell, 2002*a*].



Rysunek 2.3: Sieć w której dostęp do Internetu jest zapewniany przez router z translacją adresów.

także wykonywanie translacji warunkowej w połączeniu z listą filtrów systemu firewall [Strebe i Perkins, 2000] [Russell, 2002*b*].

Master Foo once said to a visiting programmer: "There is more Unix-nature in one line of shell script than there is in ten thousand lines of C."

– *Eric S. Raymond*, Rootless Root *[Raymond, 2003]*

Rozdział 3

Aplikacja *multispoof*

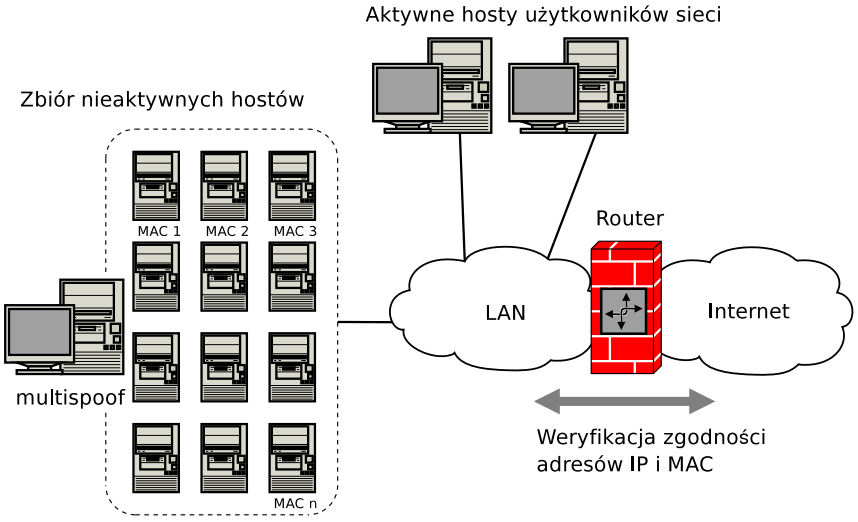
Aplikacja *multispoof* która powstała w ramach pracy dyplomowej, ma na celu praktyczną demonstrację słabości najpopularniejszej metody uwierzytelniania w sieciach osiedlowych. Program pozwala na przełamywanie zabezpieczenia usługi dostępu do Internetu, które bazuje na sprawdzaniu zgodności adresów IP i MAC komputera użytkownika z danymi przechowywanymi w bazie operatora.

Aplikacja wykorzystuje podszywanie się pod nieaktywne hosty. W tym procesie fałszowane są adresy IP i MAC wysyłanych pakietów. Cechą odróżniającą działanie aplikacji od zwykłej zmiany adresu w systemie operacyjnym jest możliwość podszywania się pod wszystkie nieaktywne komputery *jednocześnie*. Wykorzystanie programu pozwala osiągnąć szybkość transmisji, mogącą równać się sumie przepustowości poszczególnych hostów, których adresy są wykorzystywane przez aplikację. Koncepcję działania programu przedstawia rysunek 3.1.

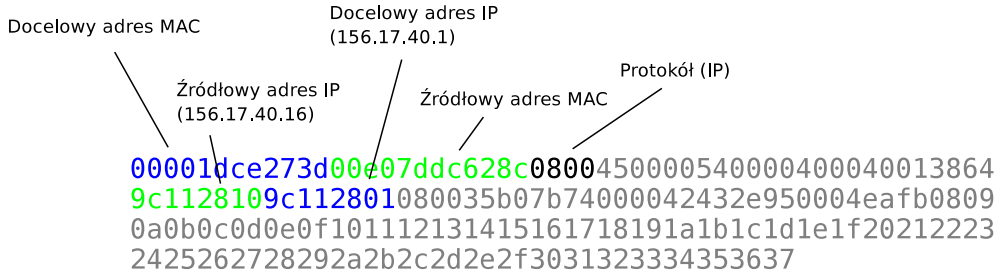
Dodatkową funkcją programu jest samodzielne rozpoznawanie sieci i tworzenie listy komputerów, które będą mogły być później wykorzystane w procesie podszywania się. Aplikacja nieustannie monitoruje sieć i jeśli zauważy, że wykorzystywany w procesie podszywania się host staje się aktywny, to natychmiast przestaje używać jego adresu, aby nie powodować konfliktów.

Należy podkreślić, że aplikacja *multispoof* powstała w celach demonstracyjnych, zaś używanie jej bez zgody właściciela sieci jest równoznaczne z kradzieżą usługi, co jest niezgodne z prawem.

Wewnętrzna architektura aplikacji *multispoof* ewoluowała w trakcie jej implementacji. Początkowo aplikacja miała mieć postać jednego programu. Szybko jednak okazało się, że pojedynczy program byłby zbyt skomplikowany, dlatego część funkcji została umieszczona w osobnym programie. Liczba programów zwiększała się wraz z identyfikacją kolejnych zadań, które aplikacja miała wykonywać. Doprowadziło to do niemal



Rysunek 3.1: Idea działania aplikacji *multispoof* w sieci lokalnej.



Rysunek 3.2: Przykładowy pakiet ICMP Echo Request zakodowany zgodnie z tekstowym protokołem ramki (dla czytelności złamany do kilku linii).

całkowitej separacji do osobnych procesów zadań, które nie były ze sobą ściśle powiązane. Jednocześnie były opracowywane interfejsy, za pomocą których procesy komunikowały się ze sobą. Mają one prosty, tekstowy charakter, który jest czytelny dla człowieka. Separacja zadań i proste interfejsy to metody modularyzacji aplikacji, które wydatnie przyśpieszyły proces pisania kodu, testowania oraz znajdowania i usuwania błędów [Raymond, 2003].

Na działanie aplikacji *multispoof* składają się trzy zasadnicze zadania:

- Zmiana adresów MAC
- Monitorowanie sieci
- Testowanie i równoważenie obciążenia

Każde z zadań jest realizowane przez zbiór procesów, nazwanych komponentami. Komponenty wykonują proste, jednowątkowe operacje z wykorzystaniem dwóch typów interfejsów, które zostaną omówione w podrozdziale 3.1. W podrozdziale 3.2 przybliżony zostanie komponent **`netdb`**, który uczestniczy we wszystkich trzech zadaniach. Następne podrozdziały będą dotyczyć kolejnych zadań, oraz wykonujących je komponentów. Rozdział zakończy opis skryptu *multispoof*, odpowiedzialnego za przygotowanie środowiska, utworzenie zadań z poszczególnych komponentów po uruchomieniu, oraz zatrzymanie ich przed zakończeniem.

3.1 Interfejsy i protokoły

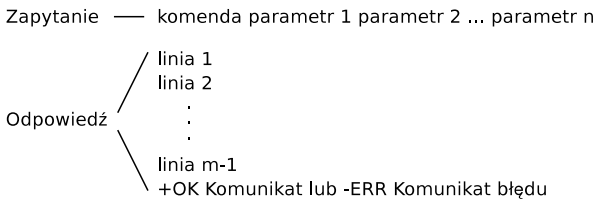
Interfejsy służą do jedno- lub dwustronnej komunikacji pomiędzy komponentami aplikacji *multispoof* z wykorzystaniem odpowiednich protokołów. Intencją przyświecającą ich budowie była prostota, dlatego dane wymieniane z ich wykorzystaniem mają postać tekstową. Decyzja ta wpłynęła na dużą elastyczność protokołów, co znacznie ułatwiało bieżącą ich rozbudowę podczas powstawania aplikacji. Oprócz tego forma tekstowa jest czytelna dla człowieka i daje się przetwarzać przy użyciu narzędzi uniksowych (np. `grep`, `sed`, `awk`). Są to pożądane cechy ułatwiające proces testowania oprogramowania.

Pierwszy typ interfejsu, nazwany *tekstowym protokołem ramek*, jest używany w potokach między komponentami przetwarzającymi ramki sieciowe. Interfejs jest jednokierunkowy. Pozwala na wysyłanie odpowiednio zakodowanych ramek, lub ich odbieranie za pośrednictwem standardowego wejścia i wyjścia. Protokół ten ma format liniowy, każda linia zawiera jedną ramkę. Bajty ramki kodowane są heksadecymalnie, najstarszy bajt pierwszy (*Big Endian*). Przykładową ramkę obrazuje rysunek 3.2.

Drugi typ interfejsu, nazywany *interfejsem* lub *protokołem* ***`netdb`***, służy do wykonywania zapytań do bazy adresów **`netdb`** (baza ta będzie opisana w następnym podrozdziale). Interfejs wykorzystuje lokalne gniazdo uniksowe[1] i jest dwukierunkowy. Komponenty mogą wysyłać zapytania, zaś baza odsyła odpowiedzi.

Zapytania mają postać komend z parametrami rozdzielonymi białymi znakami. Odpowiedzi składają się z jednej lub więcej linii, z których ostatnia ma ustandaryzowany format, przedstawiony na rysunku 3.3. Jeśli odpowiedź składa się z więcej niż jednej linii, wszystkie oprócz ostatniej mają format specyficzny dla wydanej komendy.

[1] To ułatwia testowanie, można w tym celu wykorzystać np. program `socat`, dostępny pod adresem: `http://www.dest-unreach.org/socat/`



Rysunek 3.3: Schemat komunikacji pomiędzy komponentem klienckim i **`netdb`**.

Tablica 3.1: Lista komend komponentu **`netdb`**.

Komenda	Parametry	Opis
gethost	Adres IP	Wyświetla dane hosta: adres MAC, wartości czasowe oraz znaczniki.
host	Adres IP, MAC	Dodaje lub modyfikuje dane hosta. Uaktualnia czas ostatniej aktywności.
remove	Adres IP	Usuwa dane hosta.
do	Akcja, Adres IP	Ustawia flagi dla hosta. Możliwe akcje: enable, disable, start-test, stop-test.
dump	–	Wyświetla listę adresów IP.
getvar	Nazwa zmiennej	Wyświetla zawartość zmiennej.
setvar	Nazwa zmiennej, wartość	Ustawia wartość zmiennej.
quit	–	Kończy połączenie z netdb.

3.2 netdb

Zadaniem komponentu **`netdb`** jest umożliwienie dostępu do bazy adresów oraz zmiennych innym komponentom. Baza **`netdb`** jest używana niemal przez wszystkie inne procesy, działające w ramach aplikacji *multispoof*, z wyjątkiem **`tx`**, **`rx`** i **`tapio`**. Z tego względu komponent **`netdb`** zostanie omówiony na początku.

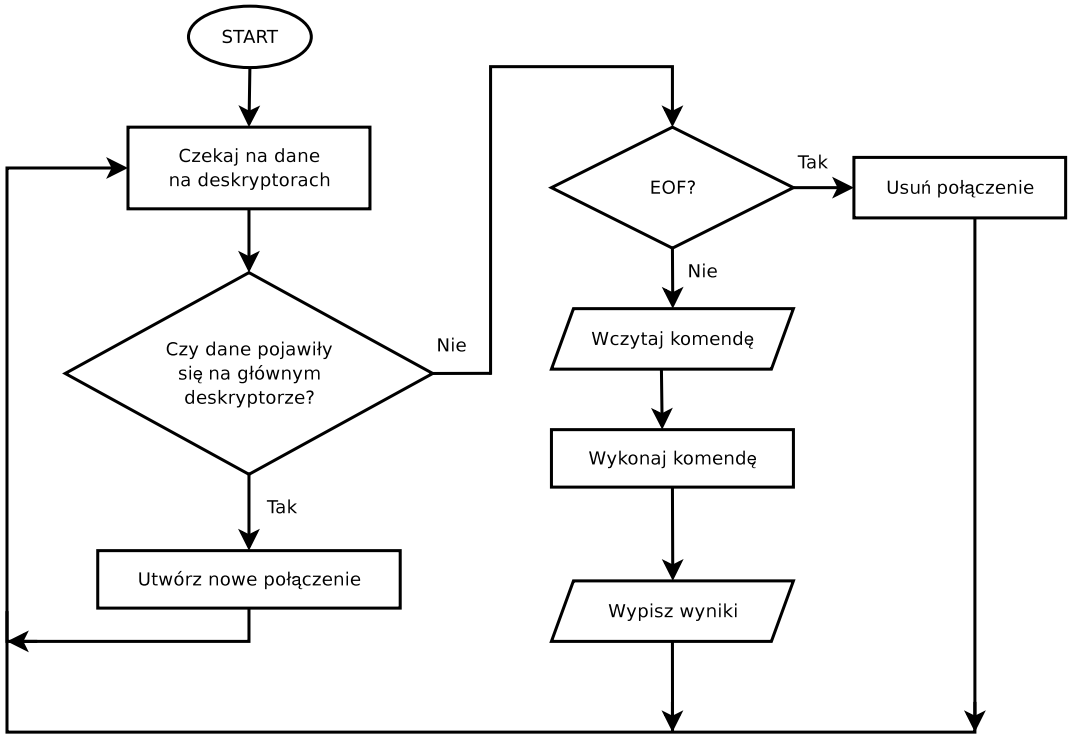
Komunikacja z innymi komponentami aplikacji jest realizowana z wykorzystaniem lokalnego gniazda uniksowego, które jest tworzone po uruchomieniu programu. Korzystając z tego gniazda inne komponenty mogą wydawać komendy i odczytywać wyniki zgodnie z tekstowym protokołem **`netdb`**. Tabela 3.1 zawiera listę dostępnych komend wraz z krótkimi opisami. Wybrane komendy będą omówione szerzej w następnych podrozdziałach.

Program **`netdb`** jest jednowątkowym serwerem opartym na funkcji *select*. Architektura ta charakteryzuje się wysoką wydajnością oraz prostotą implementacji [Brecht i Ostrowski, 2001]. Podczas ewaluacji różnych rozwiązań, została ona wybrana głównie ze względu na tę drugą cechę, w szczególności brak sekcji krytycznej[2]. Proces obsługi połączeń jest przedstawiony na rysunku 3.4.

Rozpatrywane były również inne sposoby realizacji dostępu do bazy hostów przez wiele procesów, lecz zostały odrzucone. Tablica hostów mogłaby być przechowywana w segmencie pamięci, dzielonym między komponentami, zaś synchronizacja odbywałaby się z wykorzystaniem semaforów. Zaletą tego rozwiązania jest wysoka wydajność, ponieważ operacje na pamięci są zdecydowanie szybsze niż komunikacja za pomocą gniazd lokalnych. Wadami są zwiększenie skomplikowania komponentów (potrzeba zaimplementowania synchronizacji, która sprzyja powstawaniu błędów, komplikacje związane z umieszczeniem dynamicznych struktur danych w pamięci dzielonej[3]) oraz trudność testowania (konieczne byłoby napisanie osobnego programu testującego, brak możliwości monitorowania odczytów/zapisów do/z pamięci). Podobnym pomysłem jest użycie pamięci prywatnej specjalnej biblioteki. Biblioteka udostępniałaby dostęp do bazy hostów, znajdu-

[2] Inne architektury serwerów, np. oparte na procesach lub na wątkach, wymagają stosowania mechanizmów takich jak semafory w celu wykluczenia jednoczesnego dostępu do wspólnych danych.

[3] Funkcje obsługi dynamicznych struktur danych dostępne w bibliotekach wykonują alokacje i dealokacje pamięci z wykorzystaniem wywołań systemowych `malloc()` i `free()`. Natomiast operacje na pamięci dzielonej wymagają jednorazowej rezerwacji wspólnego segmentu o wielkości, którą trzeba ustalić arbitralnie. Dynamiczne struktury danych, egzystujące w obrębie tego segmentu nie mogłyby używać funkcji zarządzania pamięcią udostępnianych przez system operacyjny i konieczne byłoby zaimplementowanie zarządzania przestrzenią pamięci segmentu. Choć np. biblioteka `glib` pozwala na wykorzystanie innych funkcji zarządzania pamięcią, to złożoność ich implementacji, oraz wspomniana wcześniej potrzeba synchronizacji dostępu do wspólnego segmentu, przeważyły nad korzyściami rozwiązania opartego na pamięci dzielonej.



Rysunek 3.4: Uproszczony algorytm obsługi połączeń w komponencie **`netdb`**.

jącej się w pamięci prywatnej za pomocą zestawu funkcji. Synchronizacja odbywałaby się tylko wewnątrz biblioteki, jednak pozostałe problemy są takie same jak w poprzednim rozwiązaniu.

Główne zadanie programu, czyli operacje na bazie hostów jest realizowane z wykorzystaniem tablicy haszującej, dostępnej w bibliotece `glib`[4]. Obsługa zmiennych (komendy `getvar` i `setvar`) wykorzystuje prostą tablicę struktur. Tablica ta jest statyczna, ponieważ zbiór zmiennych jest stały i zdefiniowany w kodzie.

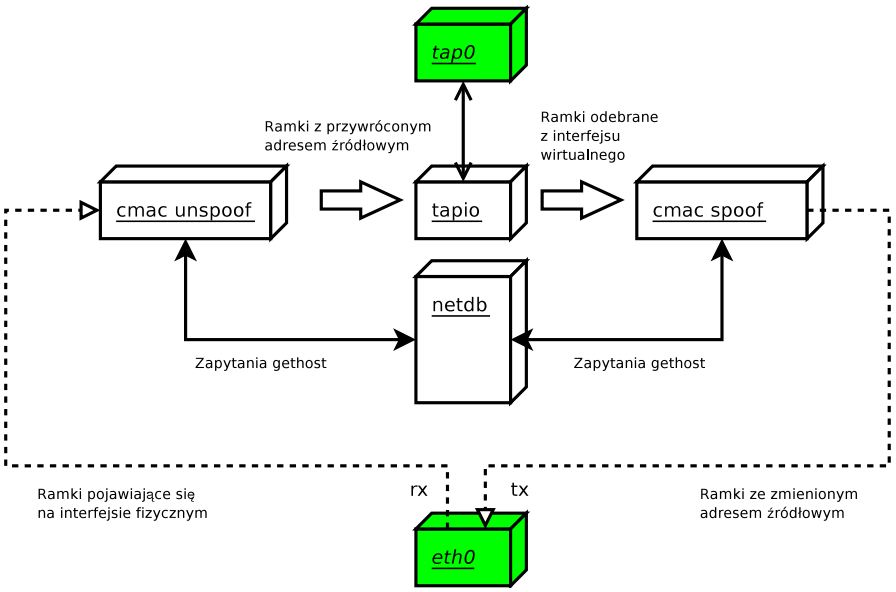
Po uruchomieniu program rejestruje gniazdo lokalne oraz wczytuje bazę hostów z pliku cache, który mógł zostać stworzony w wyniku poprzednich uruchomień. Jest to plik tekstowy, który zawiera sekwencje komend `host`. Ścieżka do pliku jest podawana jako parametr. Następnie przechwytywane są funkcje obsługi sygnałów, aby program przed zakończeniem mógł zapisać bazę hostów do pliku cache. Ostatnim krokiem jest wywołanie funkcji odpowiedzialnej za nasłuchiwanie na gnieździe i obsługę połączeń.

3.3 Zmiana adresów MAC

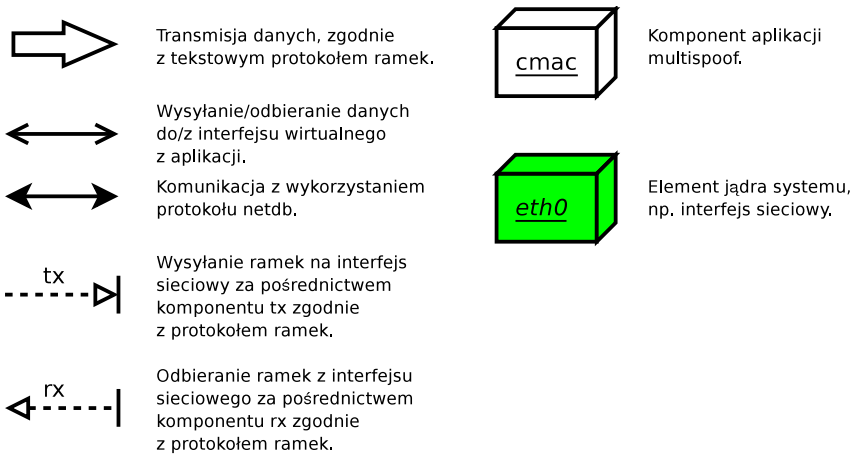
Proces zmiany adresów to podstawowe zadanie, które pochłania najwięcej zasobów w porównaniu z innymi zadaniami aplikacji *multispoof*. Dzieje się tak dlatego, że każda przychodząca lub wychodząca ramka ethernetowa musi zostać przetworzona przez szereg komponentów. Przetwarzanie obejmuje zmianę źródłowego adresu MAC w przypadku ramek wysyłanych, oraz przywracanie docelowego adresu MAC dla ramek przychodzących. Zmiana adresów następuje pomiędzy interfejsem wirtualnym a fizycznym. Interfejs wirtualny oddany jest do dyspozycji użytkownika, natomiast interfejs fizyczny jest pod pełną kontrolą aplikacji *multispoof*. Cały proces odbywa się z wykorzystaniem wspólnej dla wszystkich komponentów bazy adresów i jest przedstawiony na rysunku 3.5 (oznaczenia używane w diagramach są wyjaśnione na rysunku 3.6). Jeśli pominąć komponent **`netdb`**, jest to zwyczajny potok (dla czytelności parametry przekazywane komponentom zostały pominięte):

```
# rx | cmac unspoof | tapio | cmac spoof | tx
```

[4] Biblioteka `glib` dostępna jest pod adresem `http://www.gtk.org/`



Rysunek 3.5: Proces zmiany adresów MAC w ramkach ethernetowych.



Rysunek 3.6: Wyjaśnienie oznaczeń stosowanych na rysunkach.

Pomiędzy elementami potoku dane przekazywane są w sposób zakodowany zgodnie z omówionym wcześniej protokołem ramek. Komunikaty diagnostyczne są wysyłane na standardowe wyjście błędów, dzięki czemu nie zaburzają potoku. Każdy komunikat jest poprzedzany nazwą komponentu, który go wygenerował. Dzięki temu użytkownik jest na bieżąco informowany o istotnych wydarzeniach i jest w stanie przypisać je do konkretnego źródła. Role poszczególnych komponentów zostaną omówione w podrozdziałach.

3.3.1 rx i tx

Para programów **`rx`** i **`tx`** zajmuje się odpowiednio podsłuchiwaniem (odbieraniem ramek) oraz ich wstrzykiwaniem do sieci.

Komponent **`rx`** odbiera ramki z wykorzystaniem biblioteki `libpcap` [Carstens, 2002]. Jako parametry z linii komend podaje się interfejs, na którym program ma nasłuchiwać, oraz filtr BPF, określający które ramki będą odbierane. Podsłuchiwanie musi odbywać się w trybie *promiscuous*, ze względu na to, że aplikacja *multispoof* wykorzystuje wiele adresów MAC. Jeśli tryb ten byłby wyłączony, aplikacja otrzymywałaby tylko ramki, których docelowy adres MAC jest równy adresowi interfejsu sieciowego, oraz ramki rozgłoszeniowe.

Oprócz trybu *promiscuous*, komponent **rx** przed rozpoczęciem podsłuchiwania aktywuje filtr kierunku transmisji, tak aby odbierane były tylko ramki, które płyną z sieci, nie zaś te, które są wysyłane przez hosta. Domyślnie bowiem, biblioteka `libpcap` przechwytuje ramki płynące w obydwu kierunkach[5].

Podsłuchane ramki, które spełniają warunki narzucone przez filtr BPF oraz filtr kierunku, zostają wysłane przez komponent **rx** na standardowe wyjście, zakodowane zgodnie z tekstowym protokołem ramek.

Komponent **tx** zajmuje się wstrzykiwaniem ramek do sieci przez określony interfejs sieciowy. Ramki są wczytywane ze standardowego wejścia w formacie zgodnym z protokołem ramek. Wstrzykiwanie odbywa się z wykorzystaniem biblioteki `libnet` [Schiffman, 2004]. Jeśli ramka do wstrzyknięcia jest zbyt mała (standard Ethernet wymaga, aby minimalna wielkość wynosiła 60 bajtów), komponent **tx** powiększa ją, dopełniając zerowymi bajtami (ang. *padding*). Gdyby czynność ta nie była wykonywana i tak ostatecznie zrobiłby to sterownik karty sieciowej. Jednak wtedy analiza danych płynących przez interfejs z wykorzystaniem narzędzi typu *sniffer* nie ujawniłaby dopełnienia, co nie zgadzałoby się ze stanem faktycznym i mogłoby wzbudzać wątpliwości u osoby, która nie jest zaznajomiona z tym faktem.

Komponenty **rx** i **tx** wspólnie pozwalają na kompletną, niskopoziomową kontrolę ruchu przychodzącego i wychodzącego przez dany interfejs. Stanowią interfejs do sieci dla innych części aplikacji *multispoof*.

Rozbicie funkcji podsłuchiwania sieci oraz wstrzykiwania ramek na dwa oddzielne programy, oprócz korzyści związanych z modularyzacją aplikacji, może przydać się także w przyszłości. W kolejnej wersji można stosunkowo łatwo wprowadzić obsługę pracy w sieci bezprzewodowej. Radiowe interfejsy sieciowe charakteryzują się tym, że nie jest możliwe jednoczesne odbieranie i nadawanie. Wiąże się to z tzw. efektem oślepienia, który występuje w momencie zbyt szybkiego przełączenia interfejsu z trybu nadajnika w tryb odbiornika. Konieczne jest zastosowanie dwóch oddzielnych interfejsów sieciowych[6]: jednego tylko do wysyłania, zaś drugiego tylko do odbierania ramek. Obecna architektura aplikacji poradzi sobie z tym wymogiem, ponieważ każdy interfejs będzie obsługiwany przez osobny komponent[7].

3.3.2 tapio

Komponent **tapio** jest odpowiedzialny za utworzenie wirtualnego interfejsu *tap* oraz komunikację z nim. Program łączy z interfejsem standardowe wejście i wyjście, w ten sposób, że ramki wczytywane z wejścia są na niego wysyłane, zaś ramki z niego odbierane są wypisywane na standardowe wyjście. Proces ten jest zobrazowany na rysunku 3.7. Obsługa wejścia/wyjścia jest realizowana z zachowaniem tekstowego protokołu ramek. Nazwę interfejsu *tap*, który program ma utworzyć, podaje się jako parametr.

3.3.3 cmac

Komponent **cmac** jest programem typu filtr. Wczytuje on ramki ze standardowego wejścia, następnie jeśli spełnione zostaną kryteria akceptacji, modyfikuje je i wysyła na standardowe wyjście. Modyfikacja dotyczy zmiany adresu MAC ramki. W zależności od trybu podawanego jako parametr przy uruchomieniu, komponent podmienia adres źródłowy lub docelowy. Algorytm działania programu został przedstawiony na rysunku 3.8.

Po uruchomieniu **cmac** łączy się z komponentem **netdb**. Z każdej wejściowej ramki program wyodrębnia jej adres IP. Następnie wysyła do **netdb** zapytanie o ten adres (wykorzystując komendę `gethost`). Jeśli wpis zostanie znaleziony, to odpowiedź będzie zawierać adres MAC hosta, czas nieaktywności, czas który upłynął od ostatniego testu oraz informacje czy dany adres jest w tej chwili testowany i czy jest w stanie *enabled*. Komponent **cmac** testuje, czy uzyskane informacje spełniają opisane w tabeli 3.2 warunki zmiany adresu MAC. Jeśli test wypadnie pomyślnie adres MAC ramki zostaje zmieniony i ramka jest wypisywana na standardowe wyjście.

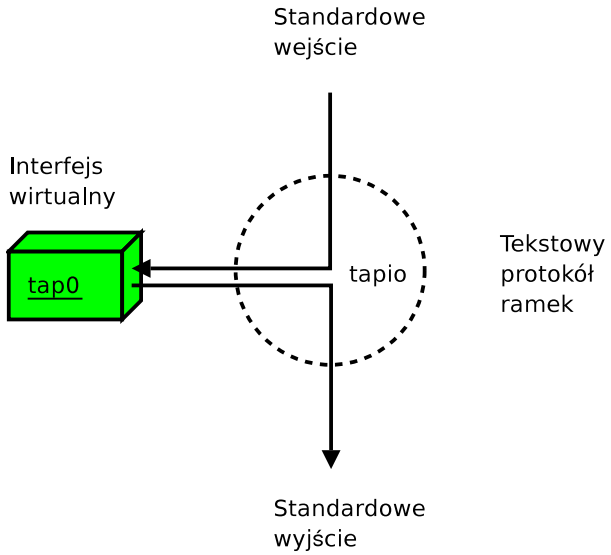
Program **cmac** musi pracować w jednym z dwóch trybów. Tryb działania ustalany jest na etapie uruchamiania. Poniżej opisane zostały oba tryby:

Spoof. Modyfikacji podlega źródłowy adres MAC, który na podstawie źródłowego adresu IP pakietu zawartego w ramce, jest zastępowany adresem uzyskanym z bazy **netdb**,

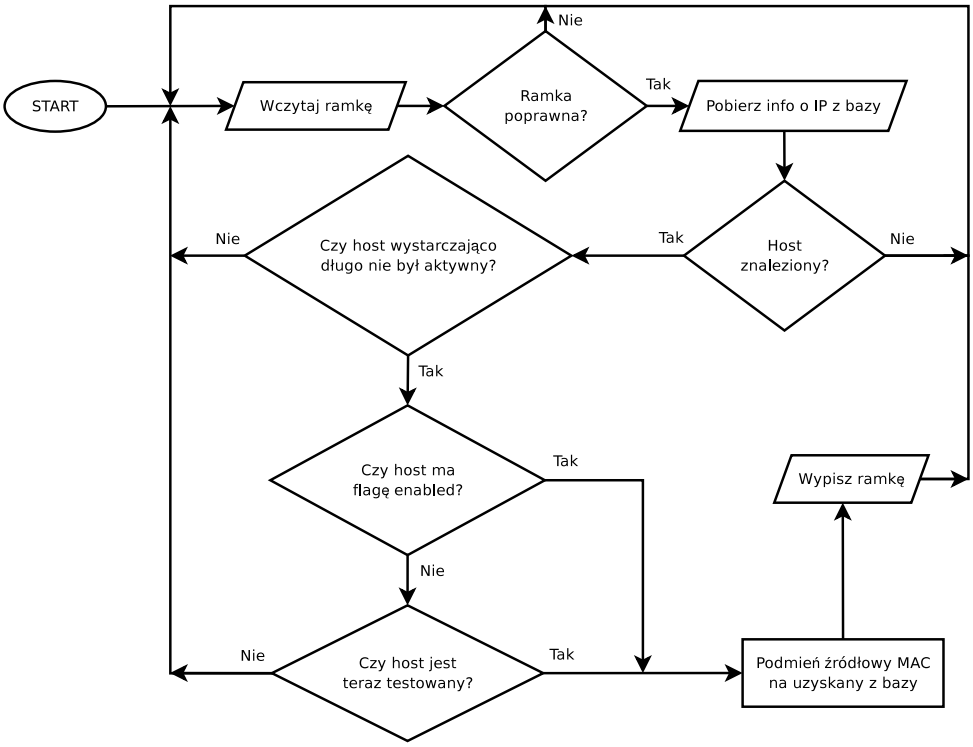
[5] Filtr kierunku transmisji jest dostępny dopiero w nowszych wersjach biblioteki `libpcap`, tj. wydanych po 3.05.2005.

[6] Dodatkowo, dobrze jeśli anteny podłączone do tych interfejsów znajdują się w odpowiedniej, niezbyt małej odległości [Szymański, 2003].

[7] Problemem do pokonania może okazać się odróżnianie ramek przychodzących od wychodzących, ponieważ interfejs nasłuchowy może także odbierać ramki wysyłane przez interfejs nadawczy.



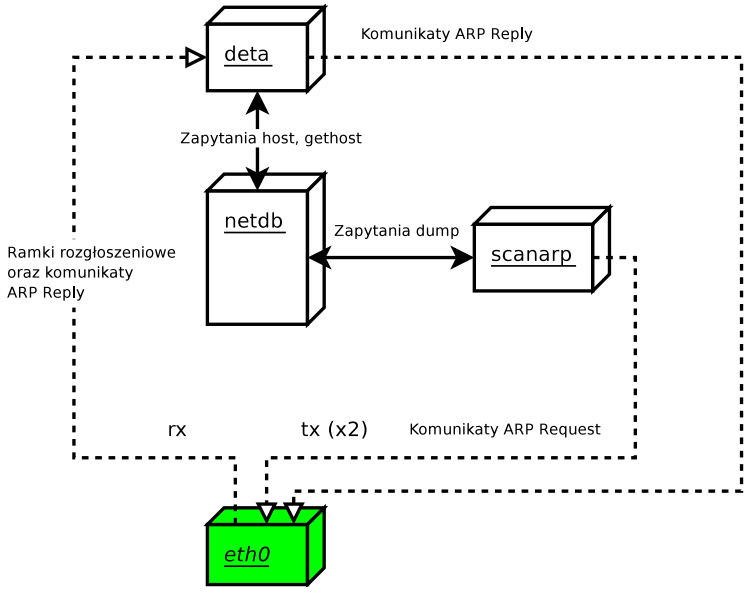
Rysunek 3.7: Schemat działania programu **`tapio`**.



Rysunek 3.8: Algorytm programu **`cmac`** w trybie spoof.

Tablica 3.2: Zestawienie warunków, które musi spełnić wpis dla hosta aby dany komponent brał go pod uwagę podczas wykonywania określonej czynności.

Warunki				Komponent	Czynność
age > min_age	test_age > min_test_age	enabled	cur_test		
Prawda	–	Prawda	–	cmac	Zmień MAC i wypisz ramkę
Prawda	–	–	Prawda		
Prawda	–	Prawda	–	deta	Wypisz odpowiedź na ARP
Prawda	–	–	Prawda		
Prawda	Prawda	–	–	conncheck	Wykonaj test połączenia
Prawda	–	Prawda	–	natman	Uwzględnij wpis w LB



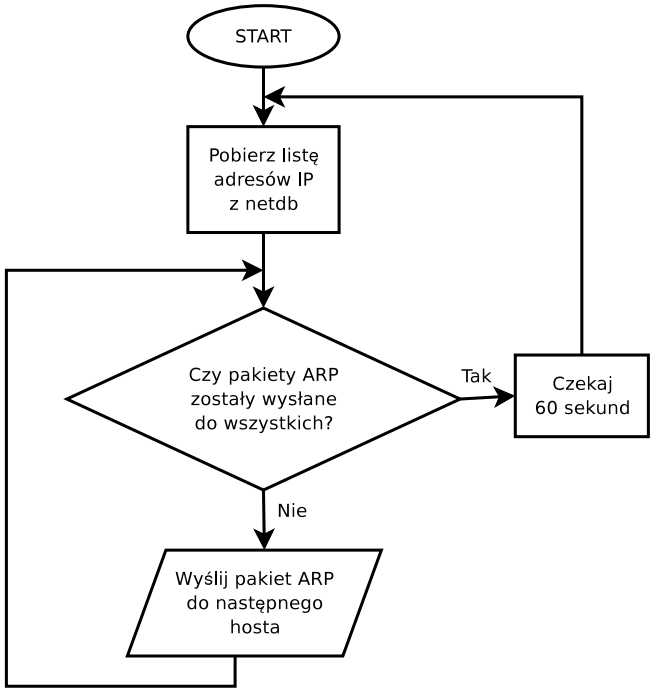
Rysunek 3.9: Proces monitorowania sieci.

Unspoof. Z ramki wyodrębniany jest docelowy adres IP pakietu. Następnie sprawdzane jest, czy w bazie **`netdb`** istnieje wpis dla tego adresu. Jeśli wpis istnieje, to docelowy adres MAC ramki jest zastępowany przez adres MAC interfejsu *tap*. Adres ten jest odczytywany z bazy **`netdb`** na etapie inicjalizacji. Należy podkreślić, że w tym trybie odczytany z bazy **`netdb`** dla każdej ramki adres MAC hosta jest ignorowany, wykorzystywane są tylko pozostałe informacje związane z hostem.

3.4 Monitorowanie sieci

Drugim zadaniem aplikacji *multispoof* jest monitorowanie sieci. Polega ono na biernym podsłuchiwaniu ruchu i aktywnym skanowaniu. W proces ten zaangażowane jest pięć komponentów, z których trzy: **`netdb`**, **`tx`** i **`rx`**, zostały omówione w poprzednich podrozdziałach. Rysunek 3.9 przedstawia schemat procesu, z uwzględnieniem połączeń między komponentami. Alternatywnie można zobrazować go za pomocą dwóch, wykonywanych równolegle potoków (parametry komponentów zostały pominięte dla czytelności):

```
# rx | deta | tx
# scanarp | tx
```



Rysunek 3.10: Algorytm komponentu **scanarp**.

3.4.1 scanarp

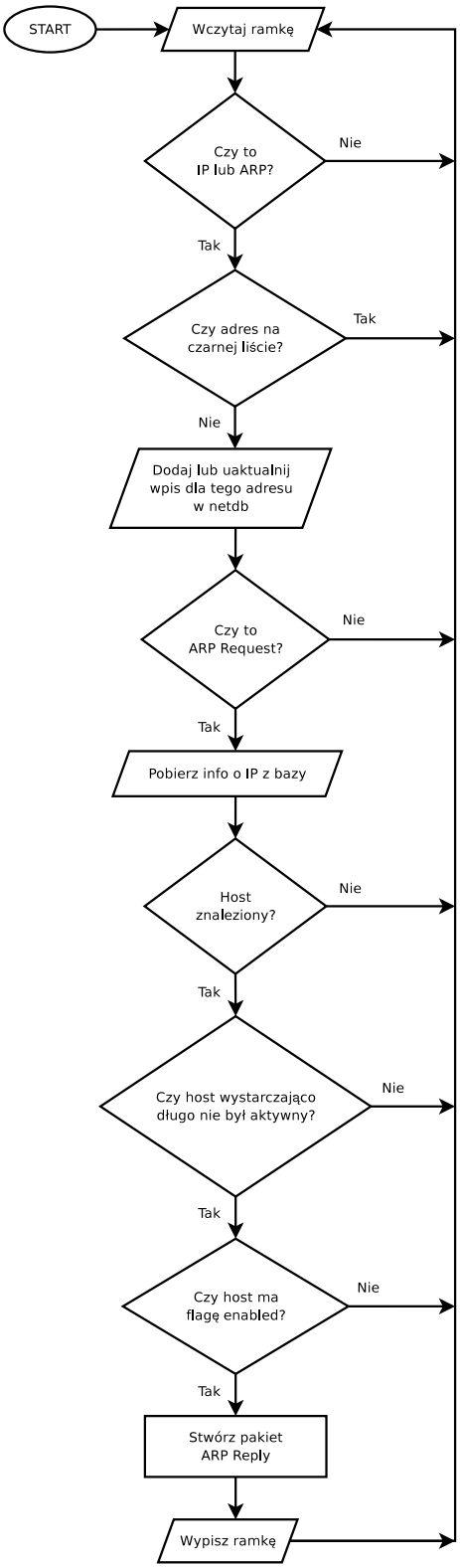
Komponent **scanarp** ma za zadanie periodyczne wysyłanie zapytań ARP Request do wszystkich hostów, których adresy są zarejestrowane w bazie **netdb**. Zapytania tego rodzaju są wysyłane, aby stwierdzić które z zarejestrowanych hostów są w danej chwili aktywne, bowiem działające komputery odpowiadają komunikatem ARP Reply[8]. Algorytm działania programu został przedstawiony na rysunku 3.10. Wysyłanie ramek w sieć jest realizowane za pośrednictwem komponentu **tx**, z wykorzystaniem protokołu ramek. Komponent pobiera listę hostów z bazy za pomocą komendy `dump`. Należy podkreślić, że aktywność sieciowa komponentu **scanarp** ogranicza się wyłącznie do wysyłania pakietów ARP Request. Odbieraniem odpowiedzi ARP Reply zajmuje się opisany dalej komponent **deta**.

3.4.2 deta

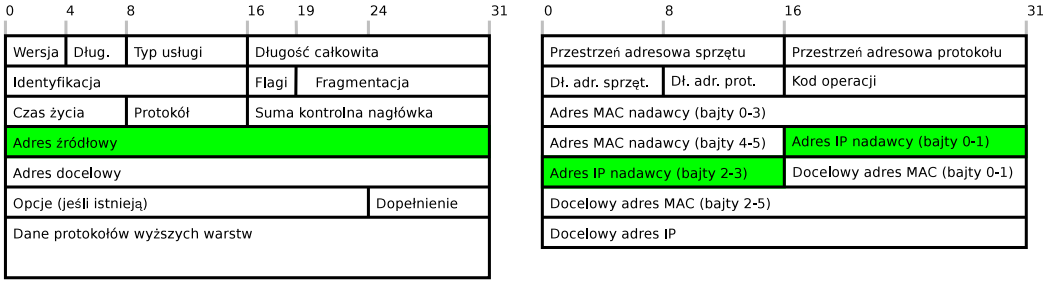
Komponent **deta** wykonuje dwa zadania, które ze względu na to, że muszą być wykonywane sekwencyjnie zostały zaimplementowane w jednym programie. Są to: wykrywanie aktywności hostów w sieci oraz odpowiadanie na zapytania ARP. Komponent nasłuchuje na fizycznym interfejsie sieciowym (za pośrednictwem programu **rx**), wyodrębnia adresy IP z przychodzących ramek, a następnie uaktualnia wpisy w bazie **netdb**. Jeśli z sieci przyjdzie zapytanie ARP Request, wtedy w zależności od zawartości bazy **netdb** może być udzielona odpowiedź i wysłana w sieć za pośrednictwem **tx**. Algorytm komponentu **deta** został przedstawiony na rysunku 3.11.

Komponent ma architekturę typu filtr. Na wejściu pojawiają się ramki, które program przetwarza, uaktualniając bazę **netdb**. Na wyjście komponent wypisuje ramki ARP Reply. Po odebraniu ramki, **deta** sprawdza jaki protokół ona niesie. Jeśli jest to protokół IP, to wyodrębniany jest źródłowy adres IP. Jeśli natomiast jest to pakiet ARP, to wyodrębniany jest adres IP komputera, który nadał ten pakiet (może to być zapytanie z sieci,

[8] Inne metody badania dostępności hostów w sieci wykorzystują pakiety ICMP, TCP lub UDP. Jednak w sieci lokalnej, metoda bazująca na protokole ARP daje pewniejsze rezultaty. Dzieje się tak, ponieważ mechanizm ten działa na pograniczu warstw drugiej i trzeciej modelu OSI oraz jest wymagany do prawidłowej pracy komputerów w sieci Ethernet. W przeciwieństwie do metod alternatywnych, zablokowanie tej metody nie jest proste i stwarza zagrożenie dla stabilności komunikacji. Przykładem programu, który pozwala na zbadanie dostępności zdalnego hosta w sieci Ethernet jest `arping`, dostępny pod adresem `http://www.habets.pp.se/synscan/programs.php?prog=arping`



Rysunek 3.11: Algorytm komponentu **`deta`**.



Rysunek 3.12: Umiejscowienie źródłowego adresu IP w pakietach IP i ARP.

bądź odpowiedź na zapytanie wysłane przez komponent **`scanarp`**). W obu przypadkach sprowadza się to do ustalenia adresu IP nadawcy, jednak dla każdego protokołu dana ta zapisana jest w nieco innym miejscu nagłówka (rysunek 3.12).

Gdy źródłowy adres IP jest już znany, komponent sprawdza, czy host ten znajduje się na czarnej liście. Lista ta jest pobierana przy starcie programu ze zmiennej *banned* komponentu **`netdb`**. Składa się ona z adresów, które są bezużyteczne w procesie podszywania się, natomiast pojawiają się w sieci. Zmienna jest definiowana w skrypcie *multispoof* (który będzie omówiony w podrozdziale 3.6) i zawiera takie adresy, jak `0.0.0.0`[9], `255.255.255.255` oraz adres domyślnej bramy[10]. Jeśli adres nie należy do czarnej listy komponent wykonuje komendę `host` na bazie **`netdb`** (patrz podrozdział 3.2, w szczególności tabela 3.1). Jako parametry do komendy zostają podane adres IP oraz źródłowy adres MAC ramki. Jeśli w bazie nie ma jeszcze wpisu dla tego hosta, jest on tworzony. Jeśli już jest, to czas, który upłynął od ostatniej aktywności, jest zerowany. Dzięki temu inne komponenty nie zaczną lub zaprzestaną wykorzystywać adres tego hosta, ponieważ stał się on aktywny. Jeśli ramki z tego hosta przestaną przychodzić, to po pewnym czasie inne komponenty z powrotem rozpoczną korzystanie z jego adresu.

Jeżeli przychodząca ramka niesie w sobie pakiet ARP Request, to następnym krokiem jest ustalenie, czy należy wysłać odpowiedź ARP. Odpowiedź ARP powinna być wysłana tylko wtedy jeśli adres, którego dotyczy zapytanie ARP, jest aktualnie w użyciu przez inne komponenty. W przeciwnym wypadku, tzn. jeśli fizyczny host o takim adresie jest aktywny, odpowiedź ARP mogłaby spowodować problemy w sieci[11]. Dlatego przed wysłaniem odpowiedzi program **`deta`** wykonuje zapytanie do bazy **`netdb`**, tym razem korzystając z komendy `gethost`. Jeśli host wystarczająco długo nie był aktywny oraz jeśli ma flagę *enabled* (patrz zestawienie warunków w tabeli 3.2 oraz opis komponentu **`conncheck`** w podrozdziale 3.5.1) to ramka z odpowiedzią ARP jest tworzona i wypisywana na standardowe wyjście.

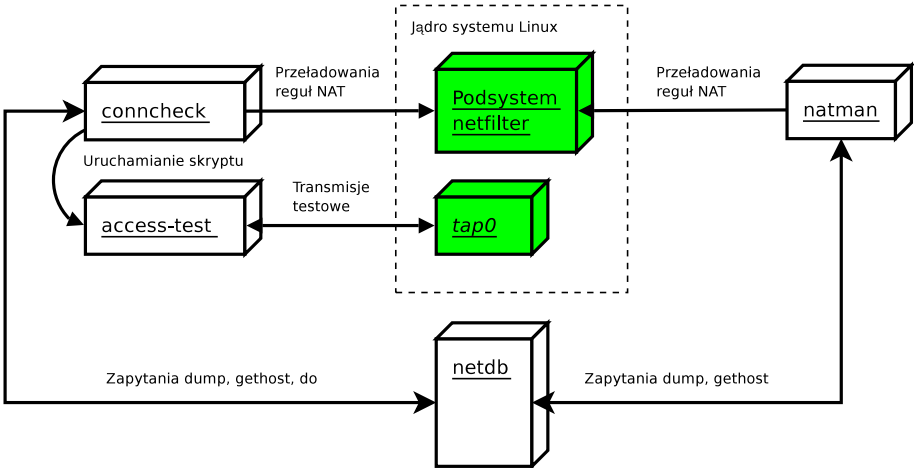
3.5 Testowanie i równoważenie obciążenia

Ostatnie zadanie aplikacji to typowanie nieaktywnych hostów które będą wykorzystywane w procesie zmiany adresów. Polega to na testowaniu łączności oraz przeładowywaniu reguł NAT odpowiedzialnych za równoważenie obciążenia. W procesach tych uczestniczą trzy komponenty, spośród których dwa nie zostały jeszcze omówione: **`conncheck`** i **`natman`**. Na rysunku 3.13 przedstawiono schemat tej części aplikacji, z uwzględnieniem połączeń między komponentami.

[9] Mogą to być np. adresy rozgłoszeniowe, używane przez protokół DHCP. Być może w następnych wersjach aplikacji *multispoof* pojawi się możliwość dodawania do czarnej listy całych sieci, a także określanie *białej listy*, co umożliwi użytkownikowi dokładne wyspecyfikowanie, które adresy mają być wykorzystywane, a które nie.

[10] Mimo, że aktywność bramy można sprawdzać w taki sam sposób, jak innych hostów, nie jest to konieczne, ponieważ podszywanie się pod adres bramy nie ma sensu. Umieszczenie jej adresu na czarnej liście *nie kosztuje prawie nic*, ponieważ lista jest już wykorzystywana do innych celów.

[11] Powodowałoby to podwójne odpowiedzi na zapytania ARP. Pierwsza odpowiedź pochodziłaby od fizycznego hosta, zaś druga od hosta, na którym działa aplikacja *multispoof*. Niektóre systemy operacyjne podczas inicjalizacji sieci wysyłają zapytanie ARP o własny adres IP, celem sprawdzenia czy nie występuje konflikt. Jeśli komponent **`deta`** udzieliłby wtedy odpowiedzi, to praca sieci zostałaby w zauważalny sposób zaburzona.



Rysunek 3.13: Procesy testowania i równoważenia obciążenia.

3.5.1 conncheck

Komponent **conncheck** decyduje które adresy z bazy **netdb** mogą być wykorzystywane w procesie podszywania się. Może się bowiem zdarzyć, że host znajdujący się w bazie **netdb** nie ma połączenia z Internetem (np. jego adres został zablokowany na routerze). Podszywanie się pod host, który nie ma łączności z siecią globalną mija się z celem aplikacji *multispoof*. Aby temu zapobiec program wybiera nieaktywne hosty i co pewien czas wykonuje dla każdego z nich test połączenia korzystając ze skryptu **access-test**. Adres hosta, który podlega testowaniu, jak i inne informacje są przekazywane do skryptu przez zmienne środowiskowe (tabela 3.3). Skrypt zwraca kod zakończenia równy 0, jeśli połączenie z Internetem zostało poprawnie zweryfikowane. Jeśli skrypt nie zdołał pozytywnie przetestować połączenia, to zwraca 1. Wersja aplikacji *multispoof*, która jest dołączona do tego opracowania, zawiera skrypt wykorzystujący program `nmap`[12]. Kod skryptu przedstawiony jest na listingu poniżej.

```
#!/bin/sh
HOST="www.mbank.com.pl"
PORT="443"

echo "access-test:␣(debug)␣checking␣$_IP␣started"
RESULT=1
nmap -sT -P0 -n -p $PORT $HOST 2> /dev/null | \
egrep "^$PORT" | grep -q "open" && RESULT=0
echo "access-test:␣(debug)␣checking␣$_IP␣finished␣(result:␣$RESULT)"

exit $RESULT
```

Po wywołaniu skryptu, program `nmap` wysyła segment TCP SYN na port 443 hosta o adresie `www.mbank.com.pl`. Jeśli otrzyma w odpowiedzi segment SYN ACK, wypisuje na standardowe wyjście stosowny komunikat. Komunikat jest przetwarzany przez narzędzie `grep`, które zwraca odpowiednią wartość logiczną, wykorzystywaną dalej przez skrypt do ustawienia zmiennej RESULT. Ostatnim krokiem jest zwrócenie wartości tej zmiennej. Metoda oparta na programie `nmap` została wybrana ze względu na to, że program ten generuje ruch sieciowy o stosunkowo niskim natężeniu. Cała transmisja składa się z jednego lub kilku zapytań DNS oraz rozpoczęcia połączenia TCP. Pakiety są małe i jest ich niewiele[13].

[12] Jest to popularny program służący do skanowania sieci komputerowych. Implementuje wiele metod skanowania [Klyagin, 2004]. Jest dostępny pod adresem `http://www.insecure.org/nmap/`

[13] Zbyt częste testowanie połączenia, z wykorzystaniem zaprezentowanej metody, może zostać odebrane przez zdalny system jako próba ataku typu odmowa usług (ang. *Denial of Service*). Z tego powodu program *multispoof* domyślnie wykonuje testy co godzinę, (dla każdego hosta czas jest liczony osobno), w sposób sekwencyjny. Dzięki temu próby nawiązywania połączeń wykonywane są stosunkowo rzadko.

Tablica 3.3: Zmienne środowiskowe przekazywane przez komponent **conncheck** do skryptu **access-test**.

Zmienna	Przykładowa zawartość (czasy w sekundach)	Opis
_IP	156.17.40.16	Adres IP hosta.
_MAC	00:e0:7d:dc:62:8c	Adres MAC hosta.
_AGE	1337	Czas nieaktywności hosta.
_TEST_AGE	6858	Czas, który upłynął od ostatniego testu.
_ENABLED	1	Stan flagi enabled w chwili testu.
_MIN_AGE	300	Minimalny czas nieaktywności hosta.
_MIN_TEST_AGE	3600	Minimalny czas między testami.
_NF_CHAIN	multispoof-test	Łańcuch dla tymczasowych reguł iptables.

Testowanie połączenia z Internetem zostało zorganizowane w powyższy sposób, aby użytkownik miał możliwość łatwego modyfikowania tego elementu i dostrajania go do sieci w której jest uruchamiana aplikacja *multispoof*. Także w tym celu do skryptu są przekazywane zmienne środowiskowe. W tej wersji skryptu bowiem nie są one wykorzystywane.

Aby przetestować połączenie z Internetem konkretnego hosta, komponent **conncheck** przed uruchomieniem skryptu **access-test** musi załadować do jądra regułę NAT[14], która spowoduje, że dane wysyłane i odbierane przez skrypt będą wykorzystywały adres IP testowanego hosta. Odbywające się aktualnie transmisje użytkownika aplikacji *multispoof* nie powinny zostać zakłócone, dlatego reguła musi dotyczyć tylko transmisji wykonywanych przez skrypt i programy przez niego uruchamiane. Aby spełnić ten warunek wykorzystywany jest moduł `owner`. Moduł ten pozwala na wyodrębnienie transmisji konkretnego procesu (wykorzystując jego PID) lub ich zbioru (za pomocą numeru SID - *session ID*[15]). Ponieważ wszystkie komponenty aplikacji *multispoof* są wywoływane przez jeden skrypt (opisany w podrozdziale 3.6), numer SID każdego procesu jest taki sam. Skrypt **access-test** jest jedyną częścią aplikacji, która wykonuje transmisje sieciowe, dlatego wykorzystanie w regule tego numeru nie zakłóci pracy programu. Jeśli numerem SID będzie 42, transmisje mają wykorzystywać interfejs *tap0*, zaś adres źródłowy ma być podmieniany na 156.17.40.16, to wywołanie programu `iptables` będzie miało poniższą formę:

```
# iptables -t nat -A POSTROUTING -o tap0 \
           -m owner --sid-owner 42 -j SNAT --to-source 156.17.40.16
```

W rzeczywistości, aplikacja *multispoof* wykorzystuje dodatkowe łańcuchy `iptables`. Jest to opisane dokładniej w podrozdziale 3.6.

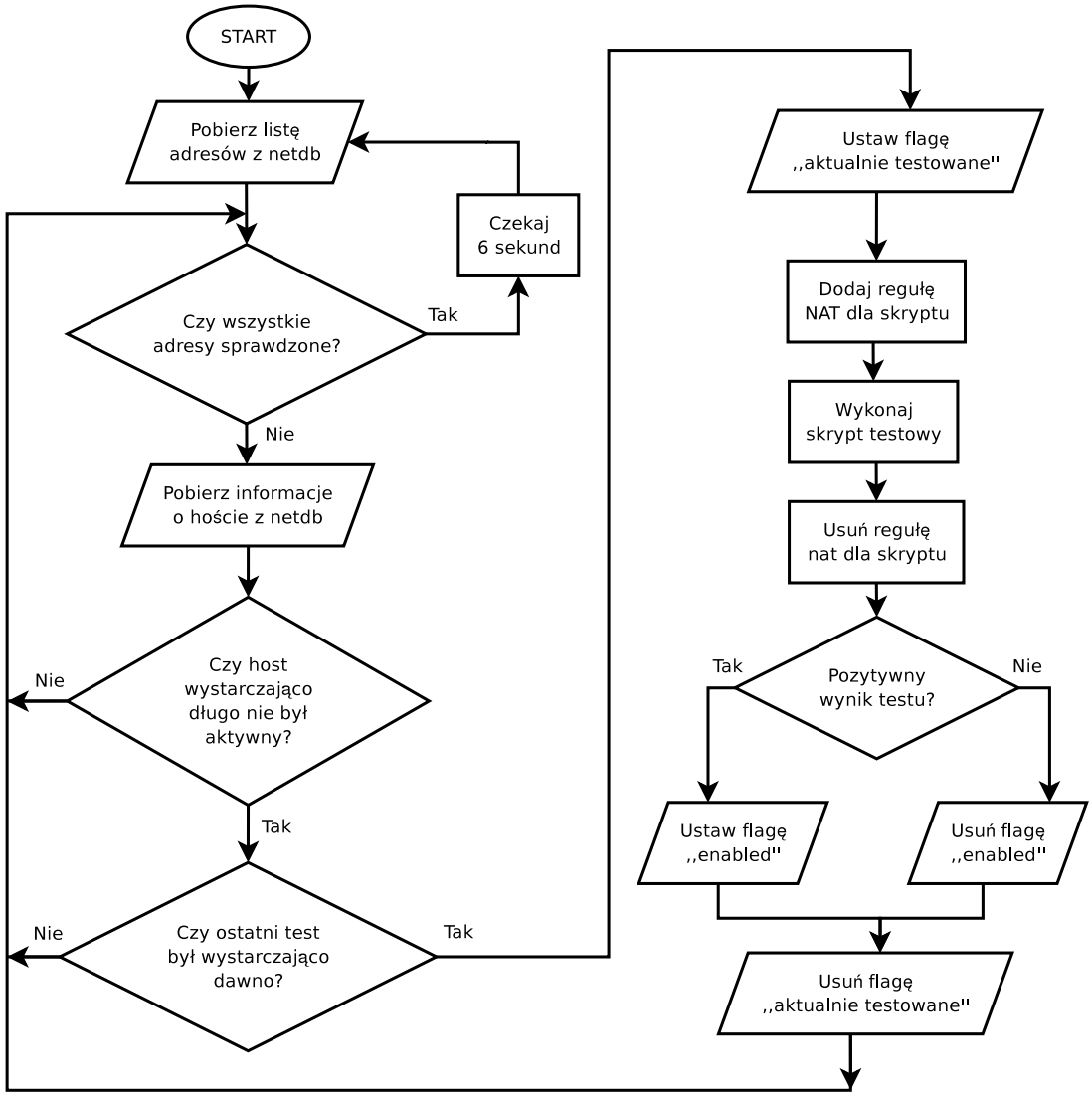
Przed uruchomieniem skryptu **access-test**, oprócz załadowania reguły NAT, **conncheck** wywołuje na bazie adresów **netdb** komendę `do start-test` (zobacz tabela 3.1). Komenda ta ustawia flagę, która informuje inne komponenty, że dany host jest aktualnie testowany. Dzięki temu np. komponent **cmac** zmienia i przesyła ramki nawet dla hostów, które nie są w stanie *enabled* (zachowanie poszczególnych komponentów w zależności od stanu flag jest przedstawione w tabeli 3.2). Dopiero po tym wywoływany jest skrypt testujący. Następnie w zależności od wyniku testu, stan hosta jest za pomocą komendy `do` zmieniany na *enabled* lub *disabled*. Zostaje jeszcze usunięcie znacznika testowania oraz reguły NAT, po czym komponent przechodzi do testowania następnego adresu. Algorytm działania programu jest przedstawiony na rysunku 3.14.

3.5.2 natman

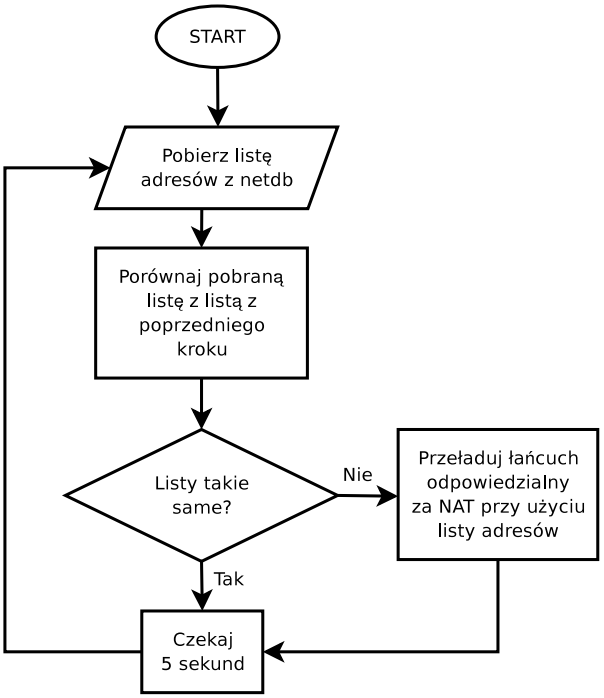
Komponent **natman** jest odpowiedzialny za utrzymywanie aktualności reguł NAT wykorzystywanych do równoważenia obciążenia. Program periodycznie pobiera listę hostów z bazy **netdb** i na jej podstawie buduje listę hostów, które są w stanie *enabled* i były nieaktywne przez wystarczającą długi czas. Tak przygotowana lista jest porównywana z listą z poprzedniej iteracji. Jeśli listy różnią się, to znaczy, że w bazie pojawiły się nowe hosty, które można wykorzystać, lub dotychczas nieaktywne komputery stały się aktywne. W obu przypadkach komponent przeładowuje listę reguł NAT, aby uwzględnić zmiany. Rysunek 3.15 przedstawia algorytm działania programu **natman**.

[14] Do tego celu komponenty aplikacji *multispoof* wykorzystują narzędzie `iptables`, które pozwala na zarządzanie regułami podsystemu `Netfilter`, dostępnego w jądrze Linuksa.

[15] Numer ten pozwala zidentyfikować grupę procesów – wszystkie procesy w sesji mają wspólny numer SID.



Rysunek 3.14: Algorytm działania komponentu **conncheck**.



Rysunek 3.15: Algorytm działania komponentu **`natman`**.

Równoważenie obciążenia to proces realizowany przez podsystem `Netfilter` jądra Linuksa, rozdzielający połączenia sieciowe i przypisujący je do poszczególnych adresów, które w danej chwili są wykorzystywane. Dzięki temu użytkownik może wygenerować ruch, którego natężenie wielokrotnie przekracza limit dla pojedynczego hosta.

Aby lepiej zrozumieć, na czym polega równoważenie obciążenia, należy prześledzić drogę pojedynczego pakietu. Pakiet, który przychodzi na interfejs wirtualny *tap*, jest porównywany z tablicą nawiązanych połączeń *conntrack*. Z każdym połączeniem związany jest adres IP. Jeśli pakiet należy do nawiązanego połączenia, to jego adres źródłowy zostanie zmieniony na ten, który jest zapisany w tablicy *conntrack*. Jeśli zaś pakiet nie należy do żadnego z istniejących połączeń, tzn. jest pierwszym pakietem połączenia (np. jest to pakiet TCP z ustawioną opcją SYN[16]), to adres, jaki zostanie mu przypisany, jest wybierany z puli dostępnych odpowiednim algorytmem rozkładania obciążenia. W obecnej wersji wykorzystywany jest algorytm *round robin*[17]. Polega on na tym, że każde nowe połączenie dostaje kolejny adres IP *modulo* liczba dostępnych adresów. Tzn. jeśli dostępnych jest 10 adresów, to pierwsze połączenie dostanie pierwszy adres, drugie drugi itd., zaś połączenie jedenaste otrzyma adres pierwszy. Jest to najprostsza metoda równoważenia obciążenia, nie uwzględniająca stopnia wykorzystania poszczególnych adresów. W kolejnych wersjach aplikacji *multispoof* mogą pojawić się inne metody.

Rozkładanie nowych połączeń metodą *round robin* na poszczególne dostępne adresy IP odbywa się z wykorzystaniem modułu `iptables` o nazwie `nth`[18]. Moduł ten zawiera liczniki trafień, które mogą być wyko-

[16] Bezstanowe protokoły jak UDP i ICMP także są reprezentowane w tablicy połączeń *conntrack*. Informacje o stanie są wyodrębniane z takich danych jak adresy IP i porty UDP lub typ i kod komunikatu ICMP.

[17] Nazwa algorytmu wywodzi się prawdopodobnie z XVII- lub XVIII-wiecznej Francji. Słowo robin powstało z francuskiego *ruban*, co oznacza wstęga. W tamtym czasie poddani chcąc poskarżyć się królowi na swój los podpisywali petycję. Zwyczajową reakcją króla była dekapitacja kilku pierwszych sygnatariuszy, tak więc nikt nie chciał, aby jego nazwisko było na początku listy. Dlatego opracowano metodę, która polegała na tym, że podpisy tworzyły okrąg (wstęgę). To uniemożliwiało stwierdzenie, kto podpisał się pierwszy. Później metoda ta przyjęła się także wśród żeglarzy floty brytyjskiej. Za `http://www.explore-dictionary.com/dictionary/R/Round-robin.html`. Polskim tłumaczeniem używanym np. w [Silberschatz i Galvin, 2001] jest *algorytm karuzelowy*, zaś chyba najbardziej trafnym, bo podkreślającym ludowe pochodzenie jest określenie *w koło Macieja*. Jednak z uwagi na fakt, że drugie tłumaczenie ma wydźwięk nie pasujący do charakteru tego opracowania, oraz ze względu na popularność wersji angielskiej, termin ten będzie używany w oryginalnym brzmieniu.

[18] W chwili pisania pracy nie jest on dostępny w standardowym jądrze Linuksa. Aby używać tego modułu należy zainstalować pakiet `patch-o-matic`, co jest opisane w rozdziale B.

rzystywane do wybierania co n-tego[19] pakietu lub połączenia. Odpowiednio wykorzystany w regułach NAT moduł `nth` pozwala na uzyskanie efektu równoważenia obciążenia. Poniżej przedstawiono przykładowy zestaw reguł dla czterech adresów IP:

```
# iptables -t nat -A POSTROUTING -o tap0 \
           -m nth --every 4 --packet 0 \
           -j SNAT --to-source 156.17.40.16
# iptables -t nat -A POSTROUTING -o tap0 \
           -m nth --every 4 --packet 1 \
           -j SNAT --to-source 156.17.40.17
# iptables -t nat -A POSTROUTING -o tap0 \
           -m nth --every 4 --packet 2 \
           -j SNAT --to-source 156.17.40.18
# iptables -t nat -A POSTROUTING -o tap0 \
           -m nth --every 4 --packet 3 \
           -j SNAT --to-source 156.17.40.19
```

Dla czytelności w powyższym przykładzie reguły umieszczane są bezpośrednio we wbudowanym łańcuchu POSTROUTING. Aplikacja *multispoof* wykorzystuje dedykowany łańcuch, w celu bardziej elastycznego zarządzania regułami. Tabela *nat* na której operują reguły posiada cechę, która nie jest spotykana w innych tabelach – mianowicie trafiają do niej tylko te pakiety, które inicjują połączenia[20]. Pozostałe pakiety ulegają translacji automatycznie, w zależności od tego do jakiego połączenia należą. Funkcja ta jest zapewniana przez moduł śledzenia połączeń *conntrack* podsystemu `Netfilter` i może być wymuszona także dla innych tabel z wykorzystaniem filtra *state*.

Śledzenie połączeń ulegających translacji jest fundamentalnych mechanizmem, dzięki któremu transmisja może odbywać się prawidłowo. W przeciwnym wypadku, tzn. gdyby mechanizm ten został zdezaktywowany, translacja dotyczyłaby pojedynczych pakietów a nie połączeń. Doprowadziłoby to do sytuacji, w której poszczególne pakiety jednego połączenia miałyby różne źródłowe adresy IP, co uniemożliwia właściwą identyfikację połączenia przez host, znajdujący się po jego drugiej stronie. Adres IP bowiem jest jedną z danych, które identyfikują połączenie[21] i przez czas jego trwania muszą pozostać niezmienne.

Ideę procesu rozkładania obciążenia została przedstawiona na rysunku 3.16. Pakiety zobrazowane są na nim jako różnokolorowe kółka. Kolor ich wypełnienia określa połączenie do którego przynależy dany pakiet. Środkowy okrąg określa miejsce, w którym przeprowadzany jest NAT według tabeli *conntrack*, która łączy połączenia z adresami IP. Na rysunku prawie wszystkie połączenia są już nawiązane i mają przypisane adresy IP. Pakiety oznaczone czerwonymi kółkami rozpoczynają połączenie, któremu adres IP zostanie przypisany zgodnie z algorytmem *round robin*.

3.6 Skrypt *multispoof*

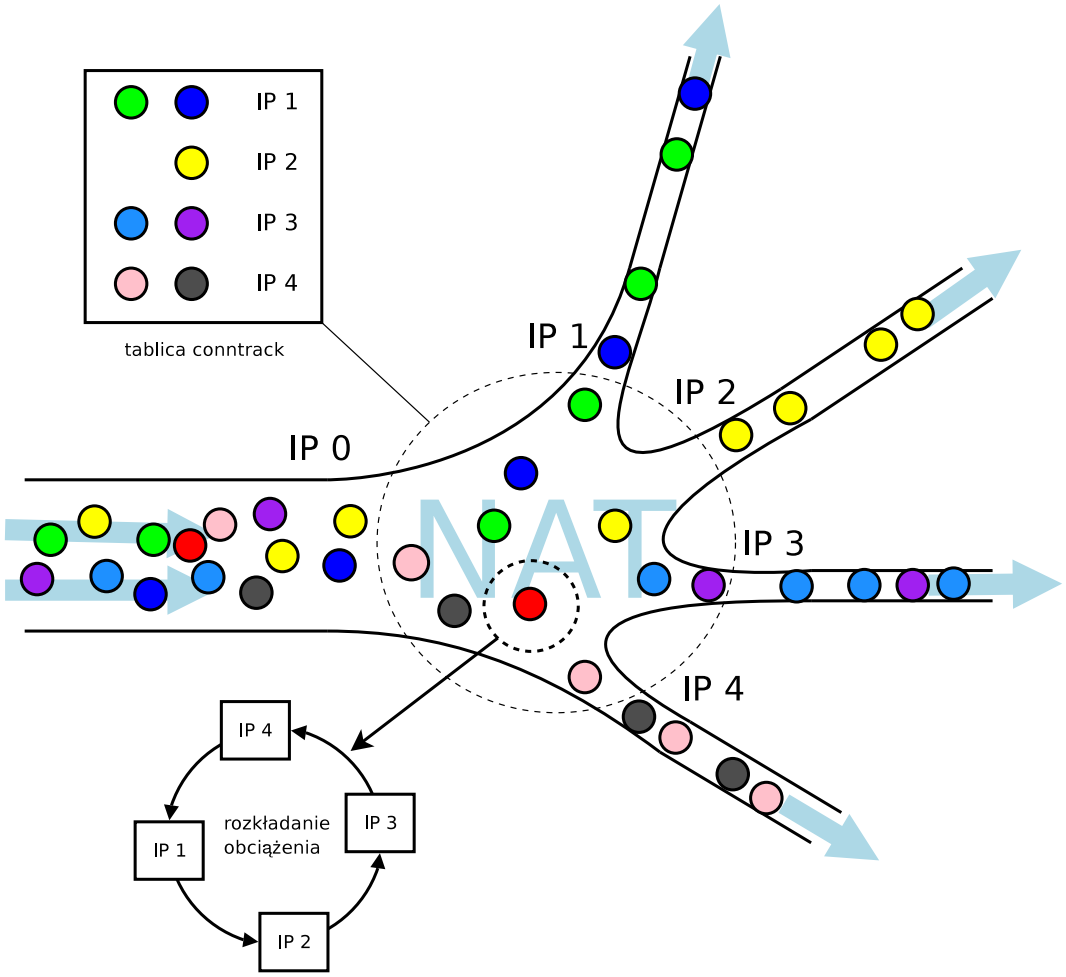
Skrypt *multispoof* stanowi tę część aplikacji, która jest odpowiedzialna za uruchamianie jej poszczególnych zadań. Chociaż zarówno skrypt, jak i pozostałe komponenty, są plikami wykonywalnymi, po instalacji całej aplikacji tylko skrypt staje się dostępny dla użytkownika w standardowej ścieżce dostępu PATH. Pozostałe komponenty są umieszczane w osobnym katalogu, nie uwzględnionym w ścieżce. Dzieje się tak, ponieważ działając samodzielnie, komponenty nie spełniają użytecznych zadań, zaś udostępnianie nieprzydatnych programów w systemie jest pozbawione sensu.

Skrypt został zaprogramowany w języku *Bourne shell*. Język ten został wybrany ze względu na to, że uruchamianie podprocesów jest w nim maksymalnie uproszczone. To samo dotyczy komunikacji międzyprocesowej z wykorzystaniem potoków. Oba te mechanizmy są intensywnie wykorzystywane w aplikacji

[19] Największą liczbą *n* jest 100, dlatego obecna wersja aplikacji *multispoof* nie pozwala na wykorzystywanie większej liczby adresów w procesie podszywania się. Jest to sztuczny limit zapisany w kodzie modułu `nth` w `iptables`. Prawdziwym ograniczeniem są ośmiobitowe liczniki używane przez moduł w jądrze systemu, które powodują, że liczba ta nie może przekroczyć 256. Autor podejmie próby rozwiązania tego problemu w następnych wersjach przez modyfikację modułu `nth`.

[20] Pozostałe tabele: *filter*, *mangle* i *raw* otrzymują wszystkie pakiety.

[21] W przypadku protokołów TCP i UDP są to cztery dane: adres źródłowy, adres docelowy, port źródłowy oraz port docelowy [Hunt, 1997].



Rysunek 3.16: Proces równoważenia obciążenia ruchu wychodzącego z wykorzystaniem śledzenia połączeń i NAT.

multispoof. Język *shell* stanowi *framework*, czyli niejako środowisko, w którym wykonują się poszczególne komponenty aplikacji [Raymond, 2003].

Skrypt wykonuje sekwencję kroków, które przygotowują środowisko oraz w odpowiedniej kolejności uruchamiają poszczególne komponenty, uwzględniając zależności między nimi. Po uruchomieniu wszystkich komponentów, aplikacja jest gotowa do pracy. Jej działanie zostało zobrazowane na rysunku 3.17, który przedstawia wszystkie zadania omówione w tym rozdziale. Poniższa lista przedstawia sekwencję kroków skryptu.

1. Odczytywanie parametrów, które użytkownik podał przy uruchomieniu[22]. Spośród wszystkich parametrów jedynym wymaganym jest nazwa interfejsu sieciowego, który ma być wykorzystywany przez aplikację.

2. Rozpoznawanie konfiguracji sieciowej: adresów IP i MAC interfejsu sieciowego, maski oraz adresów IP i MAC domyślnej bramy.

3. Weryfikacja uprawnień – jeśli użytkownik nie jest administratorem, to korzystanie z programu jest niemożliwe.

4. Utworzenie katalogu tymczasowego, w którym umieszczone zostanie gniazdo lokalne komponentu **`netdb`**.

5. Przechwycenie sygnałów, które przerywają działanie aplikacji, aby po otrzymaniu sygnału, aplikacja mogła przywrócić środowisko do pierwotnego stanu. Obejmuje to cofnięcie kroków 4 i 16.

6. Uruchomienie w tle komponentu **`netdb`** (opisanego w podrozdziale 3.2).

7. Oczekiwanie na utworzenie przez **`netdb`** gniazda lokalnego.

8. Zdefiniowanie czarnej listy adresów IP i wpisanie jej do zmiennej *banned* w bazie **`netdb`**.

9. Uruchomienie komponentów **`rx`**, **`tx`** (podrozdział 3.3.1), **`cmac`** (podrozdział 3.3.3) i **`tapio`** (podrozdział 3.3.2) w potoku odpowiedzialnym za zmianę adresów MAC.

10. Utworzenie potoku, realizującego zadanie skanowania sieci za pośrednictwem komponentów **`scanarp`** (podrozdział 3.4.1) i **`tx`**.

11. Uruchomienie komponentów **`rx`**, **`deta`** (podrozdział 3.4.2) i **`tx`** w potoku służącym do monitorowania sieci i odpowiadania na zapytania ARP.

12. Utworzenie łańcuchów podsystemu `Netfilter`. Tworzony jest jeden główny łańcuch *multispoof-main* oraz łańcuchy *multispoof-test* (wykorzystywany przez komponent **`conncheck`**) i *multispoof-sub* (do którego reguły ładowane są przez program **`natman`**). We wbudowanym łańcuchu POSTROUTING umieszczana jest reguła, która powoduje, że połączenia wychodzące przez interfejs wirtualny *tap* są kierowane do łańcucha *multispoof-main*. W łańcuchu tym jest definiowana reguła, która przekierowuje do łańcucha *multispoof-test* połączenia generowane przez komponent **`conncheck`**[23], zaś pozostałe połączenia są kierowane do łańcucha *multispoof-sub*. Łańcuchy oraz połączenia między nimi zostały przedstawione na rysunku 3.18.

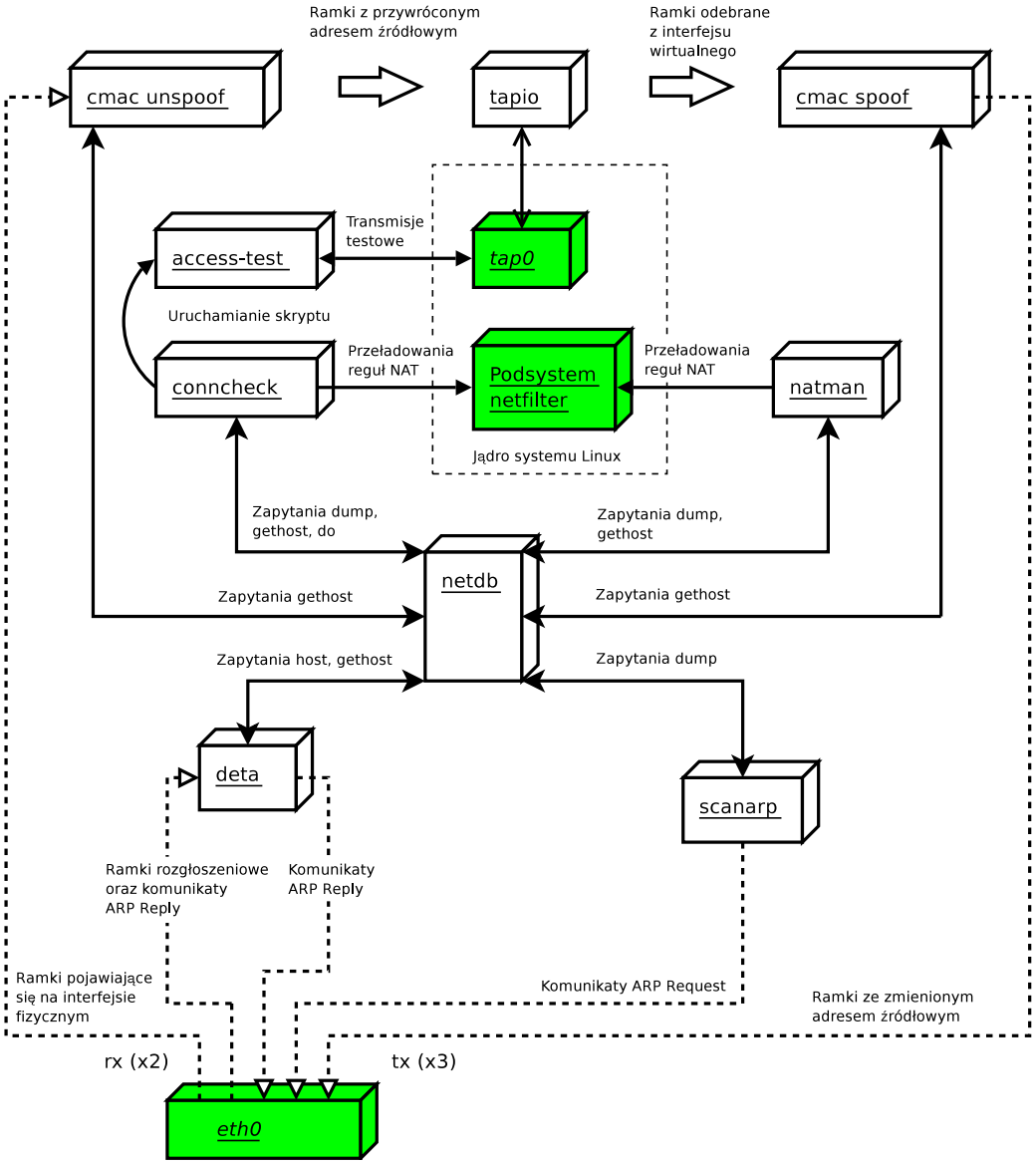
13. Uruchomienie komponentu **`natman`** (rozdział 3.5.2).

14. Oczekiwanie na utworzenie interfejsu *tap* przez komponent **`tapio`**.

15. Rejestracja adresu MAC interfejsu *tap* w bazie **`netdb`** w zmiennej *defmac*.

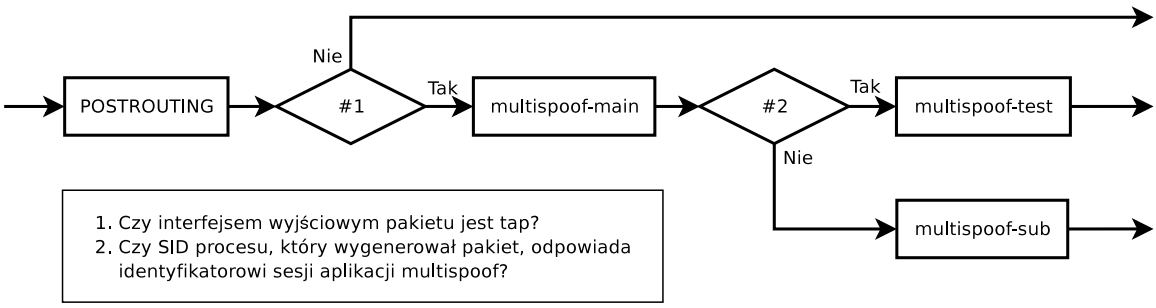
[22] Do odczytywania parametrów wykorzystywana jest komenda `getopts` dostępna w shellu.

[23] W rzeczywistości reguła ta dotyczy wszystkich połączeń, które inicjowane są przez dowolny z komponentów aplikacji. Jednak jedynym komponentem, który wykonuje połączenia sieciowe jest **`conncheck`**, a ściślej jego podproces **`access-test`**.



Rysunek 3.17: Komponenty składające się na aplikację *multispoof* i zależności między nimi.

16. Usunięcie adresów IP z interfejsu fizycznego. Pociąga to za sobą automatyczne usunięcie informacji o wszelkich trasach, które były związane z tym interfejsem, w tym o trasie domyślnej. Następnie skonfigurowanie interfejsu *tap*, tj. nadanie adresu IP, podniesienie interfejsu oraz dodanie trasy domyślnej. To ostatnie wymaga wprowadzenia statycznego wpisu ARP dla adresu IP bramy, ponieważ w obecnej wersji aplikacja *multispoof* nie pozwala na wysyłanie zapytań ARP.
17. Uruchomienie komponentu **conncheck** (rozdział 3.5.1).
18. Oczekiwanie na zakończenie procesów za pomocą funkcji *wait*.



Rysunek 3.18: Łańcuchy oraz reguły podsystemu `Netfilter` wykorzystywane przez aplikację *multispoof*.

“Regression testing? What’s that? If it compiles, it is good, if it boots up it is perfect.”

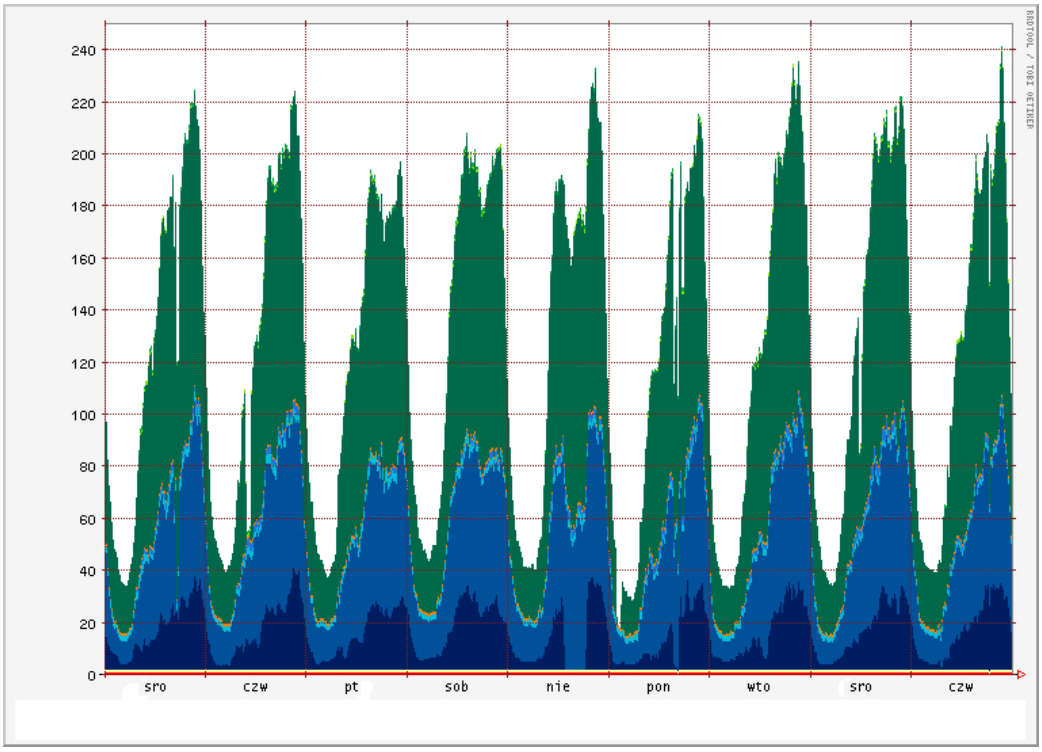
– *Linus Torvalds*, wypowiedź na liście dyskusyjnej linux-kernel, *1998*

Rozdział 4

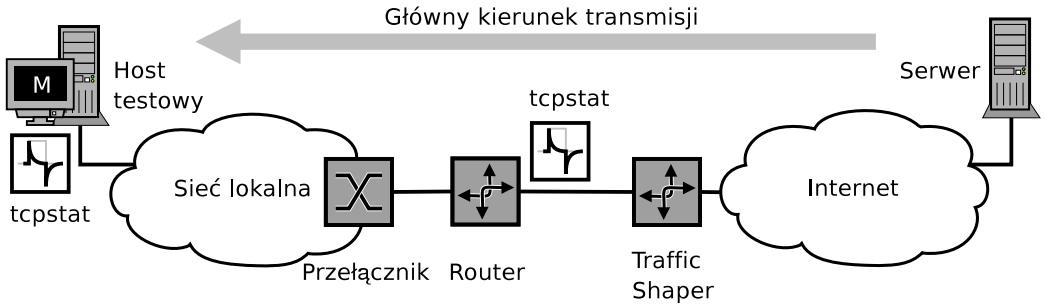
Testy

W celu oceny skuteczności aplikacji *multispoof* zostały przeprowadzone testy. Miały one miejsce w sieci osiedlowej zbudowanej z około 500 komputerów. Wszystkie komputery są podłączone do tej samej fizycznej sieci ethernetowej. Centralny przełącznik posiada agenta SNMP, co umożliwia monitorowanie wykorzystania portów ethernetowych pod względem ilości adresów MAC, które na nich się pojawiają. Każdy adres MAC jest utrzymywany w pamięci urządzenia przez 10 minut od ostatniego pojawienia się. Wykres 4.1 przedstawia liczbę aktywnych adresów MAC w funkcji czasu przy normalnym wykorzystaniu sieci (dane pochodzą z 9 dni). Można zaobserwować, że krzywa ma wyraźnie periodyczny charakter. Wynika to z faktu, że więcej komputerów korzysta z sieci za dnia, niż w nocy.

Nie wszystkie komputery działające w sieci mają dostęp do Internetu. Te które posiadają łączność z globalną siecią są zróżnicowane pod względem przydzielonej przepustowości maksymalnej. Większość hostów ma limit 80 kb/s, zaś pozostałe wykorzystują 200 kb/s lub 24 kb/s. Dwie ostatnie grupy mają podobną liczebność.



Rysunek 4.1: Liczba adresów MAC, które pojawiają się na portach przełącznika podczas normalnej pracy.



Rysunek 4.2: Układ badawczy używany podczas testów.

Celem testów było zbadanie jak dużą przepustowość uda się osiągnąć z wykorzystaniem aplikacji *multispoof* w badanej sieci. W tym celu w sieci umieszczono komputer[1] na którym była uruchomiona aplikacja oraz skrypt generujący ruch sieciowy. Skrypt wykorzystując program `netcat`[2] łączył się z serwerem usługi `chargen`, który znajdował sie za *traffic shaperem*, czyli routerem ograniczającym ruch dla poszczególnych hostów w sieci. Szybkość transmisji była mierzona przez program `tcpstat`[3], który został uruchomiony w dwóch kopiach, każda w innym miejscu sieci. Kopia uruchomiona na hoście testowym pozwalała badać przepustowość tego hosta, zaś kopia która działała na routerze służyła do mierzenia całego ruchu generowanego przez sieć. Wszystkie testy dotyczyły ruchu płynącego z Internetu do wewnątrz sieci. Schemat układu badawczego jest przedstawiony na rysunku 4.2.

`Chargen` (ang. *character generator*, generator znaków) jest standardową usługą testową, która domyślnie działa na porcie 19/TCP bądź UDP[4]. Natychmiast po podłączeniu się klienta, serwer zaczyna wysyłać do niego kolejne znaki ASCII. W testach usługę zapewniał program inetd[5]. Aby wytworzyć maksymalny ruch, skrypt uruchamiał 200 kopii programu `netcat`, co w efekcie dawało tyle samo niezależnych połączeń z usługą `chargen`. Tak duża liczba połączeń miała służyć temu, aby każdy z maksymalnie stu wykorzystywanych adresów otrzymał po dwa połączenia[6] dzięki rozkładaniu obciążenia (więcej na temat równoważenia obciążenia w podrozdziale 3.5.2). Następnie, skrypt wchodził w nieskończoną pętlę, w której co 2 sekundy najstarsza kopia programu `netcat` była usuwana z pamięci, a na jej miejsce uruchamiana była nowa. Dzięki temu co 2 sekundy wykonywane było nowe połączenie, zaś liczba procesów pozostawała stała. Ta rotacja miała zapewnić, aby w połączeniach wykorzystywane były wszystkie adresy. Zasadę działania skryptu przedstawia rysunek 4.3.

4.1 Pomiar maksymalnej przepustowości

Pierwszy test miał na celu sprawdzenie ile wynosi maksymalna przepustowość, którą może zaoferować *multispoof* w badanej sieci. Dlatego test został rozpoczęty o godzinie 7 rano, kiedy wykorzystanie sieci jest niewielkie. Dodatkowo, na podstawie otrzymanych od operatora danych wprowadzono do bazy programu adresy wszystkich zarejestrowanych komputerów. Rezultaty można zobaczyć na wykresie 4.4. Widać na nim, że aplikacja *multispoof* pozostawała nieaktywna przez 5 minut od momentu uruchomienia[7]. Następ-

[1] W testach wykorzystywany był komputer klasy PC z procesorem Pentium IV 2,8 GHz i 512 MB pamięci RAM, wyposażony w kartę sieciową opartą na układzie Realtek 8139.

[2] `netcat` łączy standardowe wejście i wyjście z gniazdem sieciowym, które tworzy przy uruchomieniu na podstawie podanych parametrów. Program można pobrać z `http://www.vulnwatch.org/netcat/`

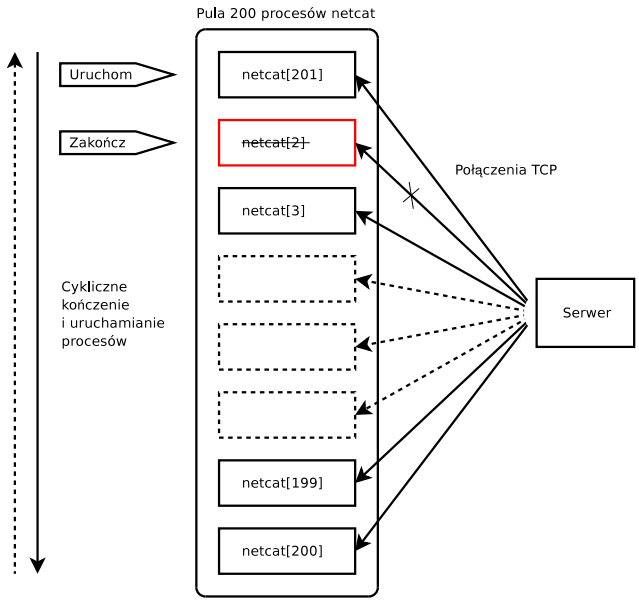
[3] `tcpstat` pozwala na analizę różnych parametrów sieciowych w czasie. W tym celu program wykorzystuje technikę podsłuchiwania sieci, omówioną w podrozdziale 2.1. Strona domowa projektu znajduje się pod adresem `http://www.frenchfries.net/paul/tcpstat/`.

[4] W testach była wykorzystywana wersja TCP.

[5] Inetd to tzw. superserwer. Jego działanie polega na nasłuchiwaniu na wielu portach TCP/UDP i przekazywaniu nadchodzących połączeń do podprocesów, w formie standardowych deskryptorów. Dzięki temu podprocesy nie muszą zajmować się obsługą zagadnień sieciowych. Jednak usługa `chargen`, ze względu na prostotę jest często wbudowywana w program inetd.

[6] Stan ten może być osiągnięty dopiero, kiedy liczba wykorzystywanych adresów osiągnie 100 i zbiór adresów będzie stabilny przez pewien czas. Przez okres stabilizacji połączenia mogą być przydzielane nierównomiernie, zaś adresy, które znajdą się na początku listy, będą faworyzowane. Może to powodować problemy, które zostaną opisane dalej w tym podrozdziale.

[7] Jest to domyślna wartość parametru określającego minimalny czas nieaktywności hosta (patrz tabela B.1).



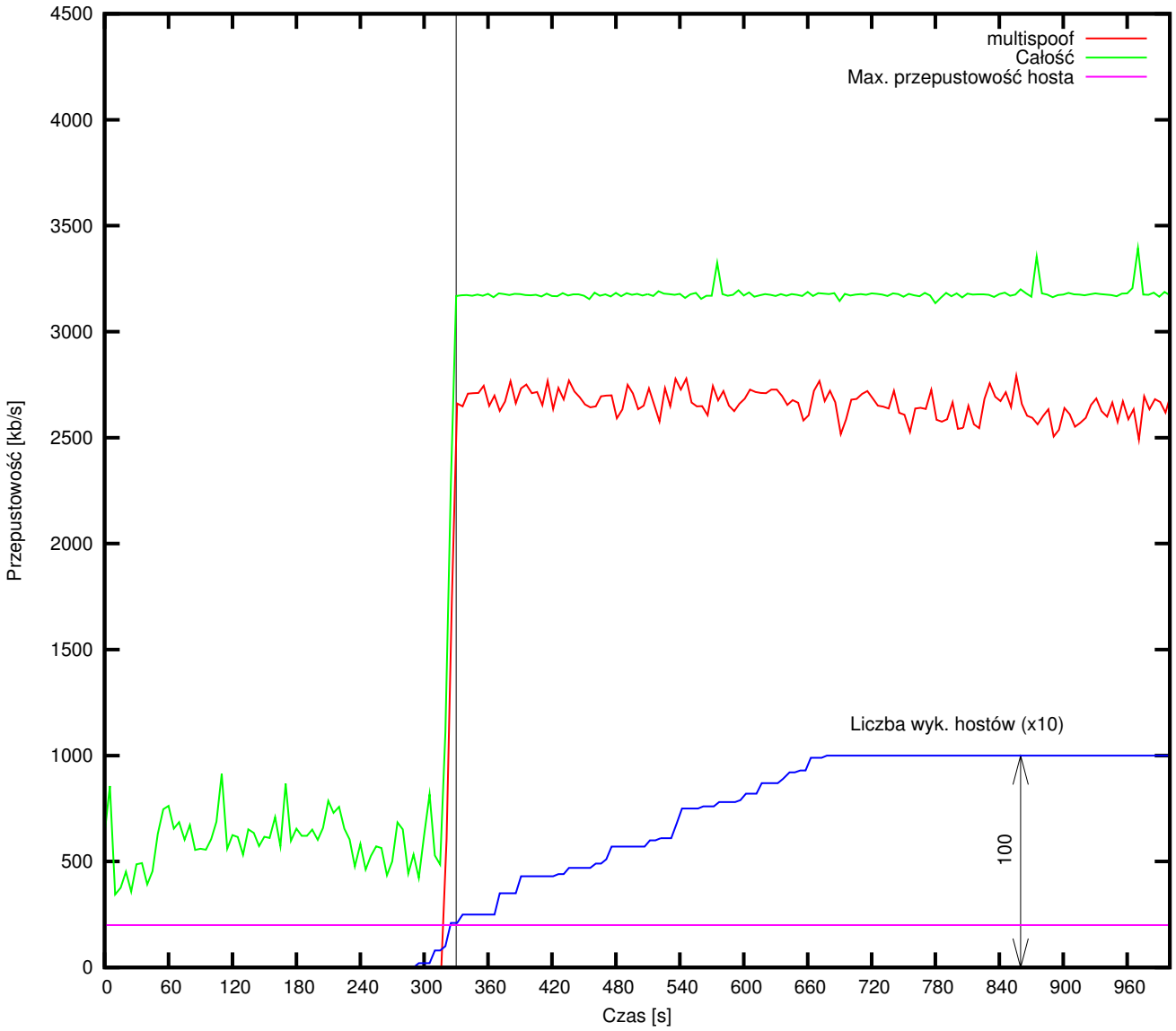
Rysunek 4.3: Schemat działania skryptu generującego ruch sieciowy.

nie, już po 30 sekundach od rozpoczęcia procesu podszywania się, szybkość transmisji osiągnęła 2,7 Mbit/s i na tym poziomie utrzymywała się przez dalszy czas. Jest to wynik wielokrotnie przewyższający (ponad 13 razy) maksymalną prędkość dostępną dla pojedynczego użytkownika. Moment osiągnięcia tej przepustowości oznaczony jest linią pionową. Jej przecięcie z krzywą wykorzystania hostów pokazuje, jak niewiele adresów było trzeba, aby osiągnąć tę prędkość – około 20. Dalsze zwiększanie tej liczby nie wpływało na przepustowość. Być może działo się tak, ponieważ całe dostępne pasmo operatora zostało zajęte, o czym świadczyć może zielona linia. Przed rozpoczęciem działania aplikacji *multispoof* całościowa przepustowość charakteryzowała się wysokim odchyleniem standardowym, co jest naturalne dla ruchu internetowego. Od momentu uaktywnienia się aplikacji przepustowość wzrosła, ale odchylenie standardowe wyraźnie zmalało, co jest sygnałem silnego wysycenia łącza. Krzywa wykorzystania hostów zatrzymała się na 100 adresach. Jest to związane z limitem nakładanym przez jądro systemu oraz narzędzie `iptables`, który zostanie omówiony w dalszej części tekstu.

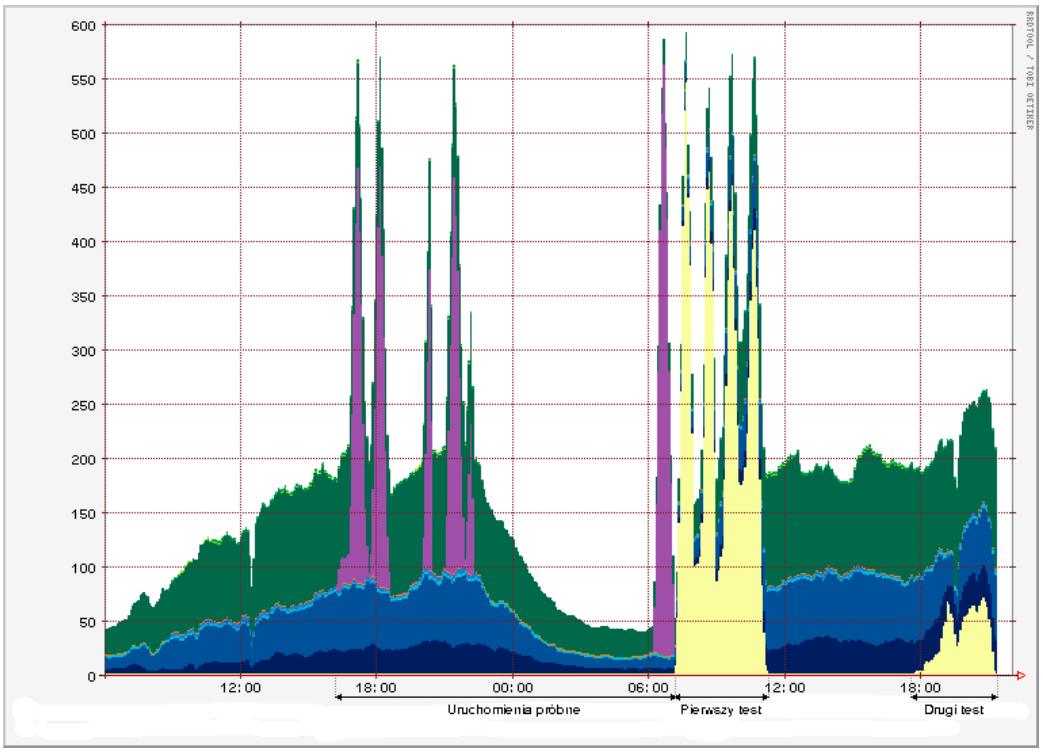
Wpływ przeprowadzonych testów na działanie sieci widać na wykresie 4.5, który przedstawia liczbę adresów MAC pojawiających się na portach przełącznika w czasie dwóch dni jego pracy. Szczególnie wyraźnie widać wpływ pierwszego testu (godzina 7 rano, kolor jasnożółty). Podczas jego trwania liczba adresów MAC pojawiających się w sieci momentami była ponad dwukrotnie wyższa niż podczas normalnej pracy sieci w godzinach szczytu. Charakterystyczne są skoki krzywej wykorzystania adresów pojawiające się równo co godzinę. Przedstawiają one testy łączności, które wykonuje aplikacja w celu sprawdzenia czy każdy nieaktywny host ma łączność z Internetem. Dla każdego sprawdzanego hosta aplikacja wysyła kilka pakietów ze sfałszowanym adresem źródłowym. Adresy te są przechwytywane przez przełącznik i zapisywane w jego pamięci, która jest odzwierciedlana na wykresie. Można zauważyć także, iż liczba wykorzystywanych adresów w teście była z reguły większa niż 100, co pokrywa się ze wspomnianym limitem.

4.2 Pomiar wydajności w warunkach typowych

Celem przyświecającym drugiemu testowi było określenie jak aplikacja *multispoof* zachowuje się w warunkach typowych. Dlatego baza **`netdb`** została wyczyszczona. Test rozpoczął się o godzinie 18, ze względu na fakt, że wtedy ruch w sieci jest nasilony i możliwe jest zgromadzenie dużej liczby adresów. Wyniki testu przedstawia wykres 4.6. Od razu można zauważyć, że uzyskana przepustowość nie jest wysoka, mimo, że ponad dwa razy większa od maksymalnej szybkości transmisji pojedynczego hosta i około sześć razy wyższa niż prędkość średnia. Test w małym stopniu wpłynął na ruch sumaryczny, którego stanowił średnio piątą



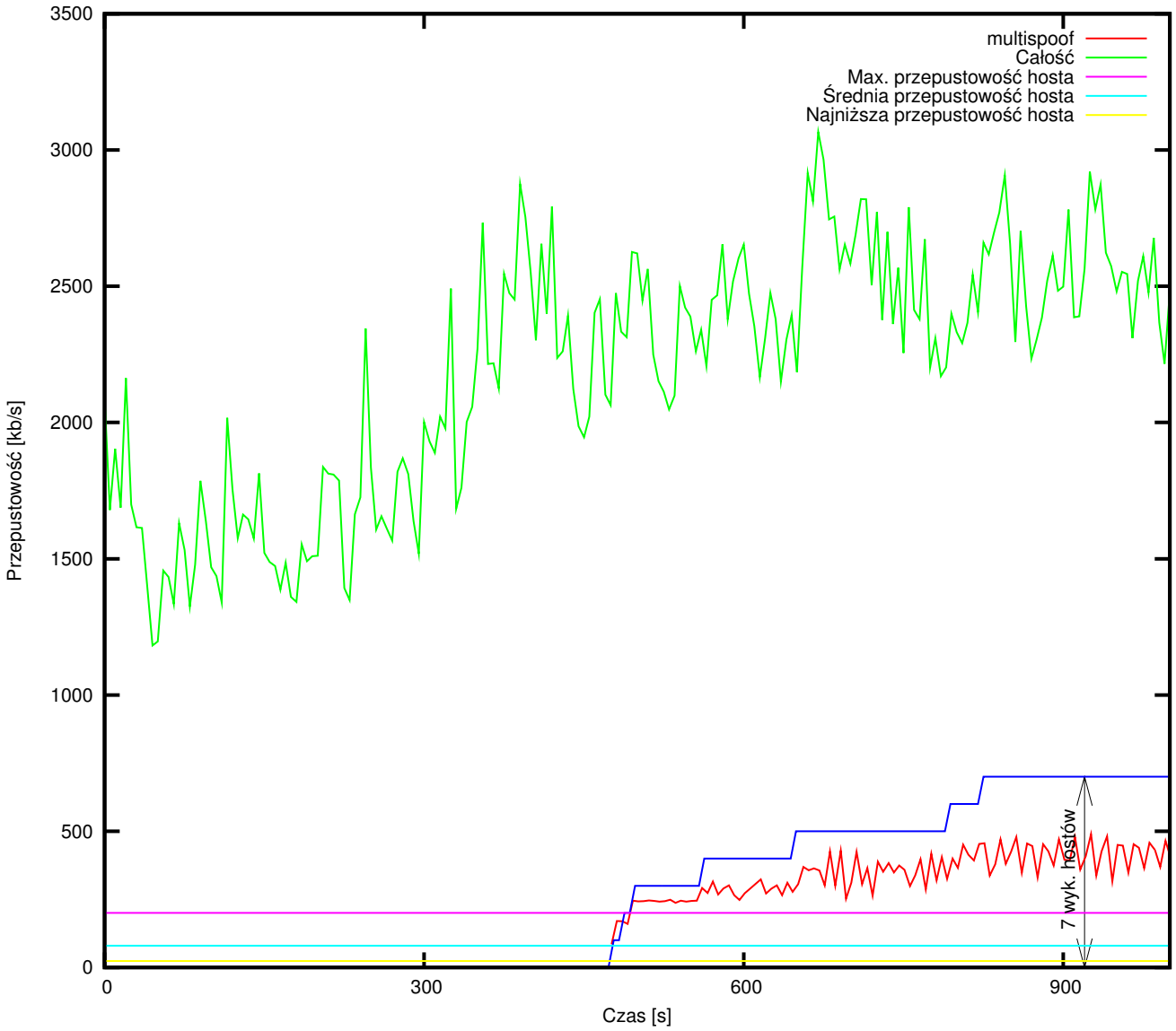
Rysunek 4.4: Wyniki pierwszego testu aplikacji *multispoof*.



Rysunek 4.5: Wpływ testów na liczbę adresów MAC rejestrowanych na przełączniku.

część. Tak słabe (w porównaniu z testem poprzednim) wyniki mają kilka przyczyn. Pierwszą z nich jest pusta baza adresów, z którą program zaczynał działanie. Czas, który upłynął od uruchomienia do rozpoczęcia aktywności był stosunkowo długi (około 8 minut), ponieważ adresy, które program kolekcjonował, były wtedy używane przez hosty fizyczne. Dlatego też krzywa wykorzystania adresów rośnie powoli. W 15. minucie od momentu rozpoczęcia działania, aplikacja wykorzystuje zaledwie 7 adresów. O ile liczba ta zwiększa się, przepustowość utrzymuje się na stałym poziomie. Najpewniej wynika to z faktu, że hosty znajdujące się w bazie aplikacji są zapisane w stałej kolejności. Połączenia sieciowe skryptu testowego są wykonywane zgodnie z tą kolejnością (wynika to z zastosowanego algorytmu równoważenia obciążenia). Dodatkowo, częste przeładowania reguł rozkładania obciążenia powodują, że faworyzowane są adresy znajdujące się na początku. Przypisanie nadmiernej liczby połączeń do adresów początkowych (szczególnie jeśli hosty, do których one należą, mają niską przepustowość) skutkuje pogorszeniem szybkości transmisji[8]. Test trwał przez kilka godzin (co widać na wykresie 4.5), jednak choć liczba wykorzystywanych adresów stopniowo się zwiększała, uzyskiwana przepustowość pozostawała bez zmian. Oprócz wspomnianego problemu związanego z nierównomiernym przydziałem połączeń, znaczenie miał także fakt, że komponent **`netdb`** w obecnej wersji posiada błąd skutkujący tzw. wyciekami pamięci. Powodują one, że aplikacja w miarę upływu czasu działa coraz gorzej[9], zaś w pewnej chwili jej działanie jest przerywane przez system operacyjny, ze względu na zbyt wysokie zużycie pamięci.

[8] Możliwe rozwiązanie tego problemu będzie przedstawione w rozdziale 6.
[9] Błąd zostanie poprawiony w następnej wersji aplikacji *multispoof*.



Rysunek 4.6: Wyniki drugiego testu aplikacji *multispoof*.

"We make poor trade-offs, giving up much in exchange for very little security. We sorround ourselves with security countermeasures that give us a feeling of security rather than the reality of security. We deceive ourselves by believing in security that doesn't work."

– *Bruce Schneier,* Beyond Fear

Rozdział 5

Metody detekcji i prewencji

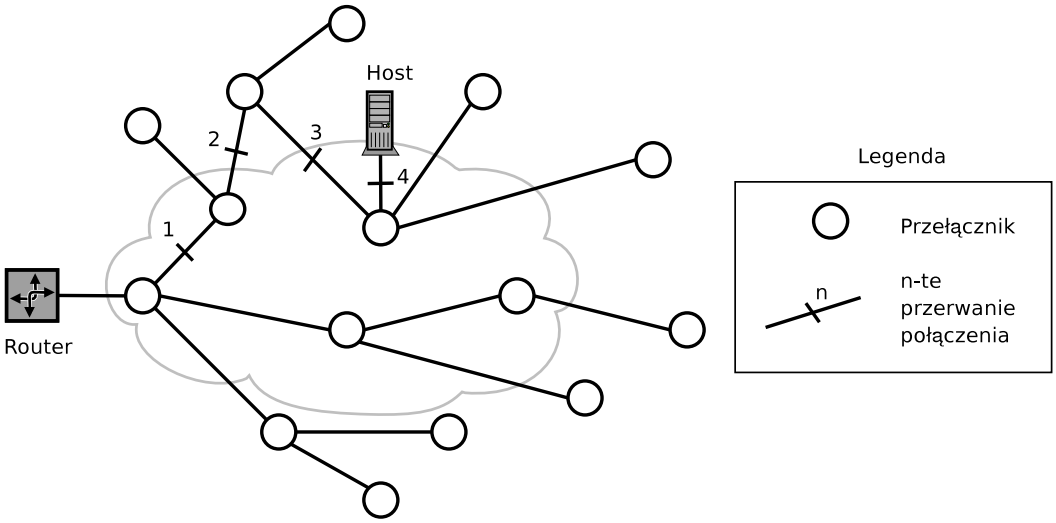
Techniki polegające na zmianie adresów IP oraz MAC, opisane wraz z metodami pomocniczymi w podrozdziale 1.3, stanowią zagrożenie, przed którym sieci lokalne powinny być chronione. W rozdziale 3 przedstawiona została koncepcja oraz implementacja programu *multispoof*, opracowanego w celu praktycznej demonstracji problemu kradzieży usługi dostępu do Internetu. Istnieją dwa rodzaje metod, które pozwalają w różnym stopniu zmniejszyć ryzyko które niesie możliwość podszywania się w celu uzyskiwania nieautoryzowanego dostępu do sieci zewnętrznej. Najskuteczniejsze są metody proaktywne, lecz koszt wdrożenia odpowiedniej infrastruktury może być wysoki. Metody reaktywne są tańsze, jednak nie tak skuteczne i na dłuższą metę mogą być mniej opłacalne od środków zapobiegawczych. Oba rodzaje metod zostaną omówione w tym rozdziale.

5.1 Metody reaktywne

Metody reaktywne to takie, które nie służą zapobieganiu, lecz koncentrują się na detekcji ataków. Czasami wystarczy, że administrator wykryje, że w sieci lokalnej, która znajduje się pod jego nadzorem, ktoś podszywając się pod host nieaktywnego użytkownika, nielegalnie korzysta z dostępu do Internetu. Jeśli dodatkowo sieć obejmuje mały obszar, zaś administrator posiada łatwy dostęp do przełączników sieciowych, to fizyczne zlokalizowanie oszusta, choć pracochłonne – jest możliwe. Administrator jest w gorszej sytuacji, jeśli osoba robi to np. w godzinach nocnych, ponieważ odnalezienie miejsca, w którym działa nieuczciwy użytkownik jest możliwe tylko wtedy, kiedy jest on aktywny. Fizyczne lokalizowanie komputera oszusta (przedstawione na rysunku 5.1) polega na selektywnym wyłączaniu przełączników i testowaniu (za pomocą wybranej metody), czy proces fałszowania adresów nadal trwa. Jeśli tak, to oznacza to, że szukanego hosta nie ma w sieci, która znajduje się za przełącznikiem. Jeśli test wykaże brak aktywności oszusta, to należy przypuszczać, że host znajduje się w sieci za wyłączonym przełącznikiem i przejść do kolejnego, który znajduje się za nim. Złożoność tego procesu zależy od rozmiarów i topologii sieci (czym więcej przełączników połączonych szeregowo tym trudniej). Przedstawiona metoda lokalizacji dotyczy sieci zbudowanych w oparciu o niezarządzalne przełączniki lub koncentratory. Jeśli sieć bazuje na urządzeniach *inteligentnych* to detekcja jest prostsza, bo nie jest wymagana fizyczna interwencja w działanie sieci. Więcej o tym w podrozdziale 5.1.8.

Kilka pierwszych z przedstawionych metod będzie dotyczyło wykrywania nielegalnego podszywania się z wykorzystaniem aplikacji *multispoof*. Są to metody skuteczne i zazwyczaj proste w implementacji, lecz ograniczają się do detekcji obecnej wersji aplikacji, nie rozwiązując problemu fałszowania adresów w ogólności. Zostały jednak zaprezentowane dla kompletności oraz ze względu na to, że demonstrują słabości programu, które mogą zostać poprawione w następnych wersjach (patrz rozdział 6). Ponieważ są to metody o charakterze tymczasowym, będą określane mianem słabych.

Kolejne metody będą koncentrowały się na detekcji procesu podszywania się pod adresy wielu nieaktywnych hostów na raz. Choć zjawiska towarzyszące temu procesowi nie są tak wyraźne, jak te związane z aktywnością aplikacji *multispoof*, to ich wykrycie jest możliwe. Metody te nazywane będą silnymi, ponieważ pozwalają na detekcję efektów ubocznych nadużyć związanych z podszywaniem się, które są wspólne dla każdej implementacji tego rodzaju ataków.



Rysunek 5.1: Proces lokalizowania hosta w sieci Ethernet.

5.1.1 Wykrywanie skanowania ARP

Jest to przykład słabej metody, która wykorzystuje fakt, że aplikacja *multispoof*, a dokładniej komponent **`scanarp`**, periodycznie wysyła zapytania ARP o adresy wszystkich hostów, które są zarejestrowane w bazie komponentu **`netdb`**. Zapytania są wysyłane jedno po drugim, domyślnie co 60 sekund. Powoduje to, że w sieci można obserwować występujące cyklicznie fale zapytań ARP. Jest to nietypowe zachowanie sieci i może być łatwo wykrywane, nawet bez użycia specjalnie do tego celu napisanego programu. W poniższym przykładzie zastosowano narzędzie `tcpstat`, które co sekundę wyświetla liczbę ramek ARP, które pojawiały się przez ten czas na interfejsie sieciowym.

```
# tcpstat -i eth0 -f arp -o "%n " 1
32 40 28 37 44 25 44 23 33 140 38 37 22 22 25 33 28 38 35 23 21 25 33 49 77 48 3
8 29 39 40 27 39 31 22 25 29 32 31 26 32 42 22 27 33 31 32 41 28 22 31 34 21 32
63 75 33 36 41 23 31 36 33 27 33 34 37 38 153 20 22 32 21 25 35 29 31 28 31 39 2
1 25 36 17 40 98 42 38 45 39 25 31 29 30 25 26 32 26 26 37 38 21 34 43 16 24 32
32 21 36 40 40 25 35 74 63 58 38 29 27 38 34 39 41 47 27 24 101 92 24 23 35 20 2
0 27 25 42 33 36 27 25 12
```

Liczby zaznaczone tłustą czcionką są nietypowo duże i świadczą o tym, że co minutę gwałtownie wzrasta liczba pakietów ARP, które pojawiają się w sieci. Może to oznaczać, że działa w niej program *multispoof*.

Choć badanie aktywności hostów w sieci jest niezbędnym elementem działania każdego programu, który wykorzystuje podszywanie sie pod nieaktywne adresy, to proces monitorowania sieci można zaimplementować inaczej, tak, aby nie było możliwe jego wykrycie tą metodą. Więcej na ten temat w rozdziale 6.

5.1.2 Wykrywanie testowania łączności

Podobną metodą jest technika bazująca na zachowaniu komponentu **`conncheck`**, który domyślnie wysyła pakiety testowe co godzinę. Pakiety są wysyłane spod wszystkich adresów, które aplikacja ma zarejestrowane w bazie i które są nieaktywne w sieci. Aktualna wersja aplikacji do testowania łączności wykorzystuje skrypt **`access-test`**, który wykonuje zawsze ten sam test, mianowicie rozpoczęcie połączenia TCP na porcie 443 z hostem `www.mbank.com.pl`. Jeśli administrator analizując ruch sieciowy zauważy, że takie połączenia odbywają się regularnie to może przypuszczać, że w sieci działa aplikacja *multispoof*.

Inną metodą detekcji bazującą na tej samej słabości jest sprawdzanie, czy każdy host, który wysyłał jakieś dane, po kilku sekundach odpowiada na zapytania ARP. Aplikacja *multispoof* przez czas działania testu

odpowiada na nie, jednak jeśli test wypadnie niepomyślnie, to jest on natychmiast blokowany, co obejmuje także ignorowanie zapytań ARP.

Obie metody można zaliczyć do klasy słabych, ponieważ proces testowania łączności może być przeprowadzany w inny, trudniejszy do wykrycia sposób. W rozdziale 6 przedstawiono możliwe techniki utrudniające detekcję z wykorzystaniem tej metody.

5.1.3 Standardowe transmisje

Kolejną słabą metodą detekcji jest analizowanie ruchu sieciowego pod kątem standardowych transmisji. Po włączeniu, komputer wysyła zazwyczaj kilka pakietów DHCP. Przed rozpoczęciem komunikacji z siecią zewnętrzną wysyła także zapytanie ARP o adres fizyczny routera. Popularne systemy operacyjne firmy Microsoft z domyślnie uaktywnionym klientem usługi udostępniania plików i drukarek co pewien czas wysyłają specyficzne pakiety rozgłoszeniowe UDP. Aplikacja *multispoof* nie generuje wymienionych transmisji. Z analizy ruchu sieciowego można wnioskować, że jeśli dany adres jest aktywny, zaś nie wykryto powiązanych z nim standardowych transmisji, to może on być wykorzystywany przez tę aplikację.

Wykrywanie braku standardowych transmisji może wymagać napisania prostego programu, który będzie analizował ruch sieciowy i uaktualniał listę aktywnych hostów, oraz sprawdzał, czy każdy z nich posiada skojarzone standardowe transmisje.

Opisana metoda nie należy do silnych ze względu na to, że standardowe transmisje można symulować, co jest opisane dokładniej w rozdziale 6.

5.1.4 Skanowanie i monitorowanie hostów

Kolejna słaba metoda bazuje na tym, że w obecnej wersji aplikacja *multispoof* blokuje wszelkie połączenia sieciowe, które są inicjowane z sieci. Adresy wykorzystywane przez aplikację nie odpowiadają na zapytania ICMP Echo oraz nie udostępniają standardowych usług sieciowych. Administrator może periodycznie skanować hosty, które znajdują się w sieci, celem sprawdzenia, czy odpowiedzi na pewne typy zapytań nie zmieniają się. Jeśli na przykład host w dzień udostępnia jakąś usługę sieciową, zaś w nocy nie, a sytuacja taka się powtarza, to może oznaczać, że w sieci działa aplikacja typu *multispoof*.

Podobną metodą jest pasywne wykrywanie systemu operacyjnego na podstawie danych, które transmituje dany host[1]. Podejrzenia może wzbudzić sytuacja, kiedy wykryty zostanie system operacyjny inny, niż zazwyczaj rozpoznawany dla badanego hosta. Może to jednak także oznaczać, że użytkownik tego komputera po prostu korzysta z więcej niż jednego systemu operacyjnego.

Opisane metody trudno zaklasyfikować do jednej z grup opisanych na początku tego podrozdziału. Skanowanie wykorzystuje ewidentną słabość aktualnej wersji aplikacji *multispoof*, dlatego może zostać uznane za słabe. Jednak zabezpieczenie się przed skanowaniem (opisane w rozdziale 6) jest możliwe tylko do pewnego stopnia. Regularne monitorowanie wykorzystywanych systemów operacyjnych także pozwala ze stosunkowo wysokim prawdopodobieństwem odpowiedzieć na pytanie, czy w sieci występują nadużycia związane z podszywaniem się.

5.1.5 Hosty pułapki

Następną metodą jest wykorzystanie komputerów, które znajdują się pod pełną lub częściową kontrolą administratora sieci. Administrator konfiguruje specjalny host, którego jedynym zadaniem jest obecność w sieci. Ta obecność powinna być zauważalna dla innych, dlatego host musi generować transmisje sieciowe. Im aktywność tego hosta jest bardziej zbliżona do prawdziwej, tym metoda jest silniejsza. Co pewien czas komputer jest wyłączany na pewien. W tym czasie administrator analizuje ruch sieciowy. Jeśli adres tego hosta pojawi się w sieci, jest to pewna oznaka, że w sieci działa oszust, którego komputer podszywa się pod adres hosta-pułapki.

[1] Aktywne rozpoznawanie systemu operacyjnego polega na inicjowaniu połączeń sieciowych z badanym hostem [Gutkowski, 2004]. Ponieważ host z uruchomioną aplikacją *multispoof* nie akceptuje nowych połączeń przychodzących, niezbędne jest wykorzystanie metod pasywnych [Hon02] [Tomaszewski *et al.*, 2005]. Do tego celu można wykorzystać narzędzie p0f, które można pobrać z `http://lcamtuf.coredump.cx/p0f.shtml`. Jedną z ciekawszych metod pasywnych jest badanie jakości generatora numerów sekwencyjnych ISN, opisane w [Zalewski, 2001] i [Zalewski, 2002]. Inną techniką jest analiza uskoku zegara, która pozwala na identyfikację zdalnego hosta na poziomie sprzętu [Kohno *et al.*, 2005].

Metodą zbliżoną jest korzystanie z pomocy zaufanych użytkowników sieci, którzy informują administratora o wszystkich włączeniach i wyłączeniach swoich komputerów. Może to być zautomatyzowane przez wykorzystanie specjalnego oprogramowania. Jeszcze jedną wariacją jest udostępnienie użytkownikom informacji o tym, kiedy ich komputery wykazywały aktywność sieciową i nakazanie zgłaszania wszelkich niezgodności.

Metody wykorzystujące hosty-pułapki są silne, zaś z wykorzystaniem odpowiedniego oprogramowania stosunkowo wygodne w użyciu.

5.1.6 Korelacja transmisji sieciowych

Wykorzystywanie procesu podszywania się w codziennej aktywności sieciowej powoduje, że mogą powstawać pewne odstępstwa od normy w zakresie transmisji. Część aplikacji, które komunikują się z Internetem, wykorzystuje w tym celu kilka połączeń (zazwyczaj z użyciem protokołu TCP). Jeśli komputer oszusta podszywa się pod wiele hostów, będą się zdarzać sytuacje, kiedy kilka połączeń jednej aplikacji zostanie przydzielonych do różnych adresów IP[2]. W wielu przypadkach zależności między połączeniami są znane (kolejność, czas który upływa między kolejnymi transmisjami, liczba bajtów i pakietów), zaś jeśli połączenia nie są szyfrowane, można wydobyć z nich dodatkowe informacje. Jako przykład można podać korzystanie z programów, które przy starcie łączą się z serwerem (np. w celu zalogowania lub pobrania aktualizacji), a następnie pobierają z sieci i wyświetlają reklamę. Do tej klasy można zaliczyć aplikacje, które oferują dostęp do bezpłatnej usługi w zamian za dostarczanie treści marketingowych, jak komunikatory internetowe lub darmowe oprogramowanie antywirusowe. Jeśli oszust podszywa się pod wiele hostów na raz, to logowanie i pobieranie reklamy, jako dwa niezależne połączenia, mogą odbywać się z użyciem dwóch różnych adresów IP. Między tymi transmisjami istnieje silna zależność czasowa, mianowicie pobranie reklamy następuje natychmiast po zalogowaniu. Dodatkowo, w przypadku wielu programów zdarza się, że identyfikator użytkownika przesyłany przy logowaniu, jest wysyłany także przy pobieraniu reklamy (np. za pomocą mechanizmu *cookies*).

Aby wykorzystać te informację w celu wykrywania fałszowania adresów, niezbędne jest stworzenie specjalnego oprogramowania, które będzie poddawać je analizie oraz korelować transmisje z różnych adresów i powiadamiać administratora w przypadku wykrycia aktywności tej samej kopii programu na wielu adresach. Metoda ta wymaga dużych nakładów początkowych, związanych z pisaniem oprogramowania i jego wdrażaniem w sieci, oraz stałych kosztów utrzymywania i aktualizacji bazy wykrywanych aplikacji. W zamian otrzymuje się silną metodę detekcji, którą trudno wprowadzić w błąd.

5.1.7 Moment pojawienia się hosta w sieci

Kolejna metoda opiera się na nietypowym efekcie, który powstaje w momencie włączenia się do sieci komputera, którego adres był do tej pory wykorzystywany w procesie podszywania się. Jeśli oszust nie chce zostać zauważony, musi stosować mechanizm unikania konfliktów w sieci, np. ten zaimplementowany w komponencie **`deta`** aplikacji *multispoof*. Polega on na podsłuchiwaniu sieci i w razie odebrania ramki pochodzącej od hosta, którego adres jest używany w procesie podszywania się, jego dalsze wykorzystanie jest natychmiast blokowane. Ten moment może być wykorzystany do wykrycia programu *multispoof*, ponieważ mocno odbiega od typowego zachowania sieci. Normalna aktywność hosta w pewnej chwili ulega natychmiastowemu przerwaniu, a w tym samym czasie pojawiają się nowe transmisje, świadczące o uruchamianiu się komputera (takie jak zapytania DHCP, ARP, DNS itd.). Przypomina to sytuację nagłego zawieszenia się komputera, po którym następuje jego wyłączenie, ponowne włączenie, uruchomienie systemu operacyjnego i konfiguracja sieci, z tym, że czas od wyłączenia do konfiguracji sieci jest skrócony do zera. Jest to moment charakterystyczny i napisanie aplikacji, która wykrywa go na podstawie ruchu sieciowego i bieżących połączeń nie jest trudne. Jest możliwe także napisanie programu, który symuluje ruch sieciowy włączającego się komputera,

[2] Można się zastanawiać, czy w takich sytuacjach aplikacja sieciowa będzie działać prawidłowo. Jeśli operator wykorzystuje w swojej sieci adresy publiczne, tzn. każdy użytkownik posiada zewnętrzny adres IP, to wiele aplikacji może nie działać poprawnie, ponieważ zdalny serwer może odrzucać połączenia z innych adresów. Jednak częściej zdarza się, że operator sieci wykorzystuje tylko jeden adres publiczny, natomiast hosty w sieci mają adresację wewnętrzną, która podlega translacji NAT. W takiej sytuacji, mimo różnych adresów wewnętrznych, zdalne serwery będą komunikowały się z jednym, publicznym adresem IP, co nie będzie miało negatywnego wpływu na działanie aplikacji.

a następnie bada reakcje transmisji, które wykorzystują ten adres. Jest to efektywna, silna metoda detekcji, przed którą niełatwo jest się zabezpieczyć.

5.1.8 Analiza wykorzystania portów przełączników

Jeśli w sieci operatora występują zarządzalne przełączniki, które udostępniają odpowiednie funkcje monitorujące, to wykrywanie fałszowania adresów IP i MAC można oprzeć na obserwacji ich działania. Zasadą jest, że czym więcej takich urządzeń, tym wykrywanie jest prostsze. Urządzenia oferują różne funkcje, dlatego należy dostosować do nich metody detekcji:

- Najprostsze urządzenia pozwalają na określenie dla każdego portu ile ramek i/lub pakietów zostało wysłanych i odebranych przez dany port. Jeśli na jakimś porcie występuje nietypowo wysokie natężenie ruchu oraz można stwierdzić, że w większości są to transmisje internetowe[3], to być może z tego portu korzysta host oszusta, który podszywa się pod nieaktywne adresy.

- Niektóre proste przełączniki i koncentratory oferują funkcje sprawdzania czy z danego portu korzysta jeden host, czy jest ich wiele. Bardziej zaawansowane urządzenia pozwalają na dokładne określenie listy adresów MAC, które w danej chwili w pamięci switcha są przypisane do danego portu. W przypadku, gdy w sieci jest aktywny oszust, analiza portów switcha może wykazać, że port, z którego do tej pory korzystał jeden lub kilka hostów, jest teraz wykorzystywany przez dużą ich liczbę. Można także zauważyć, że są to adresy, które zazwyczaj pojawiają się na innych portach lub przełącznikach.

- Pewne urządzenia oferują możliwość ustalenia dla każdego portu, czy mają na nim prawo pojawiać się różne adresy MAC, czy tylko jeden. Jeśli na porcie zaklasyfikowanym do drugiej kategorii pojawi się wiele adresów MAC, to switch wysyła komunikat do aplikacji nadzorującej pracę sieci (z wykorzystaniem protokołu SNMP).

Wszystkie wymienione techniki można zautomatyzować, co może znacznie uprościć wykrywanie. Opisane metody są silne, jednak mają kilka wad:

- Są złożone. Jeśli sieć posiada wiele przełączników, to konieczne jest monitorowanie ich wszystkich oraz synteza uzyskanych informacji.

- Jakakolwiek zmiana w konfiguracji sprzętowej sieci (wymiana lub uruchomienie nowego przełącznika, zmiana połączeń między urządzeniami, przepięcie użytkownika do innego portu) musi być uwzględniona w systemie monitorującym. Wiąże się to z powstaniem nowego, stałego kosztu administracji.

- Zarządzalne przełączniki są drogie.

5.2 Środki prewencyjne

Ten podrozdział koncentruje się na metodach proaktywnych. Polegają one na zmianie środowiska sieciowego w taki sposób, aby kradzież usługi, jaką jest dostęp do Internetu, nie była możliwa z wykorzystaniem podszywania się. Aby przegląd metod ochronnych był kompletny, na początku zostanie zaprezentowane podejście o charakterze tymczasowym, które polega na blokowaniu aplikacji *multispoof*. Następnie wprowadzony zostanie model dostępu do usługi w sesjach, wzorowany na systemie zabezpieczeń wirtualnych sieci prywatnych, który będzie służył do oceny efektywności poszczególnych metod prewencyjnych. Pod koniec rozdziału zaprezentowane zostaną techniki sprzętowe.

Wydaje się, że właśnie zabezpieczenia na poziomie sprzętu sieciowego oraz metody wykorzystujące VPN stanowią najskuteczniejsze zabezpieczenie przez podszywaniem się. Jednak administratorzy, którzy planują wprowadzić zabezpieczenia w swoich sieciach, przed wyborem metody powinni przeprowadzić ocenę zagrożeń i podatności oraz kosztów wdrożenia każdej z technik prewencyjnych, co pomoże podjąć właściwą decyzję.

[3] Na przykład na podstawie danych z innych przełączników sieciowych, analizy ruchu na routerze itd.

5.2.1 Zagłuszanie

Najprostszym sposobem na uniemożliwienie działania aplikacji *multispoof* jest symulacja aktywności hostów, które w danej chwili są wyłączone. Ponieważ aplikacja w aktualnej wersji podszywa się tylko pod hosty, które nie wykazują objawów obecności w sieci przez jakiś czas, ten sposób całkowicie uniemożliwia jej działanie. Program, który realizowałby to zadanie, można zaimplementować nawet jako skrypt shellowy, wykorzystując program `arping`[4] do badania aktywności hostów, oraz narzędzie `ifconfig` do definiowania dodatkowych adresów na interfejsie. Skrypt co zdefiniowany czas sprawdzałby, które hosty nie są aktywne[5], a następnie definiowałby je jako aliasy IP dla interfejsu sieciowego. Resztą zajmowałoby się jądro systemu – udzielałoby ono odpowiedzi na zapytania ARP o te adresy. Aplikacja *multispoof* co pewien czas wysyła takie zapytania, zaś otrzymanie odpowiedzi traktuje jako dowód na obecność danego hosta w sieci, co powoduje, że nie jest on uwzględniany w procesie podszywania się.

W momencie pojawienia się fizycznego hosta w sieci, skrypt wykrywałby to i alias dla jego adresu byłby usuwany. Aby zapobiec konfliktom IP, skrypt powinien zostać wzbogacony o wywołania `arptables`[6], które blokowałyby udzielanie odpowiedzi ARP, jeśli zapytanie dotyczyłoby adresu, który je wygenerował[7].

Innym sposobem na symulowanie aktywności hostów w sieci jest uruchomienie drugiej kopii aplikacji *multispoof* przez administratora (sic!). Jeśli do bazy aplikacji zostanie wprowadzona lista zarejestrowanych komputerów, zaś program będzie działał przez cały czas, to wszystkie zadania przedstawionego powyżej skryptu będą przezeń spełniane. Dodatkowym atutem tego rozwiązania jest automatyczna aktualizacja bazy hostów, wykonywana przez aplikację *multispoof* w pełni samodzielnie.

Opisana metoda posiada jednak szereg słabości, które mogą być wykorzystane w celu obniżenia jej skuteczności do zera. Przyszłe wersje programu *multispoof* mogą implementować mechanizmy służące do przełamywania tego zabezpieczenia.

5.2.2 Model dostępu do usługi w sesjach

Aby programowe zabezpieczenia przed podszywaniem się w sieci lokalnej były skuteczne, muszą opierać się na sesjach. Sesja rozpoczyna się logowaniem, zaś jej zakończenie wymaga wylogowania. Użytkownik może korzystać z Internetu wyłącznie podczas trwania jego sesji, która powinna spełniać następujące trzy warunki:

1. Logowanie musi być przeprowadzone w sposób bezpieczny, tzn. wykorzystywany protokół powinien zapewniać ochronę przed przechwyceniem informacji używanych do uwierzytelniania (jak np. nazwa użytkownika i hasło) oraz atakami przez powtórzenie (ang. *replay attack*), które polegają na wykorzystaniu przechwyconej transmisji[8] do ponownego zalogowania.

2. W trakcie trwania sesji, wszystkie transmisje wykonywane w jej obrębie powinny być uwierzytelnione. Choć często jest to związane z ich szyfrowaniem, nie jest to jedyny sposób.

3. Sesja może być zakończona tylko, jeśli nastąpi jedno z dwóch zdarzeń: użytkownik wyloguje się lub zostanie wylogowany wskutek braku aktywności przez pewien czas. Świadome wylogowanie z sesji jest ważne, ponieważ jeśli ono nie nastąpi, sesja może zostać przejęta przez oszusta. Muszą także zostać podjęte kroki, które uniemożliwiają przerwanie sesji przez osobę atakującą[9].

W następnych podrozdziałach zostaną przedstawione rozwiązania zabezpieczające dostęp do Internetu w sieci lokalnej przed użytkownikami, którzy fałszują adres źródłowy podając się za innych. Każde z nich (z wyjątkiem rozwiązań sprzętowych) będzie analizowane pod kątem spełniania powyższych warunków.

[4] Do pobrania z `http://www.habets.pp.se/synscan/programs.php?prog=arping`.

[5] Wymaga to, aby lista zarejestrowanych adresów była dostępna dla skryptu.

[6] Narzędzie `arptables` służy do filtrowania ruchu ARP i jest dostępne w Linuksie.

[7] Jest to związane z mechanizmem sprawdzania konfliktów adresów IP w sieci przez hosta, który ma zamiar się do niej włączyć. Polega to na tym, że w chwili konfigurowania ustawień sieciowych przy starcie systemu operacyjnego, komputer wysyła w sieć zapytanie o własny adres IP. Jeśli otrzyma odpowiedź, to oznacza to, że w sieci funkcjonuje drugi host z takim samym adresem i występuje konflikt.

[8] Przechwycone dane niekoniecznie muszą być zrozumiałe dla atakującego, np. mogą być zaszyfrowane.

[9] W przeciwnym wypadku atakujący może przeprowadzić atak odmowy usług (ang. *Denial of Service*). Atak ten, choć nie związany bezpośrednio z problemem podszywania się, może obrócić bardzo bezpieczną metodę radzenia sobie z nim w kompletnie bezużyteczną.

5.2.3 Autoryzujące serwery proxy

Metoda polega na uruchomieniu jednego bądź kilku serwerów proxy[10]. Bezpośredni dostęp do sieci zewnętrznej jest blokowany, zaś do Internetu można dostać się jedynie za pośrednictwem serwera. Można wyróżnić zasadniczo trzy rodzaje proxy:

1. Proxy aplikacyjne. To rozwiązanie wymaga uruchomienia osobnego programu serwerowego dla każdego rodzaju usługi, której wykorzystanie ma być umożliwione. Choć proxy aplikacyjne są dostępne dla wielu protokołów[11], to istnieją aplikacje, dla których nie ma dedykowanych proxy. I choć wiele nowych programów pozwala na połączenia, wykorzystując najpopularniejszy protokół HTTP, rozwiązanie bazujące na proxy aplikacyjnych ma inną wadę. Jest nią brak uwierzytelniania w przypadku większości protokołów. Jedynym, który ją umożliwia jest HTTP, jednak polega ona na przesyłaniu nazwy użytkownika i hasła w postaci jawnej.

2. Protokół SOCKS 4. Zaletą w stosunku do poprzedniego typu proxy jest fakt, że obsługiwane są prawie wszystkie rodzaje połączeń bazujących na protokole TCP. Wadą jest brak uwierzytelniania.

3. SOCKS w wersji 5[12]. W porównaniu do wcześniejszej wersji, w tej znajduje się obsługa UDP oraz uwierzytelnianie. Protokół jest zaprojektowany elastycznie i pozwala na różne metody uwierzytelniania, których dwie wymienione są w specyfikacji protokołu. Pierwsza z nich oparta jest na przesyłaniu w postaci jawnej loginu i hasła[13], zaś druga wykorzystuje GSS-API[14]. Choć ostatnia metoda wygląda na bezpieczną (silne uwierzytelnianie, szyfrowanie transmisji), to autorowi nie udało znaleźć się aplikacji SOCKS, która by ją implementowała. Natomiast pierwsza, niebezpieczna metoda jest bardzo popularna.

Wszystkie rodzaje proxy wymagają ręcznej konfiguracji każdej aplikacji na hostach klienckich. Dla SOCKS istnieją programy[15], które ujednolicają ten proces, a także pozwalają na uruchamianie aplikacji, które nie mają obsługi tego protokołu. Jednak mimo tego wygoda korzystania z tego typu rozwiązań nie jest wysoka.

Bezpieczeństwo oferowane przez to rozwiązanie nie jest wysokie. Każde połączenie sieciowe jest osobną sesją (patrz poprzedni podrozdział), co powoduje, że użytkownik podczas normalnego korzystania z Internetu wykonuje ich bardzo wiele. Logowanie polega na wysłaniu nazwy użytkownika i hasła[16], zaś wylogowanie to zwykłe zakończenie połączenia. O ile taki sposób wylogowywania przy częstych sesjach nie stanowi dużego problemu[17], to logowanie stanowi poważne zagrożenie, ponieważ poufne informacje są wysyłane w postaci jawnej. Dodatkowo, ze względu na dużą liczbę sesji w czasie, nazwa użytkownika i hasło pojawiają się w sieci często, co stanowi ułatwienie dla osoby, która zechciałaby wejść w posiadanie tych danych. Sama transmisja jest w pewien sposób zabezpieczona, ponieważ do każdego połączenia dodawane są informacje uwierzytelniające, co uniemożliwia podszywanie się bez ich znajomości. Jednak te informacje są dla każdego połączenia takie same, identyczne z danymi używanymi do logowania[18] oraz łatwe w uzyskaniu.

5.2.4 Portale i Authpf

Popularnym rozwiązaniem, zwłaszcza w sieciach bezprzewodowych, są portale (ang. *captive portal*). Dostęp do Internetu w przypadku portalu działa w następujący sposób:

[10] Serwer proxy pośredniczy w komunikacji, wykonując połączenia sieciowe w imieniu użytkownika, który je zleca. Serwery proxy mogą działać na poziomie warstwy czwartej (serwery SOCKS) lub siódmej (serwery proxy konkretnych usług) w modelu OSI.

[11] Jak na przykład HTTP, FTP, POP3, IMAP, SMTP, NNTP, TELNET. Przedstawiony zbiór jest częścią listy protokołów obsługiwanych przez aplikację `Zorp`, która jest dostępna do pobrania na stronie `http://www.balabit.com`.

[12] Specyfikacja protokołu dostępna jest w RFC 1928.

[13] Autentykacja w SOCKS 5 jest specyfikowana przez RFC 1929.

[14] Jest to uogólniony interfejs (API), wykorzystywany przy pisaniu bezpiecznych aplikacji sieciowych. Bezpieczeństwo jest zapewniane przez techniki kryptograficzne udostępniane w systemie Kerberos [Tung, 1996]. Zastosowanie GSS-API w protokole SOCKS 5 jest określone RFC 1961.

[15] Na przykład `Sockscap`, dostępny do pobrania z `http://www.socks.permeo.com/`.

[16] Z wyjątkiem GSS-API w SOCKS 5, które jednak nie jest używane w praktyce.

[17] Przerywanie częstych i krótkich sesji nie jest łatwe, zaś efekt odmowy usługi praktycznie nie występuje, jak w przypadku długich sesji.

[18] Bo cała sesja jest zawarta w jednym połączeniu.

1. Użytkownik włącza komputer i otrzymuje adres IP przez DHCP. Od tej chwili posiada dostęp do sieci, jednak ruch internetowy jest blokowany.
2. Użytkownik uruchamia przeglądarkę internetową i wpisuje dowolny adres, co powoduje, że przeglądarka wysyła zapytanie HTTP.
3. Zapytanie jest przechwytywane przez router i przekierowywane na serwer portalu, co powoduje, że użytkownikowi wyświetla się strona z formularzem logowania.
4. Użytkownik loguje się korzystając z danych, które otrzymał od operatora sieci (login i hasło).
5. Serwer portalu weryfikuje poprawność logowania. Jeśli dane są prawidłowe oraz jeśli użytkownik nie zalega z opłatami za dostęp do Internetu, to oprogramowanie portalu kontaktuje się z firewallem w celu odblokowania adresu IP użytkownika.
6. Od tej chwili użytkownik może korzystać z Internetu. Wylogowanie następuje na życzenie użytkownika (z wykorzystaniem strony logowania), po upłynięciu ustalonego czasu nieaktywności lub w momencie, kiedy użytkownikowi skończył się dostęp do usługi. Wylogowanie polega na usunięciu z firewalla reguły, która zezwala na dostęp do Internetu.

Oprogramowanie portalowe wchodzi w skład produktów komercyjnych[19] oraz projektów Open Source[20]. Zaletą rozwiązań portalowych jest łatwość obsługi, ponieważ do uwierzytelnienia wystarcza przeglądarka www.

Podobną metodą uwierzytelniania użytkowników jest `Authpf`[21]. Różnica występuje w sposobie logowania. Zamiast korzystać z formularza www, użytkownik łączy się z serwerem operatora przez SSH i podaje nazwę użytkownika oraz hasło. Jeśli logowanie SSH przebiegnie pomyślnie adres użytkownika jest odblokowywany na firewallu tak jak w przypadku rozwiązania portalowego. Zakończenie połączenia SSH powoduje, że adres jest natychmiast blokowany z powrotem.

Bezpieczeństwo obu metod jest niskie. W przypadku rozwiązań portalowych największe ryzyko wiąże się z logowaniem do sesji. Większość rozwiązań nie stosuje bezpiecznego typu połączenia, tak więc dane użytkownika, takie jak login i hasło są przesyłane w postaci jawnej. Ochrona transmisji za pomocą mechanizmu SSL tylko częściowo rozwiązuje ten problem. Jest tak, ponieważ pierwsze połączenie, przekierowujące na stronę logowania odbywa się z wykorzystaniem protokołu HTTP[22], który nie jest bezpieczny. Atakujący może przechwycić to połączenie[23] i przekierować na serwer, który znajduje się pod jego kontrolą. Serwer ten może prezentować stronę logowania, która jest identyczna z oryginalną[24]. Sfabrykowana strona może nawet wykorzystywać bezpieczne połączenie HTTPS[25], aby odróżnienie jej od prawdziwej było jeszcze bardziej utrudnione. Jeśli użytkownik zaloguje się wykorzystując fałszywą stronę logowania, oszust stanie się posiadaczem jego danych uwierzytelniających.

Logowanie w metodzie `Authpf` jest mniej podatne na zagrożenia, ponieważ domyślnie wykorzystuje bezpieczne połączenie SSH. Istnieje atak na ten typ połączeń, który pozwala na przechwycenie transmisji[26]. Jednak jest on niezauważalny dla użytkownika tylko wtedy, gdy jest przeprowadzany w momencie logowania się po raz pierwszy. Próby ataków na kolejne połączenia skutkują wyraźnym ostrzeżeniem, które wyświetla klient SSH[27].

[19] Jak rozwiązanie Garderos WLAN C-OSS (`http://www.garderos.com/`) bądź Cisco CNS SESM i SSG (`http://www.cisco.com/`).

[20] Na przykład `NoCat`, dostępny pod adresem `http://nocat.net/`, oraz `WiCap-PHP` do pobrania z `http://wiki.personaltelco.net/index.cgi/WiCap_2dPH`.

[21] `Authpf` jest dostępny w systemie OpenBSD (`http://www.openbsd.org/`).

[22] Przekierowywanie na stronę logowania połączeń HTTPS jest niemożliwe, ponieważ protokół ten zabezpiecza transmisję przed tego typu działaniami.

[23] Z wykorzystaniem technik takich jak ARP spoofing lub DNS spoofing.

[24] Fabrykowanie przez oszustów stron internetowych, które imitują oryginalne serwisy np. banków, jest określane po angielsku jako *phishing* (wymaga interwencji użytkownika) lub *pharming* (wykorzystuje inne ataki jak DNS spoofing).

[25] Jeśli serwer portalowy posiada ważny certyfikat SSL jest to utrudnione, ale z wykorzystaniem pewnych technik można oszukać nieuważnego użytkownika. Na przykład, serwer oszusta może korzystać z innej domeny internetowej (o nazwie zbliżonej do nazwy domeny operatora sieci), dla której oszust posiada ważny certyfikat.

[26] Jest to tzw. *atak z wnętrza systemu* (ang. *Man in the Middle*), implementowany np. przez program `sshmitm` z pakietu `dsniff`. Pakiet dostępny jest pod adresem `http://www.monkey.org/~dugsong/dsniff/`.

[27] Jest to związane z tym, że przy pierwszym połączeniu klient SSH zapisuje klucz publiczny serwera. Jeśli przy następnym połączeniu będzie on inny niż zapisany, z wysokim prawdopodobieństwem wskazuje to na *atak z wnętrza systemu* (ang. *Man in the Middle*).

Przy założeniu, że logowanie przebiegło w sposób bezpieczny, należy przyjrzeć się bezpieczeństwu transmisji w obrębie sesji. Zarówno rozwiązanie portalowe jak i `Authpf` nie zapewniają uwierzytelniania przesyłanych przez użytkownika danych. Powoduje to, że w trakcie trwania sesji, jeśli użytkownik jest nieaktywny, atakujący zmieniając adresy IP i MAC może korzystać z dostępu do Internetu.

Również obie metody nie zabezpieczają przed atakiem odmowy usług, związanym z nieautoryzowanym wylogowaniem się. W przypadku `Authpf` wystarczy, że atakujący przerwie sesję SSH. W metodzie opartej na portalu wylogowywanie odbywa się przez wysłanie odpowiedniego zapytania HTTP z adresu klienta, które może zostać sfałszowane przez atakującego.

5.2.5 VPN

VPN (ang. *Virtual Private Network*) to technologia tunelowania transmisji, głównie na poziomie warstwy trzeciej[28]. Dane są enkapsulowane w połączenie, które zazwyczaj jest zabezpieczone z użyciem silnej kryptografii [Wobst, 2002]. Wykorzystanie technologii VPN w celu ochrony usługi dostępu do Internetu przed podszywaniem się polega na tym, że każdy użytkownik musi nawiązać połączenie VPN z routerem. Zezwolenie na dostęp do Internetu jest udzielane tylko tym użytkowników, którzy łączą się przez bezpieczny tunel. Dostępnych jest wiele rozwiązań VPN, spośród których poniżej przedstawione zostały najpopularniejsze:

IPsec (Internet Protocol Security). Jest to najpowszechniejszy standard[29] wirtualnych sieci prywatnych, należy do najbezpieczniejszych oraz do najbardziej skomplikowanych. Pozwala na transmisję danych w dwóch trybach: ESP (ang. *Encapsulating Security Payload*), który zapewnia poufność, integralność i autentykację, oraz AH (ang. *Authentication Header*) zapewniający autentykację oraz integralność, bez zachowania poufności. Tunel IPsec może wykorzystywać praktycznie dowolne algorytmy kryptograficzne. W zależności od trybu pracy, do transmisji danych używany jest jeden z dwóch protokołów warstwy trzeciej. Oprócz tego standard definiuje protokół wymiany klucza, który wykorzystuje połączenie UDP i jest opcjonalny.

OpenVPN. Pod nazwą tą kryje się protokół oraz jedyna jego implementacja, realizowana w ramach projektu Open Source. Zapewnia autentykację, poufność i integralność, wykorzystując do tego sprawdzone standardy SSL/TLS[30]. Używane mogą być wszystkie algorytmy kryptograficzne przez nie udostępniane. Transmisja danych oraz informacji kontrolnych odbywa się z wykorzystaniem protokołu UDP lub TCP oraz portu 1194.

PPTP (Point to Point Tunneling Protocol). Protokół został opracowany przez Microsoft i jest dostępny standardowo w każdej wersji systemu operacyjnego tej firmy począwszy od Windows 95. Do szyfrowania, które jest opcjonalne, wykorzystuje protokół MPPE[31], zaś autentykacje może realizować z użyciem kilku protokołów[32]. Protokół był analizowany pod względem bezpieczeństwa, co ujawniło wiele słabości, o których należy pamiętać przy wdrażaniu wirtualnych sieci prywatnych opartych na PPTP[33]. Dane użytkownika są transmitowane w postaci dwukrotnie enkapsulowanej: na początku w protokół PPP, a następnie w GRE. Połączenie kontrolne wykorzystuje port 1723/TCP.

PPPoE (PPP over Ethernet). Zdefiniowany w RFC 2516, protokół PPPoE umożliwia zestawianie łącz punkt-punkt w sieci LAN. Wykorzystuje do tego celu standard PPP, który jest używany do szyfrowania i uwierzytelniania. Te dwa zadania realizowane są przez te same mechanizmy, co w przypadku protokołu PPTP, dlatego bezpieczeństwo obu metod jest zbliżone. PPPoE to protokół warstwy trzeciej w modelu OSI, dlatego w standardzie zdefiniowany jest także mechanizm odnajdywania hostów w sieci[34].

[28] OpenVPN pozwala także na tunelowanie ramek ethernetowych, które należą do warstwy drugiej.

[29] IPsec jest zdefiowany w dokumentach RFC 2401-2412.

[30] Protokół SSL (ang. *Secure Sockets Layer*), oraz jego następca TLS (ang. *Transport Layer Security*) służą do zestawiania bezpiecznych połączeń w warstwie czwartej. TLS jest zdefiniowany w RFC 2246.

[31] MPPE (ang. *Microsoft Point to Point Encryption*) to metoda szyfrowania wykorzystująca algorytm RC4 z kluczem o długości 40, 56 lub 128 bitów.

[32] Są to PAP, CHAP, MS-CHAP i MS-CHAPv2.

[33] Hasła użytkowników muszą być długie, powinny być także regularnie zmieniane. Ponieważ jedno hasło wykorzystywane jest w uwierzytelnianiu i szyfrowaniu, jego poznanie pozwala odszyfrować wszystkie sesje, nawiązane z jego użyciem [Schneier i Mudge, 1998] [Schneier i Mudge, 1999].

[34] Jest to mechanizm analogiczny do ARP.

L2TP (Layer 2 Tunneling Protocol). Protokół został opracowany przez firmę Cisco i ma swoje korzenie w dwóch starszych technologiach: PPTP oraz L2F[35]. Przesyłanie danych polega na enkapsulowaniu ich w PPP, a następnie w IP. Choć protokół może używać szyfrowania i uwierzytelniania na poziomie PPP, najczęściej tą funkcję zapewnia IPsec[36], który może tunelować L2TP. L2TP jest zdefiniowany w RFC 2661. Transmisja może odbywać się z wykorzystaniem różnych protokołów warstw niższych jak: ATM, Frame Ralay oraz IP. W przypadku IP wykorzystywany jest port 1701/UDP.

Inne rozwiązania VPN: CIPE, vtun, tinc, które są dostępne na zasadach Open Source nie będą brane pod uwagę, ze względu na poważne błędy projektowe, które skutkują znacznym obniżeniem ich bezpieczeństwa [Gutmann, 2003].

Nawiązanie połączenia VPN odpowiada rozpoczęciu sesji przez użytkownika i przeprowadzeniu procesu logowania. Protokoły IPsec i OpenVPN posiadają mechanizmy uwierzytelniające, które wykorzystują silne algorytmy kryptograficzne. Protokoły PPTP i PPPoE nie wykorzystują tak mocnych metod, jednak w praktyce są trudne do złamania, jeśli wykorzystywane hasła są odpowiedniej długości.

Wszystkie protokoły zabezpieczają transmisje występujące w obrębie sesji użytkownika za pomocą szyfrowania. OpenVPN oraz PPTP wykorzystują długotrwałe połączenia TCP i UDP, co stwarza zagrożenie polegające na ich przerywaniu przez atakującego. Przerwanie połączenia powoduje wylogowanie użytkownika z sesji i konieczność ponownego logowania. IPsec przy wyłączonym mechanizmie uzgadniania klucza korzysta tylko z protokołów AH lub ESP, których nie da się zresetować, dlatego jest odporny na przerywanie sesji. PPPoE jest protokołem warstwy trzeciej, co także wyklucza resetowanie połączeń. Co więcej, zagrożenia związane z DHCP i ARP także jego nie dotyczą. Jedynym zagrożeniem, które autorowi udało się znaleźć w Internecie jest możliwość podszywania się pod serwer PPPoE na etapie połączenia[37].

Decyzja o wdrożeniu autoryzacji użytkowników za pomocą technologii VPN musi uwzględniać dostępność klientów poszczególnych protokołów w używanych systemach operacyjnych. System GNU/Linux, a także wolne systemy z rodziny BSD obsługują wszystkie rodzaje wirtualnych sieci prywatnych, które zostały wymienione w tym podrozdziale. W przypadku systemów Microsoftu, w chwili pisana tego opracowania, sytuacja wygląda inaczej dla każdego protokołu. Istnieje wiele implementacji IPsec, jednak większość z nich jest płatna[38]. Systemy Windows 2000/XP mają wbudowaną obsługę L2TP tunelowanego przez IPsec, Microsoft udostępnia bez opłat to oprogramowanie w wersji dla Windows 9x/ME. OpenVPN działa w Windows 2000/XP, jednak nie obsługuje starszych systemów z Redmond. Protokół PPTP jest wbudowany we wszystkie wersje Windows, poczynając od wersji 95. PPPoE jest wbudowany w Windows 2000/XP, jednak oferowana obsługa szyfrowania jest niestabilna[39], co wymusza korzystanie z zewnętrznych klientów. Dostępne są `RASPPPOE`[40] oraz `WinPoET`[41]. Mimo, że `RASPPPOE` można pobrać bezpłatnie z Internetu, to nie można go wykorzystywać do zastosowań biznesowych, a takim najczęściej jest świadczenie dostępu do Internetu. `WinPoET` jest oprogramowaniem w pełni komercyjnym.

Istnieje wiele implementacji serwerowych wszystkich wymienionych technologii VPN, zarówno komercyjnych[42] jak i dystrybuowanych na zasadach Open Source. Są one na tyle popularne, że bez trudu można je znaleźć przy pomocy wyszukiwarki internetowej.

5.2.6 Metody pomocnicze

Ten podrozdział skupia się na technikach programowych, które pozwalają zabezpieczyć część prezentowanych metod przed większością ataków typu odmowa usług. Jedynym atakiem, którego nie udaremniają jest

[35] L2F (ang. *Layer 2 Forwarding*) jest niepopularnym i przestarzałym protokołem VPN, opracowanym przez Cisco i używanym głównie w urządzeniach tej firmy.

[36] Tunelowanie L2TP w IPsec jest opisane w RFC 3193.

[37] Problem ten jest opisany pod adresem `http://ask.slashdot.org/askslashdot/02/12/01/2333218.shtml?tid=172`

[38] Autor nie znalazł bezpłatnego klienta IPsec dla Windows 9x/ME. Natomiast dla Windows 2000/XP dostępna jest darmowa implementacja firmy SecurePoint – `Personal Firewall & VPN Client`. Aby korzystać z IPsec pod starszymi wersjami Windows niezbędny jest zakup komercyjnego rozwiązania, np. `SoftRemote` firmy SafeNet.

[39] Wbudowane w Windows oprogramowanie klienckie korzysta z szyfrowania w trybie *Statefull*, zaś nie obsługuje trybu *Stateless*. Ten pierwszy typ połączenia wykazuje dużą niestabilność przy nawet znikomych stratach pakietów, które zdarzają się w sieciach LAN.

[40] `RASPPPOE` dostępny jest na stronie `http://www.raspppoe.com/`

[41] Strona producenta programu `WinPoET` to `http://www.finepointnetworks.com/`

[42] Z wyjątkiem OpenVPN, który powstał jako projekt Open Source.

ten omówiony w podrozdziale 1.3.4, który wymaga zabezpieczeń sprzętowych bądź systemu wykrywania i alarmowania. Prezentowane poniżej techniki zwiększają także bezpieczeństwo słabych metod prewencyjnych, jak portale, `Authpf` oraz proxy.

Firewall

Odpowiednio skonfigurowany firewall pozwala zabezpieczyć tablice routingu hosta przed nieautoryzowaną modyfikacją (atak ICMP Redirect omówiony w podrozdziale 1.3.6), oraz sesję użytkownika przed złośliwym wylogowaniem spowodowanym przerwaniem połączenia (podrozdział 1.3.7).

Zabezpieczenie przed atakiem ICMP Redirect polega na filtrowaniu tego typu komunikatów ICMP.

Ochrona przez resetowaniem połączeń także jest prosta, wymaga jednak złamania specyfikacji protokołów TCP i UDP. Polega na ignorowaniu prób zakończenia połączeń, które przenoszą sesję użytkownika zarówno po stronie serwera (VPN, SSH) jak i komputera klienckiego. W praktyce można to zrealizować z wykorzystaniem oprogramowania firewall po obu stronach, które będzie blokować pakiety TCP RST, TCP FIN oraz ICMP Port Unreachable, dotyczące połączeń VPN i SSH przez okres trwania sesji. Spowoduje to, że zakończenie tych połączeń w warstwie czwartej modelu OSI nie będzie możliwe, nawet na życzenie użytkownika[43].

DHCP

DHCP jest protokołem, który nawet w połączeniu z silnymi metodami uwierzytelniania, może zostać wykorzystany do przeprowadzenia ataków odmowy usług (podrozdział 1.3.5). Dlatego, jeśli operator sieci może ponieść koszt utraty kontroli nad konfiguracją sieciową hostów powinien zrezygnować z jego wykorzystania, w celu ochrony sieci przed tymi atakami.

ARP

Jeśli sieć LAN bazuje na protokole ARP, to jest ona narażona na wiele ataków, które zostały opisane w podrozdziale 1.3.3. Można się przed nimi zabezpieczyć przez częściową rezygnację z tego protokołu. W tym celu, do tablicy ARP routera, który zapewnia dostęp do Internetu, należy wprowadzić statyczne wpisy dla wszystkich zarejestrowanych hostów. Niezbędna jest także modyfikacja tablicy ARP każdego komputera, tak aby zawierała ona statyczny wpis dla adresu routera.

Istnieje rozszerzenie standardu ARP – ARPSec, które wprowadza pewne zabezpieczenia [Etienne, 2000]. Jednak na dzień dzisiejszy popularne systemy operacyjne nie obsługują tego protokołu.

Sprawdzanie zgodności adresów IP i MAC

Jeśli weryfikacja zgodności adresów IP i MAC na routerze nie jest stosowana jako główna metoda uwierzytelniania, lecz wspólnie z inną metodą, może wydatnie zwiększyć jej bezpieczeństwo.

5.2.7 Inteligentne przełączniki sieciowe

Jeśli sieć dysponuje zarządzalnymi przełącznikami o odpowiednim zasobie funkcji to zabezpieczenie się przed zmianą adresów jest możliwe na dwa sposoby.

Port security. Jest to funkcja, która pozwala na przypisanie adresu MAC do portu w sposób statyczny[44]. Oznacza to, że jeśli na porcie pojawią się ramki o źródłowych adresach MAC innych niż zdefiniowane w przełączniku, to zostaną odrzucone.

Standard IEEE 802.1x. Przełącznik, który implementuje ten standard, we współpracy z serwerem RADIUS[45] potrafi autoryzować hosta i kontrolować jego dostęp do sieci na poziomie portu, do którego

[43] Jest możliwe takie skonfigurowanie systemu, aby po udanym wylogowaniu w warstwie siódmej następowało usunięcie reguł firewalla, które blokują próby zamykania połączenia w warstwie czwartej. Wymagałoby to napisania skryptu, który wykonywałby się przy wylogowywaniu po obu stronach połączenia.

[44] Inną nazwą jest kontrola dostępu do portu, ang. *Port Access Control*.

[45] Technologia RADIUS jest używana do centralizacji procesów autentykacji, autoryzacji i rozliczania dostępu do usług.

jest on podłączony. Host przed rozpoczęciem korzystania z sieci musi się uwierzytelnić. Jeśli proces ten przebiegnie pomyślnie, przełącznik odblokowuje port hosta, zaś w przeciwnym razie port pozostaje zablokowany, co całkowicie uniemożliwia dostęp do sieci. Większość przełączników po odblokowaniu portu automatycznie filtruje wszystkie ramki, których źródłowy adres MAC jest inny niż autoryzowanego hosta.

Ze względu na fakt, że powyższe metody są implementowane na poziomie przełączników sieciowych, są one wyjątkowo skuteczne. Ramki ze sfałszowanym adresem źródłowym są filtrowane natychmiast na przełączniku, co uniemożliwia ich dotarcie nie tylko do routera, ale nawet do innych hostów w sieci. To powoduje, że podszywanie się jest całkowicie niemożliwe i wszelkie związane z nim ataki (nawet ten opisany w podrozdziale 1.3.4) są udaremniane.

Jednak skuteczność przewyższająca wszystkie metody prewencyjne opisane w tym rozdziale jest okupiona wieloma wadami, związanymi z wdrożeniem i administracją. Obie metody wymagają rozlokowania inteligentnych przełączników we wszystkich węzłach sieci. Są one i zawsze będą (w szczególności sprzęt obsługujący standard 802.1x) droższe od typowych urządzeń niezarządzalnych. Pomijając jednorazowy koszt wdrożenia, należy brać pod uwagę koszty związane z wymianą uszkodzonych jednostek oraz kradzieże[46] i zabezpieczanie się przed nimi.

W przypadku metody Port Security, dochodzi do tego konieczność zarządzania konfiguracją przełączników, ponieważ każda fizyczna zmiana w sieci musi uwzględniać ich uaktualnienie. Jeśli wystąpią pomyłki lub istnieje potrzeba zdiagnozowania jakiegoś problemu sieciowego, filtrowanie w warstwie drugiej może skutecznie utrudniać pracę administratorom.

W przypadku metody 802.1x jej wdrożenie wymaga uruchomienia serwera RADIUS oraz konfigurację systemów klienckich. Istnieje wiele implementacji RADIUS, w tym dostępne na zasadach Open Source. Windows XP posiada wbudowanego klienta standardu 802.1x, zaś Microsoft udostępnił także klienta dla systemu Windows 2000. Starsze systemy wymagają zainstalowania oddzielnego, płatnego oprogramowania[47]. Dla systemów uniksowych dostępny jest program `Xsupplicant`[48], dystrybuowany na zasadach Open Source.

[46] Naturalne jest, że drogi sprzęt jest bardziej pożądany przez złodziei.
[47] Na przykład program `Odyssey Client`, który jest oferowany przez firmę Funk Software `http://www.funk.com/`.
[48] Można go pobrać z `http://www.open1x.org/`

“I never think of the future – it comes soon enough.”

– Albert Einstein

Rozdział 6

Kierunki dalszego rozwoju

Już podczas pisania aplikacji *multispoof*, zaś w szczególności na etapie testów i mierzenia wydajności, pojawiały się pomysły na ulepszenia. Dotyczą one zwiększania odporności na wykrywanie, przełamywania innych metod uwierzytelniania oraz podnoszenia wydajności i komfortu korzystania z programu. Proponowane usprawnienia mogą zostać wprowadzone w przypadku kontynuowania rozwoju aplikacji *multispoof*.

Jednocześnie zostaną przedstawione zarysy nowych technik detekcji, skoncentrowanych na wykrywaniu bardziej zakamuflowanych ataków. Należy podkreślić, że silne techniki prewencyjne, opisane w rozdziale poprzednim, udaremniają ataki wykorzystujące proponowane udoskonalenia. Z tego powodu w tym rozdziale nie zostaną zaprezentowane metody zapobiegawcze.

6.1 Utrudnianie wykrywania

W tej klasie znajdują sie takie propozycje udoskonaleń, które koncentrują się na redukcji nietypowych zachowań sieciowych programu, czyniąc go trudniejszym do wykrycia. Należy mieć na uwadze, że nawet najsilniejsze środki techniczne nie są w stanie ukryć działania aplikacji *multispoof*, jeśli jej użytkownik działa w sposób zbyt zwracający uwagę[1].

6.1.1 Randomizacja

Propozycja zakłada wprowadzenie losowości w każdym punkcie zachowania aplikacji, który wykazuje jakiekolwiek regularności. W szczególności dotyczy to transmisji sieciowych, generowanych przez program, ponieważ są one szczególnie narażone na przechwycenie i analizę statystyczną[2]. Dlatego randomizacji powinny ulec:

Odstępy czasowe. Dotyczy to okresów czasu, który upływa pomiędzy cyklicznymi zdarzeniami, występującymi w trakcie działania aplikacji. Są to: skanowanie ARP (domyślnie co 60 sekund) oraz testowanie łączności hosta z Internetem (standardowo godzina). Zamiast stałych odstępów czasowych występujących pomiędzy krótkimi okresami zmasowanego skanowania lub testowania, należałoby wprowadzić losowej długości przerwy występujące po każdym teście bądź wysłaniu zapytania ARP. Hosty, których aktywność została stwierdzona przez komponent **`deta`**, nie powinny podlegać skanowaniu ARP, co wprowadzi dodatkową niepewność do analizy statystycznej.

Kolejność. Kolejne kroki wymienionych powyżej procesów powinny odbywać się w losowej kolejności. W każdym obiegu kolejność powinna być randomizowana od nowa. To samo dotyczy algorytmu równoważącego obciążanie – adresy używane do podszywania się powinny być ułożone w przypadkowej kolejności.

[1] Na przykład wysycając 100% łącza internetowego operatora.

[2] Przykładami prostej analizy statystycznej są metody detekcji omówione w podrozdziałach 5.1.1 i 5.1.2.

Metody testowania. W tej chwili jest zaimplementowany zaledwie jeden rodzaj testowania łączności z Internetem. Idealnie powinno być ich wiele, tak aby symulowały normalną aktywność sieciową, zaś adresy używane do testów powinny być wybierane losowo[3].

6.1.2 Symulacja normalnej aktywności

Drugą metodą, która ma na celu utrudnianie detekcji aplikacji *multispoof* jest zwiększanie podobieństwa między jej zachowaniami sieciowymi, a standardową aktywnością hostów fizycznych. Ideałem byłoby, gdyby każdy adres wykorzystywany w procesie podszywania się, był nie od odróżnienia zdalnie od prawdziwego hosta fizycznego. Wtedy metody wykrywania opisane w podrozdziałach 5.1.3 i 5.1.4 byłyby bezużyteczne.

Jedną z technik symulacji hosta fizycznego jest odpowiadanie na pakiety dotyczące standardowych usług, oraz komunikaty ICMP. W tym celu każdy host fizyczny musiałby zostać zbadany w celu zdalnego rozpoznania systemu operacyjnego, udostępnionych usług oraz innych zachowań sieciowych[4], co prowadziłoby do stworzenia bazy profili komputerów pracujących w sieci. Profile te następnie byłyby ładowane do bazy programu honeyd[5]. Głównym zastosowaniem tej aplikacji jest tworzenie komputerów-pułapek, które imitują różne, słabo zabezpieczone systemy operacyjne. Maszyna z uruchomionym programem honeyd jest następnie podłączana do Internetu i ściśle monitorowana w celu obserwacji metod używanych przez włamywaczy [Piotrowski, 2005].

Wykorzystanie tego programu w aplikacji *multispoof* odwraca role: to administrator sieci ma ulec wrażeniu, że ma do czynienia z prawdziwymi komputerami, a nie z oszustem, który tworzy tę iluzję. Każdy wirtualny host imitowałby host fizyczny o odpowiadającym mu adresie.

Kolejną techniką symulacji zachowania hosta fizycznego jest generowanie standardowych transmisji sieciowych, które pojawiają się przy zwyczajnym używaniu komputera. Mogą to być na przykład zapytania DHCP wysyłane na etapie uruchamiania systemu operacyjnego, zapytania ARP o adres routera oraz zapytania DNS. Te transmisje mogłyby, jak w poprzedniej technice, być badane dla każdego hosta fizycznego, a następnie generowane podczas wykorzystywania jego adresu w procesie podszywania się[6].

6.1.3 Kontrola wykorzystania adresów

Metoda detekcji opisana w podrozdziale 5.1.7 jest bardzo skuteczna i nie da się przed nią całkowicie zabezpieczyć. Jednak proponowana zmiana zachowania aplikacji do pewnego stopnia pozwoli zmniejszyć nietypowe zachowania sieci, co spowoduje, że administrator może dłużej nie zauważać działań oszusta.

Pomysł polega na tym, aby aplikacja sama decydowała o zaprzestaniu podszywania się pod hosty, które mogą zostać włączone w bliskiej przyszłości. Wymagałoby to ciągłej analizy zachowania poszczególnych hostów i, na podstawie zgromadzonych danych, przewidywania kiedy dany host zostanie uruchomiony.

6.1.4 *Antyradar*

Jeśli pomimo zabezpieczeń przedstawionych w tym rozdziale, administrator zorientuje się, że w zarządzanej przez niego sieci działa aplikacja *multispoof*, to jego następnym krokiem będzie ustalenie na jakim komputerze została uruchomiona. Jeżeli sieć jest oparta na niezarządzalnych przełącznikach, to próby lokalizacji będą polegały na wyłączaniu kolejnych urządzeń, co zostało opisane dokładnie w podrozdziale 5.1.

Z punktu widzenia intruza, przydałby się mechanizm, który będzie ostrzegał przed próbami lokalizacji. Technicznie nie jest to skomplikowane i polega na ciągłym testowaniu, czy sieć nie została przerwana. Naiwnym sposobem mogłoby być sprawdzanie, czy router jest osiągalny, na przykład za pomocą komunikatów ICMP Echo Request lub ARP. Przebiegły administrator może jednak podczas procesu lokalizacji wysyłać na nie odpowiedzi, aby zmylić oszusta. Dlatego testowanie powinno odbywać się z wykorzystaniem metod, które uniemożliwiają podszywanie się[7], jak np. połączenia szyfrowane. Dodatkową zaletą będzie mniejsza

[3] Ewentualnie z puli najczęściej używanych adresów w Internecie.

[4] Jak na przykład nieudzielanie odpowiedzi na komunikaty ICMP Echo Request ze względu na zainstalowane oprogramowanie firewall.

[5] Strona domowa projektu to `http://www.honeyd.org/`

[6] Niektóre transmisje, jak zapytania DHCP, musiałyby być wysyłane jeszcze przed rozpoczęciem podszywania się, aby nie mieszały się z innym ruchem, co mogłoby wzbudzić podejrzenia administratora i być pomocą w wykrywaniu.

[7] Tym razem to administrator, aby wprowadzić w błąd oszusta, podszywa się pod testowany przez niego host.

podatność na wykrycie samego procesu testowania, jeśli do tego celu zostanie użyte zwyczajne połączenie HTTPS, na przykład z bankiem internetowym.

6.2 Przełamywanie innych metod uwierzytelniania

Program *multispoof* można przystosować do pracy w sieci opartej na innym mechanizmie uwierzytelniania[8]. Wymaga to jednak stosunkowo rozległych modyfikacji, w zależności od wybranej metody dostępu do usługi.

6.2.1 Metody warstwy trzeciej

Przełamywanie metody `Authpf` zostanie omówione jako pierwsze. Ponieważ korzystanie z Internetu jest możliwe tylko podczas trwania sesji SSH użytkownika, to zakładając, że oszust nie posiada danych pozwalających na samodzielne nawiązanie sesji, konieczne byłoby podszywanie się pod adres danego hosta w trakcie jego aktywności. Wiąże się to z następującymi zmianami w aplikacji:

- Należałoby opracować inną technikę wykrywania aktywności hostów, która opierałaby się na stwierdzeniu, które hosty mają nawiązane połączenie SSH z routerem. Mogłoby to polegać na podsłuchiwaniu ruchu włączonych komputerów z wykorzystaniem techniki ARP Spoofing.
- Konieczna byłaby zmiana zachowania programu w przypadku wykrycia aktywności hosta. Zamiast natychmiastowego przerwania procesu podszywania się, musiałby się on właśnie wtedy rozpoczynać.
- Należałoby dbać o to, aby połączenie SSH nie zostało zakończone. W tym celu pakiety wchodzące w jego skład otrzymane od routera musiałyby być przesyłane do klienta, zaś te otrzymane od klienta – do routera. Jest to istotne, ponieważ protokół SSH w `Authpf` jest konfigurowany w ten sposób, aby informacje kontrolne[9] były często wymieniane między stronami połączenia. Jeśli zbyt wiele takich informacji nie dotrze do adresata, połączenie jest przerywane[10].
- Powyższe odnosi się także do innych połączeń klienta. Dzięki technice ARP Spoofingu legalny użytkownik może korzystać z dostępu do Internetu wspólnie z oszustem.

Bardzo podobnie można przełamywać zabezpieczenie dostępu do Internetu w modelu portalowym. Metoda ta zazwyczaj nie wykorzystuje specjalnego połączenia, które musi być nawiązane przez cały czas trwania sesji, dlatego z reguły podszywanie się będzie prostsze niż w przypadku `Authpf`. Dodatkowo oszust, wykorzystując ARP Spoofing, może uniemożliwić użytkownikowi wylogowanie, dzięki czemu zapewni sobie nieprzerwane korzystanie z jego sesji.

6.2.2 Wykorzystywanie danych uwierzytelniających

Inną metodą jest przechwytywanie danych niezbędnych do rozpoczęcia sesji. Choć w przypadku metod wykorzystujących portal lub proxy wystarczy aktywny podsłuch, to aby przechwycić dane innych metod, wymagane są bardziej inwazyjne metody. Aby przejąć nazwę użytkownika i hasło połączeń SSH wykorzystywanych przez `Authpf`, konieczne jest przeprowadzenie *ataku z wnętrza systemu* (ang. *Man in the Middle*), zaś poznanie danych używanych w metodach opartych na wirtualnych sieciach prywatnych wymaga bezpośredniego ataku na host, który te dane przetrzymuje w swojej pamięci.

Kiedy oszust dysponuje już danymi uwierzytelniającymi, może używać ich w celu dostępu do Internetu jak zwykły użytkownik, z tą różnicą, że nie ponosi kosztów finansowych tej usługi. Jednak aby taki dostęp nie powodował konfliktów w sieci oraz aby możliwe było uzyskiwanie powiększonego pasma transmisyjnego, korzystne mogłoby być wykorzystanie w tym celu aplikacji *multispoof*. Wymagałoby to jednak wbudowania w program mechanizmów autoryzacji. W przypadku metody portalowej jest to stosunkowo proste, ponieważ wymaga tylko, aby przed rozpoczęciem podszywania się została wykonana procedura logowania dla danego

[8] Alternatywne techniki uwierzytelniania zostały omówione w podrozdziale 5.2.

[9] Są to pakiety wysyłane w celu sprawdzenia, czy druga strona jest osiągalna. Pakiety te stanowią bezpieczną analogię komunikatów ICMP Echo Request.

[10] Częste sprawdzanie stanu połączenia ma stanowić zabezpieczenie przed podszywaniem się.

hosta. Podobnie byłoby z wykorzystaniem metody `Authpf`, z tym że po uwierzytelnieniu połączenie musiałoby być utrzymywana dla każdego hosta, zaś w przypadku metody VPN, wszelkie transmisje należałoby enkapsulować w odpowiedni tunel.

Natomiast metoda wykorzystująca SOCKS 5 wymaga, aby każde połączenie używało tego protokołu, co wiązałoby się z poważnymi modyfikacjami aplikacji. Należałoby zrezygnować z koncepcji opartej na interfejsie wirtualnym *tap* i zastąpić ją lokalnym serwerem SOCKS. Użytkownik łączyłby się z nim, natomiast serwer przekazywałby te połączenia do serwera proxy operatora używając autentykacji.

6.2.3 Metody dodatkowo zabezpieczone

Jeśli dostęp do Internetu jest chroniony z wykorzystaniem jednej z metod prewencyjnych, zaś dodatkowo wprowadzone są zabezpieczenia warstwy drugiej, opisane w podrozdziale 5.2.6, to podszywanie się jest utrudnione. Dotyczy to korzystania z adresów hostów, które są w danej chwili aktywne. W tym podrozdziale zostanie pokazane, że stosowanie nawet obu tych technik na raz w celu zabezpieczenia słabej metody uwierzytelniania nie uniemożliwia kradzieży usługi dostępu do Internetu. Wymaga to jednak poważnych modyfikacji w aplikacji *multispoof*.

Współdzielenie dostępu do Internetu z nieświadomym tego użytkownikiem, jest możliwe w metodach portalowych i `Authpf`. W obu przypadkach należy wyszukiwać hosty, które są uwierzytelnione, jednak nie korzystają intensywnie z sieci zewnętrznej.

Portale

Technika korzystania z istniejącej już sesji użytkownika polega na wprowadzeniu w błąd przełączników sieciowych. Ponieważ host użytkownika nie generuje prawie żadnego ruchu, to ramki z jego adresem źródłowym, ale płynące z komputera oszusta, spowodują, że ramki powrotne będą przesyłane także do niego.

Ponieważ host użytkownika co pewien czas może generować drobne transmisje, będzie to powodować zmianę dynamicznej konfiguracji przełączników i tymczasowe dostarczanie ramek przeznaczonych dla oszusta do legalnego komputera. Może to powodować resetowanie połączeń, jednak jeśli ruch generowany przez oszusta będzie miał wystarczająco wyższe natężenie[11], to przełączniki ulegną ponownemu przeprogramowaniu i dalsze ramki będą docierały już do jego hosta.

Authpf

W przypadku `Authpf` korzystanie z nawiązanej już sesji użytkownika opiera się na tej samej zasadzie co powyżej. Jednak ze względu na to, że połączenie SSH nie może zostać zerwane wymaga to dodatkowego mechanizmu. Połączenie będzie nawiązane tak długo, dopóki obie jego strony, tzn. router i klient SSH są w stanie się komunikować. Zbyt długie przerwy skutkują zerwaniem sesji.

Maksymalną długość przerwy można ustalić, zaś pakiety kontrolne SSH są wysyłane co pewien, również możliwy do zmierzenia, czas. Pakiety wysyłane przez klienta do routera są niedostępne dla atakującego, natomiast komunikaty kontrolne SSH wysyłane przez router, mimo, że skierowane do użytkownika trafiają do oszusta. Po odebraniu maksymalnej liczby pakietów kontrolnych[12] host oszusta musiałby wymuszać na komputerze użytkownika wysłanie ramki sieciowej. Do tego celu może być wykorzystane np. zapytanie ARP. Ważne jest także, aby po jego wysłaniu przerwać wszystkie transmisje, które wykorzystują źródłowy adres MAC hosta użytkownika. Komunikat ARP Reply, wygenerowany przez host użytkownika, przeprogramuje przełączniki tak, że z powrotem zaczną przesyłać dane skierowane pod adres użytkownika do jego hosta. Natychmiast po tym, oszust musiałby wysłać ostatni otrzymany komunikat SSH do użytkownika, fałszując adres źródłowy, tak aby ramka wyglądała jakby przyszła z routera. Klient SSH otrzyma tę ramkę i wygeneruje na nią odpowiedź, co uratuje sesję przed zakończeniem. Od tej chwili możliwe jest dalsze korzystanie z Internetu przez oszusta, zaś opisana operacja musi być powtarzana co ustalony czas.

[11] Oszust może generować ciągły, lokalny ruch profilaktycznie. Aby było to skuteczne, oraz zbytnio nie obciążało sieci, ramki powinny być małe, ale pojawiać się często.

[12] Tzn. maksymalnej liczby komunikatów SSH, które mogą pozostać bez odpowiedzi, jednocześnie nie powodując zerwania sesji.

6.3 Wydajność, wygoda, kontrola

Ostatnią klasą usprawnień są te dotyczące zwiększenia możliwej przepustowości, podnoszące komfort korzystania z aplikacji, oraz oferujące metody kontroli i bieżącego monitorowania jej pracy.

6.3.1 Tryby pracy

W obecnej wersji program zoptymalizowany jest pod kątem pobierania z Internetu danych w sposób masowy, tak szybko jak to możliwe. Jednak inne rodzaje aktywności sieciowej nie są tak dobrze obsługiwane. Aplikacja *multispoof* mogłaby oferować różne tryby pracy, w zależności od zadań, które użytkownik zamierza wykonywać.

Interaktywne korzystanie z globalnej sieci najlepiej funkcjonowałoby, gdyby w procesie podszywania się były wykorzystywane adresy tylko tych abonentów, którzy wykupili usługę najwyższej jakości. Wymagałoby to od aplikacji, aby była w stanie zmierzyć takie parametry sieciowe jak przepustowość i opóźnienia, które zapewniane są poszczególnym użytkownikom przez operatora.

W niektórych sytuacjach przydatny byłby tryb, który maksymalizuje anonimowość oraz zmniejsza prawdopodobieństwo wykrycia kosztem jakości dostępu. Mogłoby polegać to na podszywaniu się tylko pod jeden adres na raz. Aplikacja wprawdzie nie oferowałaby powiększonej przepustowości, ale korzystanie z cudzego adresu utrudniałoby śledzenie użytkownika programu. Dodatkowo wykorzystanie tylko jednego adresu nie budziłoby podejrzeń administratora sieci.

Ostatnim trybem byłoby maksymalne przyśpieszenie transmisji, nawet kosztem ułatwienia wykrycia. Użytkownik aplikacji aktywowałby ten tryb w przypadku zaistnienia nagłej potrzeby wykonania dużej transmisji tak szybko jak to możliwe. Aby zrealizować swoje zadanie, aplikacja *multispoof* mogłaby podszywać się pod wszystkie komputery, nawet te, które w danej chwili są aktywne.

6.3.2 Statystyki

Możliwość generowania różnego rodzaju statystyk z działania programu byłaby użyteczną funkcją. Użytkownik aplikacji mógłby analizować zależność od czasu takich informacji jak przepustowość łączną i dla każdego wykorzystywanego adresu, liczbę pakietów na sekundę, liczbę adresów, które biorą udział w procesie podszywania się itd. Pozwalałoby to na skuteczną ocenę wyników działania aplikacji.

6.3.3 Inne poprawki

Obecna wersja aplikacji *multispoof* nie może wykorzystywać więcej niż 100 adresów w procesie podszywania się. Poprzez współpracę z autorem modułu `nth` (opisanego w podrozdziale 3.5.2), należałoby go zmodyfikować tak, aby przełamać to ograniczenie. W tym celu niezbędna jest modyfikacja części rezydującej w jądrze, ponieważ używa ona liczników ośmiobitowych, które należałoby zastąpić szesnastobitowymi. Pozwoliłoby to na zwiększenie maksymalnej liczby adresów do ponad 65 tysięcy[13].

Kolejne poprawki to zlokalizowanie i usunięcie wycieku pamięci w komponencie **`netdb`**, oraz wprowadzenie zabezpieczenia przed niekorzystnym wpływem klienta DHCP na działanie aplikacji. To ostatnie polegałoby na dodaniu do skryptu *multispoof* reguł podsystemu `Netfilter`, które blokują ruch DHCP na interfejsie zewnętrznym.

Należałoby także wykonać dokładny *profiling* aplikacji, co pozwoliłoby zlokalizować te elementy, które w największym stopniu wpływają na wydajność.

6.4 Detekcja przełamywania innych metod uwierzytelniania

Zaproponowane w podrozdziale 6.2 metody przełamywania innych technik uwierzytelniających wykazują pewne cechy charakterystyczne, które mogą zostać wykorzystane do ich wykrycia.

[13] Choć protokół IP pozwala zaadresować ponad 4 miliony komputerów, to rzadko zdarza się, aby w jednym segmencie sieci było powyżej tysiąca stacji. 65 tysięcy to liczba wynikająca z pojemności szesnastobitowego licznika ($2^{16} = 65536$).

6.4.1 Wykorzystywanie usługi wspólnie z użytkownikiem

Techniki detekcji opisane w tym podrozdziale mają zastosowanie w przypadku wykorzystywania uwierzytelniania metodą portalową oraz `Authpf`. Pozwalają wykryć pasożytnicze korzystanie z usługi dostępu do Internetu, z wykorzystaniem podszywania się pod aktywne hosty.

Wydaje się, że efektywne może być zastosowanie popularnego i stosunkowo bogatego zestawu metod wykrywania nielegalnego podziału łącza z wykorzystaniem translacji adresów. Nielegalny podział łącza polega na udostępnianiu usługi dostępu do Internetu innym użytkownikom. Efekty uboczne tego procederu są bardzo zbliżone do metody wykorzystywania adresu aktywnego hosta przez oszusta. Techniki te są dokładnie opisane w [Tomaszewski *et al.*, 2005], [Zalewski, 2001], [Zalewski, 2002], [Kohno *et al.*, 2005], zaś w Internecie można znaleźć gotowe narzędzia je implementujące[14].

Jeśli w sieci wdrożono metodę portalową lub `Authpf` bez dodatkowych zabezpieczeń w warstwie drugiej, to jednoczesne korzystanie z usługi może odbywać się z wykorzystaniem ataku ARP Spoofing. Atak ten wykazuje bardzo charakterystyczne objawy, które polegają m.in. na tym, że w sieci można obserwować większą liczbę adresów MAC niż IP. Istnieją narzędzia, które wykorzystują te słabości w celu wykrywania ataku[15].

Jeżeli oszust nie chce stosować ataku ARP Spoofing, obawiając się wykrycia, bądź nie ma takiej możliwości, ze względu na zaimplementowane w sieci dodatkowe zabezpieczenia w warstwie drugiej, może wykorzystywać metodę opartą na wprowadzaniu w błąd przełączników sieciowych (podrozdział 6.2.3). Jeśli używane w sieci przełączniki pozwalają na zdalny odczyt listy adresów MAC wraz z powiązanymi z nimi portami, to detekcja tego nadużycia jest stosunkowo prosta. Polega na periodycznym odpytywaniu przełącznika i sprawdzaniu, czy adresy MAC nie migrują między portami. Jeśli sytuacja taka ma miejsce często, to jest to niemal pewna oznaka, że w sieci jest aktywny oszust.

Powyższa technika jest jednak bezużyteczna w sieciach opartych na prostych, niezarządzalnych przełącznikach. W tym przypadku metoda detekcji jest bardziej skomplikowana i opiera się na badaniu strat transmisji. Straty te wynikają z faktu, że w momentach reasocjacji par MAC-Port na przełącznikach, ramki mogą płynąć nie do tego hosta, który na nie oczekuje. Powoduje to, że nawiązane połączenia są resetowane. Takie zachowanie nie jest typowe, dlatego jeśli występuje często, można przypuszczać, że w sieci działa oszust.

6.4.2 Przechwytywanie danych umożliwiających dostęp

Zdobywanie danych uwierzytelniających

Część z zaprezentowanych metod ochrony dostępu do Internetu pozwala na bardzo łatwe przejęcie danych używanych do uwierzytelniania. Np. metody portalowe i oparte na proxy wymagają od oszusta jedynie aktywnego podsłuchiwania z wykorzystaniem jednej z metod omówionych w podrozdziale 1.3. Istnieje wiele technik detekcji tego rodzaju nadużycia, np. zaprezentowane w poprzednim podrozdziale wykrywanie ataku ARP Spoofing.

Przechwycenie danych uwierzytelniających w metodzie `Authpf` nie jest już tak proste, wymaga bowiem ataku na połączenie SSH. Detekcja tego typu nadużycia jest wbudowana w klienta tego protokołu, jednak wymaga od administratora współpracy z użytkownikami.

Próby zdobywania danych uwierzytelniających w metodach opartych na wirtualnych sieciach prywatnych są najczęściej możliwe tylko po uprzednim przejęciu kontroli nad komputerem użytkownika. Wymaga to od atakującego dokonywania włamań elektronicznych. Wykrywanie tego procederu może polegać na monitorowaniu sieci z wykorzystaniem systemu detekcji intruzów (ang. *Intrusion Detection System*).

Wykorzystywanie danych uwierzytelniających

Jeśli oszust zdołał już przechwycić dane uwierzytelniające, będzie próbował ich użyć w celu uzyskania dostępu do usługi. Problem detekcji tego typu nadużycia polega na odróżnianiu połączeń legalnych użytkowników od tych, nawiązywanych przez oszusta. Nie jest to łatwe, ale możliwe, jeśli osoba ta postępuje nieostrożnie. W tym celu należy wykorzystywać informacje podprogowe, jak np. adres MAC (o którego zmianie oszust mógł zapomnieć).

[14] Np. `p0f`, dostępny pod adresem `http://lcamtuf.coredump.cx/p0f.shtml`.
[15] Np. narzędzie `arpwatch NG`, dostępne do pobrania z `http://freequaos.host.sk/arpwatch/`.

Jeżeli atakujący próbuje zwiększyć swoje pasmo transmisyjne podszywając się pod wielu użytkowników, to zastosowanie mają metody opisane w podrozdziale 5.1.

Dodatek A

Wybrane kody źródłowe

A.1 Skrypt *multispoof*

```
#!/bin/sh

function usage
{
  echo "Usage:"
  echo "␣␣multispoof␣[-fvh]␣[-a␣<age>]␣[-e␣<age>]␣[␣-s␣<int>]␣[-t␣<name>]␣-i␣<iface>"
  echo ""
  echo "␣␣-i␣iface␣␣Uses␣given␣network␣interface"
  echo "␣␣-t␣name␣␣␣Assigns␣given␣name␣to␣tap␣device;␣default:␣$TAP_IFACE"
  echo "␣␣-f␣␣␣␣␣␣␣Starts␣with␣flushed␣cache;␣default:␣don't␣flush"
  echo "␣␣-a␣␣␣␣␣␣␣minimal␣age;␣default␣$MIN_AGE"
  echo "␣␣-e␣␣␣␣␣␣␣minimal␣test␣age;␣default␣$MIN_TEST_AGE"
  echo "␣␣-s␣␣␣␣␣␣␣arp␣scan␣interval;␣default␣$SCAN_INTERVAL"
  echo "␣␣-d␣␣␣␣␣␣␣Dummy␣mode,␣exits␣after␣initial␣setup"
  echo "␣␣-v␣␣␣␣␣␣␣Verbose␣mode"
  echo "␣␣-h␣␣␣␣␣␣␣Shows␣this␣help"
  echo "␣␣-V␣␣␣␣␣␣␣Displays␣the␣current␣version"
  echo ""
  echo "All␣time␣related␣options␣have␣to␣be␣specified␣in␣seconds."
  exit
}

# Sets up netfilter rules
function setup_nf_rules
{
  MULTISPOOF_SID=‘ps -o sid= $$‘

  iptables -t nat -N $MAIN_CHAIN 2> /dev/null
  iptables -t nat -N $SUB_CHAIN 2> /dev/null
  iptables -t nat -N $TEST_CHAIN 2> /dev/null
  iptables -t nat -F $MAIN_CHAIN
  iptables -t nat -F $SUB_CHAIN

  iptables -t nat -A $MAIN_CHAIN -o $TAP_IFACE -m owner \
    --sid-owner $MULTISPOOF_SID -j $TEST_CHAIN
  iptables -t nat -A $MAIN_CHAIN -j $SUB_CHAIN
```

```
  iptables -t nat -A $MAIN_CHAIN -j DROP
  iptables -t nat -D POSTROUTING -o $TAP_IFACE -j $MAIN_CHAIN \
     2>/dev/null
  iptables -t nat -I POSTROUTING -o $TAP_IFACE -j $MAIN_CHAIN
}

# Cleans up netcfilter rules
function clean_up_nf_rules
{
  iptables -t nat -D POSTROUTING -o $TAP_IFACE -j $MAIN_CHAIN \
        2> /dev/null
  iptables -t nat -F $MAIN_CHAIN
  iptables -t nat -F $SUB_CHAIN
  iptables -t nat -F $TEST_CHAIN
  iptables -t nat -X $MAIN_CHAIN
  iptables -t nat -X $SUB_CHAIN
  iptables -t nat -X $TEST_CHAIN
}

# Restores IP address and default gateway on network interface
function restore_iface_config
{
  ip addr add $SCAN_IP/$NETMASK dev $REAL_IFACE
  ip route add default via $GATEWAY_IP
}

# Signal handler - shutdowns app gracefully
function clean_up
{
  trap 15
  echo "multispoof:␣cleaning␣up"
  # Take down tap interface, so associated routes get cleaned
  ip link set $TAP_IFACE down
  # After tap interface is down, we can restore network config
  restore_iface_config
  clean_up_nf_rules
  # Remove netdb temporary directory
  rm -rf $NDB_DIR
  # Kill all processes
  kill -SIGTERM -$$
}

function spoof_pipeline
{
  rx $REAL_IFACE "ip␣and␣not␣ether␣broadcast" tapio | \
  cmac unspoof $MIN_AGE $NDB_SOCKET | tapio $TAP_IFACE | \
  cmac spoof $MIN_AGE $NDB_SOCKET | tx $REAL_IFACE tapio
}

function scanarp_pipeline
{
  scanarp $NDB_SOCKET $SCAN_INTERVAL $SCAN_IP $SCAN_MAC | \
  tx $REAL_IFACE scanarp
}
```

```
function deta_pipeline
{
  rx $REAL_IFACE "ether␣broadcast␣or␣arp" deta | deta $NDB_SOCKET \
  $MIN_AGE | tx $REAL_IFACE deta
}

# Registers default mac in netdb
function register_mac
{
  local IFACE=$1
  local MAC=‘ip link|grep -A 1 $IFACE | tail -n 1 | cut -d ’ ’ -f 6‘
  if [ ! -z "$MAC" ]
  then
    ndbexec $NDB_SOCKET setvar defmac $MAC
  else
    echo "Problem␣getting␣mac␣address␣of␣interface␣$IFACE."
    kill $$
  fi
}

# Sets banned list in netdb
function set_variables
{
  local BANNED="$GATEWAY_IP:0.0.0.0:255.255.255.255"
  ndbexec $NDB_SOCKET setvar banned "$BANNED"
}

# Moves configuration from real interface to tap interface
function setup_ifaces
{
  local REAL=$1
  local TAP=$2
  local CIDR="$SCAN_IP/$NETMASK"
  ip addr flush dev $REAL 2> /dev/null
  ip addr add $CIDR dev $TAP
  ip link set $TAP up
  arp -i $TAP -s $GATEWAY_IP $GATEWAY_MAC
  ip route add default via $GATEWAY_IP
}

# Returns if given socket exist
function wait_for_socket
{
  local SOCKET=$1
  while [ ! -S "$SOCKET" ]
  do
    sleep 0.1
  done
}

# Returns if given network interface exist
function wait_for_iface
{
```

```
  local IFACE=$1
  while [ ‘ip link show dev $IFACE > /dev/null 2> /dev/null \
           || echo false‘ ]
  do
    sleep 0.1
  done
}

# Static configuration created by make
COMPONENTS_DIR=/usr/local/lib/multispoof
CACHE_FILE=/var/local/cache/multispoof/netdb.cache
VERSION=0.6.1

FLUSH_CACHE=""
VERBOSE_MODE=""
DUMMY_MODE=""
REAL_IFACE=""
TAP_IFACE="tap0"
# Defaults: all intervals and ages in seconds
# How long the host needs to be quiet to be considered as inactive.
MIN_AGE=300
# How often individual host should be tested.
MIN_TEST_AGE=3600
# How often natman should poll netdb
NATMAN_INTERVAL=5
# How often conncheck should poll netdb
CONNCHECK_INTERVAL=6
# How often scanarp should send arp requests
SCAN_INTERVAL=60

# Parse commandline parameters
while getopts Vdfvht:i:a:e:s: NAME
do
  case $NAME in
  f) FLUSH_CACHE="-f"
     ;;
  d) DUMMY_MODE="-d"
     ;;
  v) VERBOSE_MODE="-v"
     ;;
  i) REAL_IFACE="$OPTARG"
     ;;
  t) TAP_IFACE="$OPTARG"
     ;;
  a) MIN_AGE="$OPTARG"
     ;;
  e) MIN_TEST_AGE="$OPTARG"
     ;;
  s) SCAN_INTERVAL="$OPTARG"
     ;;
  V) echo "multispoof␣version␣$VERSION"
     exit
     ;;
  *) usage
```

```
    ;;
  esac
done

if [ -z "$REAL_IFACE" ]
then
  echo "multispoof:␣You␣need␣to␣specify␣interface␣to␣use."
  exit 1
fi

if [ ! -z "$VERBOSE_MODE" ]
then
  echo "multispoof:␣PID␣$$"
  echo "multispoof:␣Discovering␣network␣setup."
fi

GATEWAY_IP=‘ip route | grep default | grep $REAL_IFACE | head -n 1 \
           | cut -d "␣" -f 3‘
if [ -z "$GATEWAY_IP" ]
then
  echo "multispoof:␣Default␣gateway␣bound␣to␣specified␣interface␣required."
  exit 1
fi

_IP_ADDR=‘ip addr show dev $REAL_IFACE | egrep "inet\␣" | head -n 1 \
         | awk ’{ print $2 }’‘
SCAN_IP=‘echo $_IP_ADDR | cut -d "/" -f 1‘
if [ -z "$SCAN_IP" ]
then
  echo "multispoof:␣Couldn’t␣get␣interface␣IP␣address."
  exit 1
fi

SCAN_MAC=‘ip link show dev $REAL_IFACE | grep "link/ether" \
         | awk ’{ print $2 }’‘
if [ -z "$SCAN_MAC" ]
then
  echo "multispoof:␣Couldn’t␣get␣interface␣MAC␣address."
  exit 1
fi

NETMASK=‘echo $_IP_ADDR | cut -d "/" -f 2‘
if [ -z "$NETMASK" ]
then
  echo "multispoof:␣Couldn’t␣get␣interface␣netmask."
  exit 1
fi

GATEWAY_MAC=‘/usr/sbin/arp -na $GATEWAY_IP | grep $REAL_IFACE \
            | tail -n 1 | cut -d "␣" -f 4 | grep ":"‘
if [ -z "$GATEWAY_MAC" ]
then
  ping -c 1 -w 2 $GATEWAY_IP 2>&1 > /dev/null
  GATEWAY_MAC=‘/usr/sbin/arp -na $GATEWAY_IP | grep $REAL_IFACE \
```

```
            | tail -n 1 | cut -d "␣" -f 4 | grep ":"‘
fi
if [ -z "$GATEWAY_MAC" ]
then
  echo "multispoof:␣Couldn’t␣obtain␣gateway\’s␣MAC␣address."
  exit 1
fi

if [ ! -z "$VERBOSE_MODE" ]
then
  echo "multispoof:␣Real␣interface:␣$REAL_IFACE,␣tap␣interface:␣$TAP_IFACE"
  echo "multispoof:␣Gateway␣ip:␣’$GATEWAY_IP’␣mac:␣’$GATEWAY_MAC’"
  echo "multispoof:␣Scan␣ip:␣’$SCAN_IP/$NETMASK’␣mac:␣’$SCAN_MAC’"
  echo "multispoof:␣Min␣age:␣$MIN_AGE,␣min␣test␣age:␣$MIN_TEST_AGE"
  echo "multispoof:␣Intervals␣-␣natman:␣$NATMAN_INTERVAL,␣conncheck:␣\
␣␣$CONNCHECK_INTERVAL,␣scanarp:␣$SCAN_INTERVAL"
fi

if [ ! -z "$DUMMY_MODE" ]
then
  exit
fi

if [ ‘whoami‘ != "root" ]
then
  echo "multispoof:␣Root␣privileges␣required."
  exit 1
fi

NDB_DIR=‘mktemp -td multispoof.XXXXXXXX‘ || FAIL=1
if [ ! -z "$FAIL" ]
then
  echo "multispoof:␣Couldn’t␣create␣temporary␣directory."
  exit 1
fi
NDB_SOCKET="$NDB_DIR/socket"
MAIN_CHAIN="multispoof-main"
SUB_CHAIN="multispoof-sub"
TEST_CHAIN="multispoof-test"
TEST_SCRIPT="${COMPONENTS_DIR}/access-test"
PATH=$COMPONENTS_DIR:$PATH

trap clean_up 15 2

netdb $NDB_SOCKET $CACHE_FILE $FLUSH_CACHE &
wait_for_socket $NDB_SOCKET
set_variables
spoof_pipeline &
scanarp_pipeline &
deta_pipeline &
setup_nf_rules
natman $SUB_CHAIN $MIN_AGE $NATMAN_INTERVAL $NDB_SOCKET &
wait_for_iface $TAP_IFACE
register_mac $TAP_IFACE
```

```
setup_ifaces $REAL_IFACE $TAP_IFACE
conncheck $NDB_SOCKET $TEST_CHAIN $CONNCHECK_INTERVAL \
  $MIN_AGE $MIN_TEST_AGE $TEST_SCRIPT &

# Wait for all children processes.
wait
```

A.2 Skrypt generujący ruch sieciowy

```
#!/bin/sh

TEST_HOST=$1

function launch_netcat
{
  nc $TEST_HOST 19 > /dev/null &
}

function initial_launch
{
  i=200
  while [ $i -gt 0 ]
  do
    launch_netcat
    sleep 0.1
    i=$[ $i - 1 ]
  done
}

function clean_up
{
  trap 15
  trap 2
  /bin/kill -SIGTERM -$$
}

trap clean_up 15 2

initial_launch

while [ true ]
do
  CANDIDATE=‘ps -C nc -o pid=,etime=|tr -d ":"|sort -k 2|head -n 1|awk ’{print $1}’‘
  echo Killing $CANDIDATE
  kill $CANDIDATE
  launch_netcat
  sleep 2
done
```

Dodatek B

Instalacja i obsługa aplikacji *multispoof*

B.1 Wymagania

Wymagania programu *multispoof* zostały przedstawione na poniższej liście:

- Interfejs *tun/tap* w jądrze systemu.
- Obsługa lokalnych gniazd uniksowych.
- Narzędzie `ip` z pakietu `iproute2`.
- Obsługa SNAT oraz filtrów `owner` i `nth` w podsystemie `Netfilter` jądra Linuksa oraz w narzędziu `iptables`. Obsługa filtra `nth` w momencie pisania pracy nie jest obecna w standardowym jądrze, dlatego niezbędne jest wykorzystanie pakietu `patch-o-matic` [Marie, 2002] w celu dodania tej funkcjonalności do jądra oraz programu `iptables`. Pakiet `patch-o-matic` jest dostępny do pobrania z witryny projektu `Netfilter`: `http://www.netfilter.org/`.
- Powłoka systemowa zgodna z `sh` z obsługą `getopts`. Jedyną testowaną powłoką był `Bash` 3.0, który jest dostępny pod adresem `http://www.gnu.org/software/bash/bash.html`.
- Biblioteka `libpcap` w wersji, która zostanie wydana po 3.05.2005. W chwili pisania tego dokumentu nie jest dostępne oficjalne wydanie, dlatego nie została podana dokładna wersja. Można użyć wersji *snapshot* z systemu kontroli wersji CVS. Biblioteka jest dostępna na stronach projektu `tcpdump`: `http://www.tcpdump.org/`.
- Biblioteka `libnet` 1.1.2.1. Prawdopodobnie starsze wersje także będą działać. Bibliotekę można pobrać z `http://www.packetfactory.net/libnet/`.
- Biblioteka `glib` w wersji 2.6 lub wyższej. Można ją pobrać z `http://www.gtk.org/`.
- Standardowe narzędzia uniksowe (`sed`, `grep` itp.).

Dodatkowo, aby przeprowadzić kompilację należy posiadać w systemie:

- Kompilator C. Testowano proces kompilacji jedynie z wykorzystaniem `gcc` w wersji 3.3.5. Pakiet `gcc` jest dostępny pod adresem `http://gcc.gnu.org/`.
- Pliki nagłówkowe bibliotek `glib`, `libnet` i `libpcap`.
- Pliki nagłówkowe jądra (katalog `/usr/include/linux`) muszą opisywać API jądra wykorzystywanego do uruchamiania aplikacji *multispoof*. Praktyka pokazuje, że nawet drobne różnice w synchronizacji nagłówków z jądrem powodują trudne w diagnozie problemy – w szczególności chodzi o nagłówki obsługi interfejsów *tun/tap*.

- Narzędzie `make`. Testowane było jedynie `GNU Make` w wersji 3.80. Program dostępny jest na `http://www.gnu.org/software/make/make.html`.
- Narzędzie `pkg-config`. Testowana była wersja 0.15.0. Program dostępny jest na `http://pkgconfig.freedesktop.org/wiki/`.
- Narzędzie `install`, dostępne w pakiecie `coreutils`. Testowana była wersja 5.2.1. Pakiet `coreutils` dostępny jest na `http://www.gnu.org/software/coreutils/`.

Na powyższych listach nie zostały uwzględnione wymagania poszczególnych bibliotek i narzędzi, jak np. obsługa gniazd typu *PACKET* w Linuksie, od której zależy działanie biblioteki `libpcap`.

B.2 Kompilacja i instalacja

Aby przeprowadzić kompilację i instalację aplikacji *multispoof*, należy przejść do katalogu ze spakowanymi źródłami programu (znajdują się one w pliku `multispoof-0.6.1.tar.gz`, który jest dołączony do pracy), a następnie wydać komendę:

```
$ tar zxf multispoof-0.6.1.tar.gz
$ cd multispoof-0.6.1
```

Spowoduje to, że źródła zostaną rozpakowane do osobnego katalogu. Po tej czynności można zmodyfikować parametry instalacyjne. Domyślnie aplikacja instalowana jest w katalogach `/usr/local/` i `/var/local/` jednak można je zmienić, modyfikując plik `Makefile`. Istotne jest, aby zmiany te wprowadzać *przed* wydaniem polecenia `make`, ponieważ oprócz instalacji, wpływają one także na proces budowania plików wykonywalnych[1]. Następnym krokiem jest kompilacja, którą przeprowadza się wydając komendę:

```
$ make
```

Jeśli kompilacja przebiegnie bez błędów to ostatnim etapem jest instalacja aplikacji. W tym celu należy wykonać poniższą komendę (niezbędne są uprawnienia administratora systemu):

```
# make install
```

B.3 Obsługa

Po pomyślnej instalacji program *multispoof* powinien znajdować się w ścieżce dostępu. Aplikacja do poprawnego działania wymaga uprawnień administratora, ponieważ korzysta z bezpośredniego dostępu do sieci z pominięciem stosu TCP/IP (podsłuchiwanie i wstrzykiwanie danych do sieci – podrozdział 3.3.1), tworzy wirtualny interfejs *tap* (podrozdział 3.3.2), przeładowuje reguły `iptables` (podrozdziały 3.5.1 i 3.5.2) oraz zmienia adresy IP a także manipuluje tablicami ARP i routingu (podrozdział 3.6). Przed uruchomieniem należy sprawdzić, czy w systemie działa klient DHCP i jeśli znajduje się on w pamięci konieczne jest jego zakończenie. Aplikacja *multispoof* na czas działania zdejmuje adres IP z interfejsu fizycznego i przydziela go do interfejsu wirtualnego *tap*. Klient DHCP mógłby przydzielić adres z powrotem do interfejsu fizycznego, co doprowadziłoby do sytuacji, w której dwa interfejsy dzieliłyby jeden adres IP.

Uruchomienie aplikacji *multispoof* z parametrem `-h` spowoduje wyświetlenie skróconego opisu obsługi programu wraz z listą opcji. Lista ta została przedstawiona w tabeli B.1.

Jedynym wymaganym parametrem jest `-i`. Aby uruchomić aplikację w trybie diagnostycznym korzystając z interfejsu eth1 i używając parametrów domyślnych należy wydać komendę:

[1] Na etapie kompilacji kod skryptu *multispoof* jest modyfikowany tak, aby zawierał właściwe ścieżki do katalogu komponentów oraz do pliku bazy programu **`netdb`**.

Tablica B.1: Opcje akceptowane przez aplikację *multispoof*.

Opcja	Opis
-i	Program będzie używać podanego interfejsu sieciowego.
-t	Interfejs wirtualny tap otrzyma podaną nazwę. Domyślnie tap0.
-f	Program nie wczyta zawartości pliku cache z listą hostów. Domyślnie lista jest wczytywana.
-a	Minimalny czas nieaktywności hosta, podawany w sekundach. Domyślnie 300.
-e	Minimalny czas pomiędzy testami łączności dla hosta, podawany w sekundach. Domyślnie 3600.
-s	Odstęp czasowy pomiędzy skanowaniami ARP. Domyślnie 60 sekund.
-d	Tryb testowy, program nie uruchamia komponentów. Przydatne przy -v.
-v	Tryb diagnostyczny, program wyświetla dodatkowe informacje o swoim działaniu.
-h	Wyświetla skrócony opis obsługi wraz z listą opcji.
-V	Wyświetla wersję programu.

```
# multispoof -vi eth1
```

Spowoduje to, że na standardowe wyjście błędów wyświetlone zostaną informacje wygenerowane przez poszczególne komponenty i skrypt zarządzający:

```
multispoof: PID 6858
multispoof: Discovering network setup.
multispoof: Real interface: eth1, tap interface: tap0
multispoof: Gateway ip: '192.168.1.201' mac: '00:20:AF:C4:6F:E3'
multispoof: Scan ip: '192.168.1.33/24' mac: '52:54:00:12:34:57'
multispoof: Min age: 300, min test age: 3600
multispoof: Intervals - natman: 5, conncheck: 6, scanarp: 60
netdb: Listening on /tmp/multispoof.XXtIpgfL/socket
rx (tapio): listening on eth1
rx (deta): listening on eth1
tx (scanarp): using device eth1
tx (deta): using device eth1
tapio: virtual interface: tap0
tx (tapio): using device eth1
cmac (unspoof): Using 7e:e5:3e:48:3b:5f as default mac
```

Z powyższych informacji można odczytać PID (dotyczy on skryptu zarządzającego, poszczególne komponenty są uruchamiane jako procesy potomne mają więc inne identyfikatory), wykrytą konfigurację sieci oraz gniazdo lokalne komponentu **`netdb`**, używane do komunikacji pomiędzy komponentami aplikacji. Po uruchomieniu można odnotować zmiany w konfiguracji sieci. Poniżej została przedstawiona konfiguracja przed:

```
$ ip addr show dev eth1
2: eth1: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 52:54:00:12:34:56 brd ff:ff:ff:ff:ff:ff
    inet 172.20.0.2/16 brd 172.20.255.255 scope global eth0
$ ip route
192.168.1.0/24 dev eth1  proto kernel  scope link  src 192.168.1.33
default via 192.168.1.201 dev eth1
$ arp -na
? (192.168.1.201) at 00:20:AF:C4:6F:E3 [ether] on eth1
```

Natomiast uruchomienie aplikacji spowodowało, że interfejs eth1 został pozbawiony adresu IP, który został przypisany do interfejsu *tap0*. Dodatkowo, domyślna trasa prowadzi przez interfejs *tap*, zaś w tablicy ARP pojawił się statyczny wpis dla adresu domyślnego routera:

```
$ ip addr show dev eth1
3: eth1: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 52:54:00:12:34:57 brd ff:ff:ff:ff:ff:ff
$ ip addr show dev tap0
5: tap0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 500
    link/ether 7e:e5:3e:48:3b:5f brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.33/24 scope global tap0
$ ip route
192.168.1.0/24 dev tap0  proto kernel  scope link  src 192.168.1.33
default via 192.168.1.201 dev tap0
$ arp -na
? (192.168.1.201) at 00:20:AF:C4:6F:E3 [ether] PERM on tap0
```

Jeśli w sieci w której działa aplikacja *multispoof* są aktywne jakieś hosty, to stopniowo powinny być one wykrywane i dodawane do wewnętrznej tablicy, która jest utrzymywana przez komponent **`netdb`**. Wykrycie nowego hosta jest sygnalizowane przez wypisanie komunikatu:

```
deta: Adding host 192.168.1.42 (00:00:86:3d:bb:fd)
```

Aby wyświetlić zawartość tablicy wykrytych hostów trzeba nawiązać połączenie z komponentem **`netdb`** za pośrednictwem lokalnego gniazda uniksowego, którego lokalizacja została podana przez aplikację przy starcie. Można to zrobić korzystając z programu `socat`[2]:

```
# socat UNIX-CONNECT:/tmp/multispoof.XXtIpgfL/socket -
+OK Welcome
dump
192.168.1.23
192.168.1.42
192.168.1.69
192.168.1.47
+OK Dump complete
quit
+OK Bye
```

Wykryte hosty przed zakończeniem programu zostaną zapisane do pliku[3], tak aby przy następnym uruchomieniu nie było potrzeby budowania listy od początku. Jeśli program działa wystarczająco długo aby stwierdzić nieaktywność przynajmniej jednego hosta i jeśli choć jeden z nieaktywnych komputerów posiada dostęp do Internetu, to aplikacja przeładowuje reguły NAT, co można zobaczyć korzystając z komendy `iptables`:

```
# iptables -t nat -nL multispoof-sub
Chain multispoof-sub (1 references)
target     prot opt source               destination
SNAT       all  --  0.0.0.0/0            0.0.0.0/0 \
every 2th packet #0 to:192.168.1.23
SNAT       all  --  0.0.0.0/0            0.0.0.0/0 \
every 2th packet #1 to:192.168.1.42
```

W chwili wykonywania powyższej komendy w procesie podszywania się są wykorzystywane adresy `192.168.1.23` i `192.168.1.42`. Aby sprawdzić prawidłowość działania aplikacji, tzn. czy adresy IP i MAC faktycznie są zmieniane, można uruchomić równolegle narzędzie `ping` oraz analizator pakietów. Program nasłuchujący na interfejsie fizycznym[4] zarejestruje ruch podobny do poniższego:

[2] Program jest dostępny pod adresem: `http://www.dest-unreach.org/socat/`
[3] Domyślną lokalizacją dla tego pliku jest `/var/local/cache/multispoof/netdb.cache`.
[4] W testach wykorzystywany był program `tcpdump`.

```
# tcpdump -eni eth1 icmp
tcpdump: WARNING: eth1: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 96 bytes
23:21:27.997833 00:20:ed:b6:78:43 > 00:20:af:c4:6f:e3, ethertype IPv4 (0x0800), \
length 98: IP 192.168.1.23 > 212.77.100.101: icmp 64: echo request seq 1
23:21:28.039748 00:20:af:c4:6f:e3 > 00:20:ed:b6:78:43, ethertype IPv4 (0x0800), \
length 98: IP 212.77.100.101 > 192.168.1.23: icmp 64: echo reply seq 1
23:21:28.992051 00:02:44:7b:09:11 > 00:20:af:c4:6f:e3, ethertype IPv4 (0x0800), \
length 98: IP 192.168.1.42 > 212.77.100.101: icmp 64: echo request seq 2
23:21:29.036361 00:20:af:c4:6f:e3 > 00:02:44:7b:09:11, ethertype IPv4 (0x0800), \
length 98: IP 212.77.100.101 > 192.168.1.42: icmp 64: echo reply seq 2
23:21:29.992307 00:20:ed:b6:78:43 > 00:20:af:c4:6f:e3, ethertype IPv4 (0x0800), \
length 98: IP 192.168.1.23 > 212.77.100.101: icmp 64: echo request seq 3
23:21:30.046712 00:20:af:c4:6f:e3 > 00:20:ed:b6:78:43, ethertype IPv4 (0x0800), \
length 98: IP 212.77.100.101 > 192.168.1.23: icmp 64: echo reply seq 3
23:21:30.994180 00:02:44:7b:09:11 > 00:20:af:c4:6f:e3, ethertype IPv4 (0x0800), \
length 98: IP 192.168.1.42 > 212.77.100.101: icmp 64: echo request seq 4
23:21:31.098238 00:20:af:c4:6f:e3 > 00:02:44:7b:09:11, ethertype IPv4 (0x0800), \
length 98: IP 212.77.100.101 > 192.168.1.42: icmp 64: echo reply seq 4
```

Każdy pakiet reprezentowany jest przed dwie linijki, spośród których pierwsza zawiera adresy MAC, a druga adresy IP. Oba typy adresów są pogrubione dla czytelności. Wyjście analizatora wyraźnie pokazuje, że kolejne pakiety (widać to po numerach sekwencyjnych) generowane przez program `ping` mają zmieniane adresy źródłowe. Ponieważ podczas testu były wykorzystywane tylko dwa adresy, pakiety nieparzyste mają adres `192.168.1.23`, zaś nieparzyste – `192.168.1.42`. Adresy MAC także ulegają zmianie, tak aby pary IP i MAC zgadzały się.

W przykładzie transmisji programu `ping`, zmianie ulega każdy kolejny pakiet. Jednak w przypadku protokołów połączeniowych, takich jak TCP adresy będą zmieniane dla połączeń, tzn. wszystkie pakiety w obrębie jednego połączenia będą miały taki sam adres. Należy podkreślić, że aplikacja *multispoof* nie zajmuje się śledzeniem połączeń, za tę czynność odpowiedzialne jest jądro systemu. Aplikacja ustala jedynie reguły podsystemu `Netfilter`. Więcej na ten temat można znaleźć w podrozdziale 3.5.2.

Skład i środowisko programistyczne

Niniejsza praca została w całości sporządzona za pomocą oprogramowania Open Source. Do składu posłużył doskonały, oparty na XML'u system **t**book[5], do wprowadzania tekstu używany był edytor `Vim`[6], w sprawdzaniu poprawności pisowni pomogły narzędzia `GNU Aspell`[7] oraz `vimspell`[8]; diagramy powstały w programie `Dia`[9]. Wykresy przepustowości zostały stworzone w programie `Gnuplot`[10] z wykorzystaniem danych zmierzonych programem `tcpstat`[11], zaś wykresy aktywności przełącznika sieciowego powstały przy użyciu aplikacji `RRDtool`[12] zasilanej informacjami uzyskiwanymi przez narzędzia z pakietu `Net-SNMP`[13]. Wygenerowanie wersji drukowalnej zostało przeprowadzone z wykorzystaniem systemu LaTeX w dystrybucji teTeX[14], zaś w automatyzacji tego procesu bardzo pomógł program `GNU Make`[15]. Wszystkie elementy pracy były wprowadzane do znakomitego systemu kontroli wersji `Darcs`[16], zaś całość działała pod kontrolą systemu `Ubuntu Linux`[17].

Aplikacja, która powstała w ramach pracy była rozwijana z wykorzystaniem kompilatora `GCC`[18], debuggera `GDB`[19], edytora `Vim`, systemu kontroli wersji `Darcs`, narzędzia `GNU Make` oraz innych, pomniejszych narzędzi uniksowych. Cały proces odbywał się pod kontrolą systemu `Debian/GNU Linux`[20] uruchomionego w emulatorze `QEmu`[21].

[5] `http://tbookdtd.sourceforge.net/` [Bronger, 2004]
[6] `http://www.vim.org/`
[7] `http://aspell.sourceforge.net/`
[8] `http://www.vim.org/scripts/script.php?script_id=465`
[9] `http://www.gnome.org/projects/dia/`
[10] `http://www.gnuplot.info/`
[11] `http://www.frenchfries.net/paul/tcpstat/`
[12] `http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/`
[13] `http://net-snmp.sourceforge.net/`
[14] `http://www.tug.org/teTeX/`
[15] `http://www.gnu.org/software/make/`
[16] `http://abridgegame.org/darcs/`
[17] `http://www.ubuntulinux.org/`
[18] `http://gcc.gnu.org/`
[19] `http://www.gnu.org/software/gdb/gdb.html`
[20] `http://www.debian.org/`
[21] `http://fabrice.bellard.free.fr/qemu/`

Bibliografia

AL-HERBISH, Thamer, 1999: *Raw IP Networking FAQ*.
URL `http://www.whitefang.com/rin/`

BAUTTS, Tony, Terry DAWSON i Gregor PURDY, luty 2005: *Linux Network Administrator's Guide, Third Edition*. O'Reilly.

BELLOVIN, Stephen, maj 1989: *Security Problems in the TCP/IP Protocol Suite*. Computer Communication Review.

BRECHT, Tim i Michal OSTROWSKI, list. 2001: *Exploring the performance of select-based Internet servers*. Rap. tech., HP Labs.

BRONGER, Torsten, 2004: *tbook: a system for XML authoring, version 1.5.2*.

CARSTENS, Tim, 2002: *Programming with pcap*.
URL `http://www.tcpdump.org/pcap.htm`

ETIENNE, Jerome, 2000: *ARPsec: An ARP security extension*. Ottawa Linux Symposium 2000.

GUSTA, Marek i Maciej SZMIT, 2003: *Sniffing w ethernecie z przełącznikami*. Hakin9, (2).

GUTKOWSKI, Marek, 2004: *Kilka ciekawych metod rozpoznawania systemu operacyjnego*. Hakin9, (5).

GUTMANN, Peter, 22 wrz. 2003: *Linux's answer to MS-PPTP*. Wiadomość z grupy dyskusyjnej.
URL `http://diswww.mit.edu/bloom-picayune/crypto/14238`

Hon02, 4 marz. 2002: *Know Your Enemy: Passive Fingerprinting. Identifying remote hosts, without them knowing*.
URL `http://www.honeynet.org/papers/finger/`

HUNT, Craig, 1997: *TCP/IP - Administracja sieci*. Read Me.

KLYAGIN, Konstantin, 2004: *Skanowanie portów z punktu widzenia administratora*. Hakin9, (8).

KOHNO, Tadayoshi, Andre BROIDO i KC CLAFFY, maj 2005: *Remote physical device fingerprinting*. IEEE Symposium on Security and Privacy.
URL `http://www.caida.org/outreach/papers/2005/fingerprinting/`

KRASNYANSKY, Maxim i Florian THIEL, 2002: *Universal TUN/TAP device driver*. Linux Kernel Documentation.

MARIE, Fabrice, 22 sier. 2002: *Rozszerzenia Netfilter HOWTO*.

MCCANNE, Steven i Van JACOBSON, 19 grudz. 1992: *The BSD Packet Filter: A New Architecture for User-level Packet Capture*.

Net99, 1999: *Vademecum teleinformatyka - tom 1*. IDG Poland S.A.

PIOTROWSKI, Michał, 2005: *Honeypoty - lep na robaki*. Hakin9, (11).

RAYMOND, Eric, Stevens, September 2003: *The Art of Unix Programming*. Addison-Wesley. ISBN 0131429019.

RUSSELL, Rusty, 29 lip. 2001: *Linux Networking-concepts HOWTO*.

RUSSELL, Rusty, 14 stycz. 2002*a*: *Linux 2.4 NAT HOWTO*.

RUSSELL, Rusty, 24 stycz. 2002*b*: *Linux 2.4 Packet Filtering HOWTO*.

SCHIFFMAN, Mike, luty 2004: *The Evolution of Libnet*.

SCHNEIER, Bruce i MUDGE, list. 1998: *Cryptanalysis of Microsoft's Point-to-Point Tunneling Protocol (PPTP)*. W *Proceedings of the 5th ACM Conference on Communications and Computer Security*.

SCHNEIER, Bruce i MUDGE, 1999: *Cryptanalysis of Microsoft's PPTP Authentication Extensions (MS-CHAPv2)*. W *CQRE '99*, str. 192–203. Springer-Verlag.

SHIMOMURA, Tsutomu, 1995: *How Mitnick Hacked Tsutomu Shimomura with an IP Sequence Attack*. URL `http://www.totse.com/en/hack/hack_attack/hacker03.html`

SILBERSCHATZ, Abraham i Peter GALVIN, 2001: *Podstawy systemów operacyjnych*. Wydawnictwa Naukowo-Techniczne.

STREBE, Matthew i Charles PERKINS, 2000: *Firewalls - ściany ogniowe*. Mikom.

SZYMAŃSKI, Michał, 2003: *Wardriving w praktyce*. Hakin9, (3).

TOMASZEWSKI, Mariusz, Maciej SZMIT i Marek GUSTA, 2005: *Sprzątanie pajęczyn - detekcja nielegalnego współdzielenia łącza*. Hakin9, (11).

TUNG, Brian, grudz. 1996: *The Moron's Guide to Kerberos, Version 1.2.2*.

WATSON, Paul, 30 paźdz. 2003: *Slipping in the Window: TCP Reset attacks*. URL `http://www.osvdb.org/reference/SlippingInTheWindow_v1.0.doc`

WOBST, Reinhard, 2002: *Kryptologia. Budowa i łamanie zabezpieczeń*. Read Me.

ZALEWSKI, Michał, 2001: *Strange Attractors and TCP/IP Sequence Number Analysis*. URL `http://www.bindview.com/Services/Razor/Papers/2001/tcpseq.cfm`

ZALEWSKI, Michał, 2002: *Strange Attractors and TCP/IP Sequence Number Analysis - One Year Later*. URL `http://lcamtuf.coredump.cx/newtcp/`