Bezpieczeństwo w Praktyce

Piotr Praczyk

Data utworzenia :2007-01-11

Streszczenie:

Bezpieczeństwo jest cechą systemu komputerowego, która zapewnia poufność oraz integralność szeroko rozumianych danych przechowywanych w systemie. Tekst ten mówi o kilku ważnych powodach naruszenia bezpieczeństwa, oraz opisuje sposoby zwiększania jego poziomu poprzez przeciwdziałanie zagrożeniom.

Spis treści

“Security is NOT:

- Security is NOT installing a firewall ..
- Security is NOT a Product or Service .. (by Schneier, Bruce)
- Security is Not a Product; It's a Process .. (by Schneier, Bruce)
- A Security Audit is NOT "running a port scan and turning things off" ..

Security is:

- Security is 'Can you still continue to work productively/safely, without compounding the security breach'
- Security is only as good as your 'weakest link'
- Security is 'risk management of your corporate resources(computers) and people'
- Security is 'Can somebody physically walk out with your computers, disks, tapes, .. '
- Security is a Process, Methodology, Policies and People
- Security is 24x7x365 ... constantly ongoing .. never ending
- Security is 'learn all you can as fast as you can, without negatively affecting the network, productivity and budget' “

http://www.linux-sec.net/

1 Błąd przepełnienia bufora

1.1 Czym jest buffer overflow

Możliwość przepełnienia bufora (*ang. Buffer overflow*) jest jednym z najczęściej pojawiających się błędów programistycznych. W wielu językach programowania, jak C, często C++ - do reprezentacji napisów wykorzystywane są zwykłe tablice znaków. Jedynym znacznikiem końca napisu jest wystąpienie znaku o kodzie 0. Większość standardowych funkcji obsługujących napisy, jak *sprintf, strcypy* i inne, nie porównuje długości bufora w którym mają zapisać wynikowy łańcuch znaków, z długością tego napisu. W efekcie, nadpisywane są dane zawarte bezpośrednio za buforem, lub w przypadku kiedy proces nie ma prawa zapisywać w danym fragmencie pamięci, program kończy się z błędem naruszenia ochrony pamięci (pod linuxem *segmentation fault*). Jeśli program kontynuuje działanie po wystąpieniu przepełnienia bufora, wówczas najczęściej niektóre z jego danych są w nieprzewidziany sposób zmodyfikowane. Konsekwencje takiego stanu mogą być bardzo różne w zależności od konkretnego programu.

Rozważmy poniższy przykład:

```
1#include <stdio.h>
2
3int main()
4{
5    char bufor[10];
6    sptintf(bufor,"Wywolanie bledu przepelnienia bufora");
7    return 0;
8}
```

Ponieważ tablica w której przechowywany ma być napis, zawiera miejsce na jedynie 10 znaków, cały napis który chcemy w niej zapisać nie zmieści się. Z tego powodu nastąpi błąd przepełnienia bufora.

Zazwyczaj błąd buffer overflow występuje w nieco innej postaci. Pewne dane tekstowe wczytywane są od użytkownika bez sprawdzenia długości napisu. Taka postać luki pozwala wykorzystać ją w celu zmuszenia programu do wykonania żądanego przez nas kodu. Wynika to ze sposobu, w jaki w wielu językach programowania wywoływane są funkcje.

1.2 Kilka słów o wykorzystywanych technologiach

W dalszej części, jeśli nie zostanie to specjalnie zaznaczone, rozważany będzie zawsze program napisany w języku C (lub korzystający z konwencji wywoływania funkcji charakterystycznej dla tego języka. Będziemy również zakładać, że program wywoływany jest na maszynie o architekturze x86. Znaczy to, że wkładając dane na stos, przechodzimy do danych o coraz mniejszych adresach w pamięci. Kolejnym istotnym założeniem jest to, że kod uruchamiany jest pod kontrolą systemu operacyjnego Linux. Ponieważ w jądrach systemu Linux, serii 2.6.x wprowadzone zostały

mechanizmy zabezpieczające przed podatnością programów na błąd przepełnienia bufora, korzystać będziemy z systemu w wersji 2.4.x. Wszystkie opisane tutaj przykłady uruchamiane były na systemie RedHat Linux w wersji 9.

W przypadku innych architektur, języków programowania, systemów operacyjnych idee przedstawione tutaj pozostają takie same. Zmieniają się jednak szczegóły implementacyjne.

1.3 Wywoływanie funkcji

Zanim program w języku C wywoła funkcję, odkładane są na stos jej parametry. Następuje to w odwrotnej kolejności niż są one wymienione w kodzie źródłowym. Na przykład, jeśli wywołamy funkcję *jakas_funkcja(a,b,c)*, najpierw na stos zostanie włożony argument c, później b a na końcu a. Jest to przydatne przy tworzeniu funkcji przyjmujących zmienną ilość argumentów. Nie jest jednak specjalnie istotne z punktu widzenia wykorzystywania błędu przepełnienia bufora.

Kiedy program wykonujący się na architekturze x86 wywołuje funkcję, wykonywana jest instrukcja call. Powoduje ona odłożenie na stosie adresu powrotu (Jest to adres następnej instrukcji za instrukcją call) umożliwiając kontynuowanie przetwarzania po zakończeniu wykonywania podprogramu. Sterowanie jest następnie przekazywane pod adres wskazany jako argument instrukcji call. W większości przypadków pierwsze instrukcje znajdujące się pod adresem do którego skoczyliśmy wyglądają następująco:

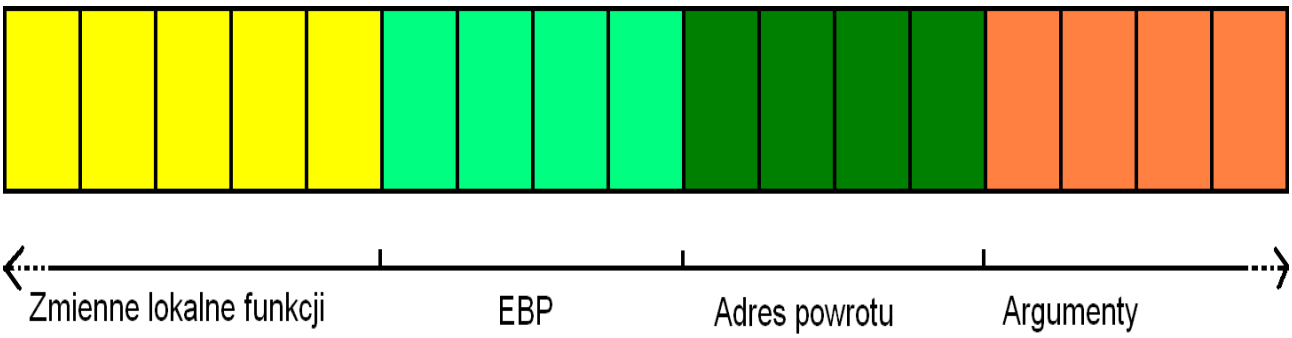
```
1pushl %ebp
2movl %esp,%ebp
3subl N,%esp
```

Najpierw jest więc odkładany na stosie obecny rejestr bazowy (%ebp). Nową wartością rejestru bazowego staje się aktualny znacznik stosu. N jest łącznym rozmiarem zmiennych lokalnych (ewentualnie odpowiednio powiększonym, o czym później). Zmniejszenie wskaźnika stosu o N jest równoważne na architekturze x86 odłożeniu N bajtów na stosie. Jest to miejsce przeznaczone na zmienne lokalne wywoływanej funkcji.

Rejestr bazowy jest bardzo pomocny, ponieważ zmienne lokalne odkładane są również na stosie.

Jeśli wewnątrz kodu funkcji nastąpiło by włożenie czegokolwiek na stos (na przykład wywołanie funkcji), wówczas odwoływanie się do konkretnej zmiennej wymagało by skomplikowanego obliczania jej przesunięcia względem wierzchołka stosu. Mając w rejestrze ebp stan wskaźnika stosu z początku funkcji, zmienne mają zawsze stałe przesunięcia względem wartości tego rejestru. Wracając z funkcji, potrzeba odtworzyć stan tego znacznika sprzed jej wywołania. Z tego powodu wartość ta jest odkładana na stosie.

Struktura stosu w okolicy wierzchołka wygląda więc zaraz po wywołaniu funkcji w następujący sposób :



Adresy rosną od lewej strony w prawo.

1.3.1 Przykład

Rozważmy następujący program:

```
1 void jakas_funkcja(int a, int b)
2 {
3     char bufor1[7];
4     char bufor2[5];
5 }
6 int main()
7 {
8     jakas_funkcja(1,2);
9 }
10
```

Aby zobaczyć w jaki sposób powyższy program zostanie przetłumaczony na asembler, wywołajmy

gcc -S -o przyklad.s przyklad.c

Dostajemy :

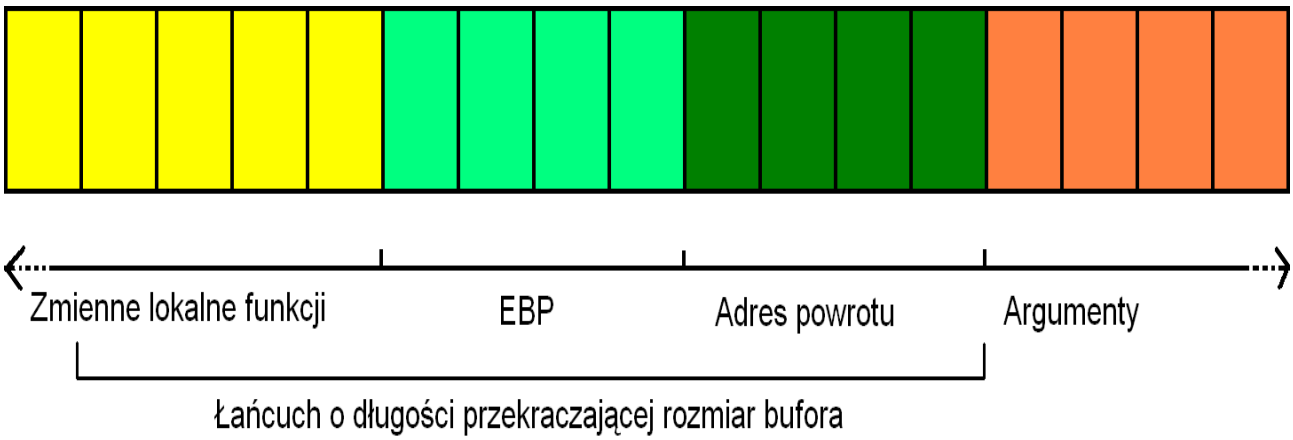
```
        .file   "przyklad1.c"
        .text
.globl jakas_funkcja
        .type   jakas_funkcja,@function
jakas_funkcja:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $40, %esp
        leave
        ret
.Lfe1:
        .size   jakas_funkcja,.Lfe1-jakas_funkcja
.globl main
        .type   main,@function
main:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $8, %esp
        andl    $-16, %esp
        movl    $0, %eax
        subl    %eax, %esp
        subl    $8, %esp
        pushl   $2
        pushl   $1
        call    jakas_funkcja
        addl    $16, %esp
        leave
        ret
.Lfe2:
        .size   main,.Lfe2-main
        .ident  "GCC: (GNU) 3.2.2 20030222 (Red Hat Linux 3.2.2-5)"
```

Na tym przykładzie wyraźnie widać jak odkładane są na stos parametry funkcji, a potem – alokację miejsca na zmienne lokalne.

Pewną niespodzianką jest to, że zamiast rozmiaru zmiennych lokalnych, program rezerwuje na stosie aż 40 bajtów. Liczba ta jest zależna od wersji kompilatora gcc.

1.3.2 Wykorzystanie struktury stosu w celu zmiany adresu powrotu z funkcji

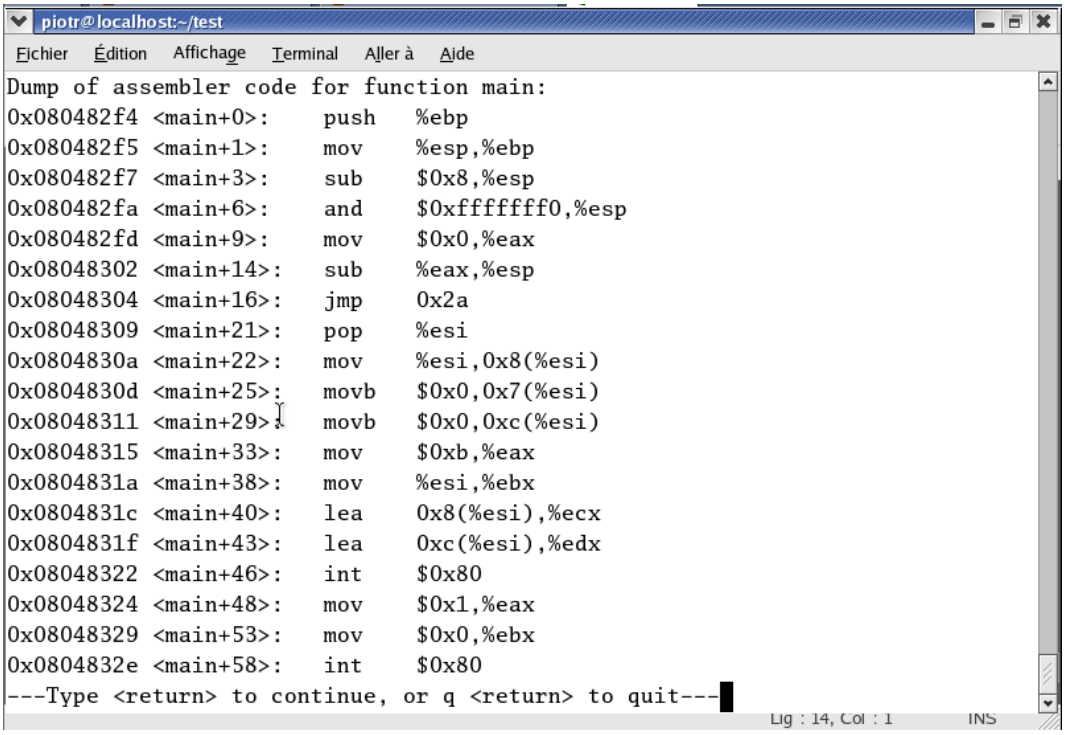
Ważną cechą architektury x86, ułatwiającą eksploatowanie błędów typu buffer overflow, jest to, że stos rośnie w kierunku małych adresów w pamięci. Dzięki temu, jeśli do zmiennej lokalnej pewnej funkcji zapiszemy odpowiednio spreparowany napis, jesteśmy w stanie zmienić adres powrotu z funkcji. Po wykonaniu się funkcja zdejmie ze stosu adres powrotu (który został zmodyfikowany przez przepełnienie bufora), następnie wykona skok pod wskazany adres, różny od adresu następnej instrukcji po wywołaniu funkcji. Ilustruje to poniższy rysunek:



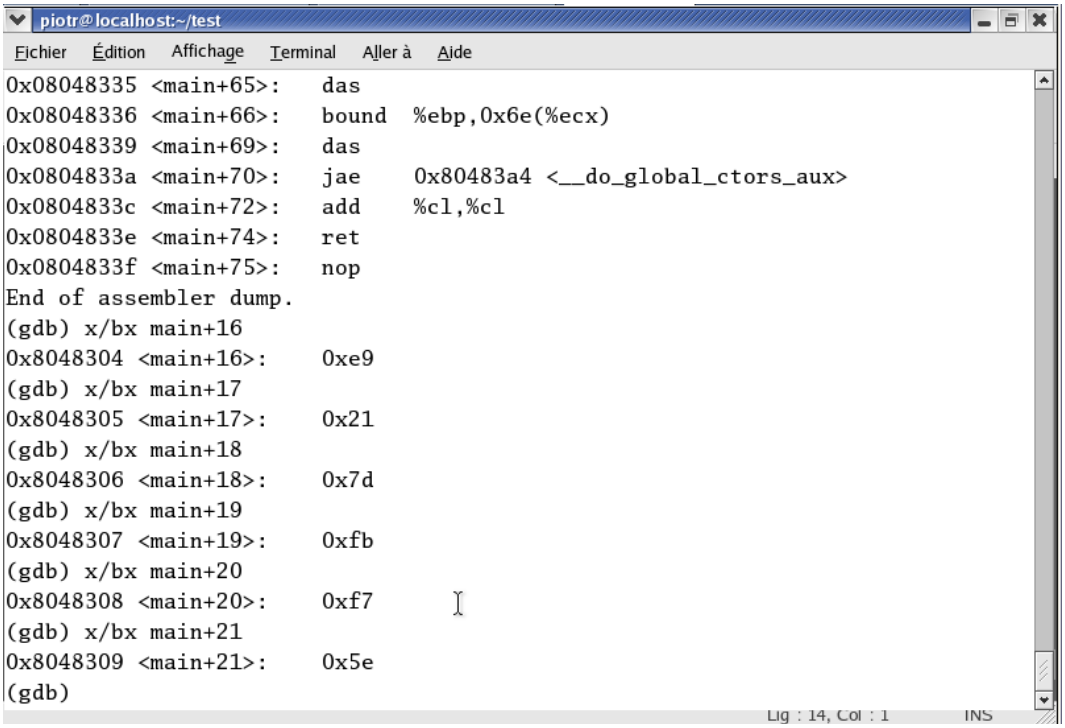
1.4 Uzyskiwanie szesnastkowej reprezentacji utworzonego kodu

Kod nadpisany w buforze musi być bezpośrednim zapisem instrukcji procesora. Z tego powodu musimy znać sposób na uzyskanie tego kodu z napisanego programu. Oto metoda na zrobienie tego

1) Napiszmy program w assemblerze/c (Asembler ma tę przewagę, że dokładniej możemy kontrolować rozmiar i wykonywane instrukcje).
2) Program skompilujmy dodając symbole debuggera.
3) Uruchommy gdb i otwórzmy w nim plik wykonywalny
4) Zdekompilujmy funkcję zawierającą nasz kod. (instarukcja : *disassemble nazwa_funkcji*)



5) Debugger wyświetla informacje o adresach poszczególnych instrukcji assemblera. Posłużmy się nimi w celu obliczenia długości przygotowanego kodu.
6) Za pomocą instrukcji x/bx można teraz uzyskać szesnastkowe reprezentacje kolejnych bajtów pamięci.



7) Z uzyskanych reprezentacji, stwórzmy napis akceptowany przez większość funkcji typu sprintf:

```
\xc9\x21\x7d\xbf\xf7\x5e
```

8) Uzyskany w ten sposób kod wystarczy uzupełnić na początku znakami o kodzie 0x90, a na końcu dopisać adres powrotu z funkcji (zazwyczaj trzeba to wyznaczyć metodąprób i błędów)

1.5 Kilka zasad tworzenia napisu służącego do wykorzystania przepełnienia bufora.

- **Brak binarnych zer**. Ponieważ funkcje obsługujące napisy traktują znak o kodzie zero, jako znacznik końca łańcucha znaków, kod przygotowany celem przejęcia kontroli nad programem, nie może zawierać tego znaku. W przeciwnym przypadku, napis zostanie skopiowany jedynie do tego znaku. W szczególności błąd przepełnienia bufora może w ogóle w takiej sytuacji nie wystąpić. Oto kilka przydatnych sztuczek :
  - Zamiast pisać *movl $0,%eax*, można napisać. *Xorl %eax,%eax.*
    - Przypisanie małej wartości do rejestru może się odbywać w 2 krokach. :
      - Wyzerowanie rejestru
      - Przypisanie liczby do mniej znaczącej części (stosując ewentualne przesunięcia binarne)
- **Pobranie adresu zmiennej** może się odbywać przez wykonanie skoku względnego z początku kodu, do instrukcji stojącej przed zmienną, następnie wykonanie instrukcji call za instrukcję skoku. W tym momencie na stosie znajduje się adres zmiennej.

```
Jmp przesuniecie_do_call
popl %eax    # teraz EAX zawiera adres bufora
...
...
call przesuniecie_do_popl
.string „/bin/sh”
```

- Ponieważ zgadnięcie adresu początku bufora może być trudne, można **wypełnić cały bufor znakami o kodzie 0x90** (instrukcja nop), kod umieścić pod koniec bufora, a adres powrotu zgadywać. Przy takiej modyfikacji nie trzeba podać dokładnego adresu powrotu. Jeśli bufor będzie odpowiednio długi, wystarczy trafić gdzieś w instrukcje nop poprzedzające właściwy kod. Znacznie zmniejsza to ilość prób potrzebną do wykonania swojego kodu.
- Dobrze jest używać **bezpośrednich odwołań systemowych (przerwanie 0x80)** zamiast skoków do funkcji bibliotecznych. Uwalnia to programistę od wielu problemów związanych ze zgadywaniem adresu, pod którym załadowana jest biblioteka..
- Wygodnie jest dodać **kilka instrukcji nop przed i za kodem,** który chcemy wyciągnąć z pliku binarnego. Ułatwi to rozpoznanie fragmentu, gdzie ów kod się znajduje.

1.6 Przeciwdziałanie

1.6.1 Sprawdzanie rozmiaru napisów wkładanych do buforów

Najbardziej oczywistą metodą zapobiegania błędom przepełnienia bufora jest oczywiście pisanie poprawnego kodu. Zanim jakiekolwiek dane zostaną wpisane do bufora, należy sprawdzać ich długość.

Zazwyczaj niebezpiecznymi fragmentami programu są konstrukcje w rodzaju:

```
int i=0;
char bufor[ROZMIAR];
while ((bufor[i] =
read_character()) !='\n') i++;
bufor[i]=0;
```

1.6.2 Randomizacja przestrzeni adresowej procesu (Adress Space Layout Randomization)

Opisana tutaj metoda ma na celu przeciwdziałanie wykorzystywaniu błędów przepełnienia bufora poprzez wykonywanie skoków do funkcji bibliotecznych.

W standardowej sytuacji, biblioteki są ładowane pod z góry przewidziany adres w pamięci. Z tego

powodu mając pewną wiedzę na temat exploitowanego programu, można wykonywać dowolne funkcje z tych bibliotek. Ideą tego zabezpieczenia jest to, aby ładować biblioteki pod losowe adresy w pamięci operacyjnej.

1.6.2.1 ASLR w Windows Vista

W Windows Vista przyjęto model, gdzie biblioteki dzielone (DLL) oraz pliki wykonywalne (EXE) mogą zostać załadowane pod jeden z 256 różnych adresów. Wybór dokonywany jest losowo. Nie eliminuje to całkowicie problemu, lecz zmniejsza szanse trafienia na kod biblioteki. Technika ta stosowana razem z innymi, zmniejsza prawdopodobieństwo udanego wykorzystania błędu przepełnienia bufora.

1.6.2.2 ASLR w Linuksie

Za implementację mechanizmów bezpieczeństwa w systemie Linux, odpowiedzialny jest projekt PAX. Zawiera on moduł, który pozwala na randomizację przestrzeni adresowej wykonywanych procesów. Również nowe jądra serii 2.6.x zawierają wbudowane tego typu mechanizmy.

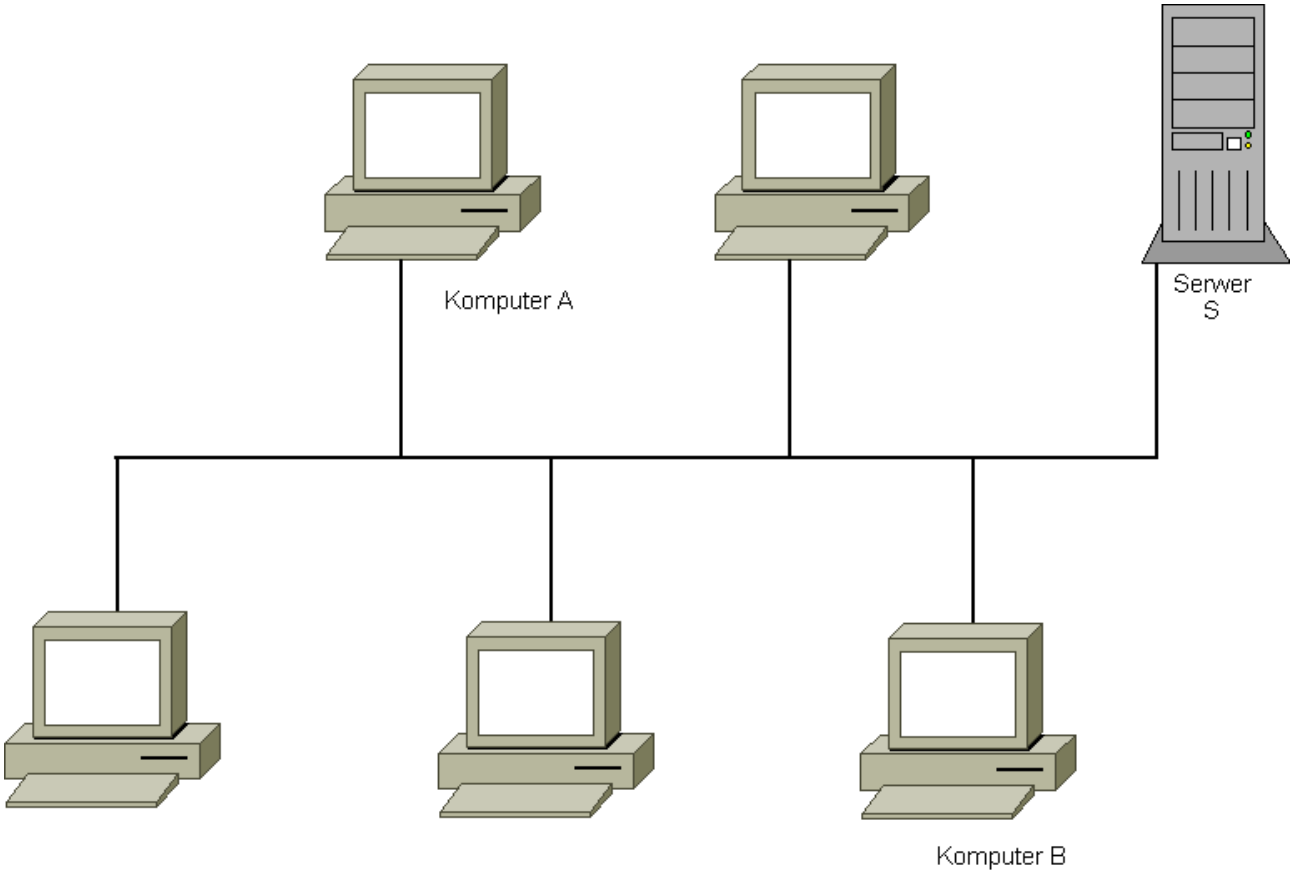
2 Problemy z bezpieczeństwem w sieci lokalnej

Specyfikacja protokołu TCP/IP, Ethernet, które (na 2 różnych poziomach) są najczęściej wykorzystywane do konstrukcji sieci lokalnych, sprawiają, że w sieciach lokalnych ruch odbywa się w sposób nie szyfrowany. Dane wysyłane są w porcjach zwanych pakietami. Każdy pakiet zawiera między innymi adres swojego odbiorcy. O ile nie zachodzi żadne filtrowanie danych (na przykład za sprawą routerów, switchów itp), wszystkie informacje wysłane przez, oraz adresowane do dowolnego komputera z sieci lokalnej, są widoczne dla wszystkich maszyn. To, które informacje zostaną uznane za adresowane do konkretnej maszyny, jest ustalane już przez sam system operacyjny. Jeżeli mamy dostęp do komputera podłączonego do sieci lokalnej, oraz odpowiednie uprawniani w działającym na nim systemie operacyjnym, możemy więc podsłuchiwać wszystkie dane, które przepływają przez sieć lokalną. Programy umożliwiające takie działanie nazywają się *snifferami*.

2.1 Przykład wykorzystania podsłuchiwania ruchu sieciowego w celu przejęcia poufnych informacji

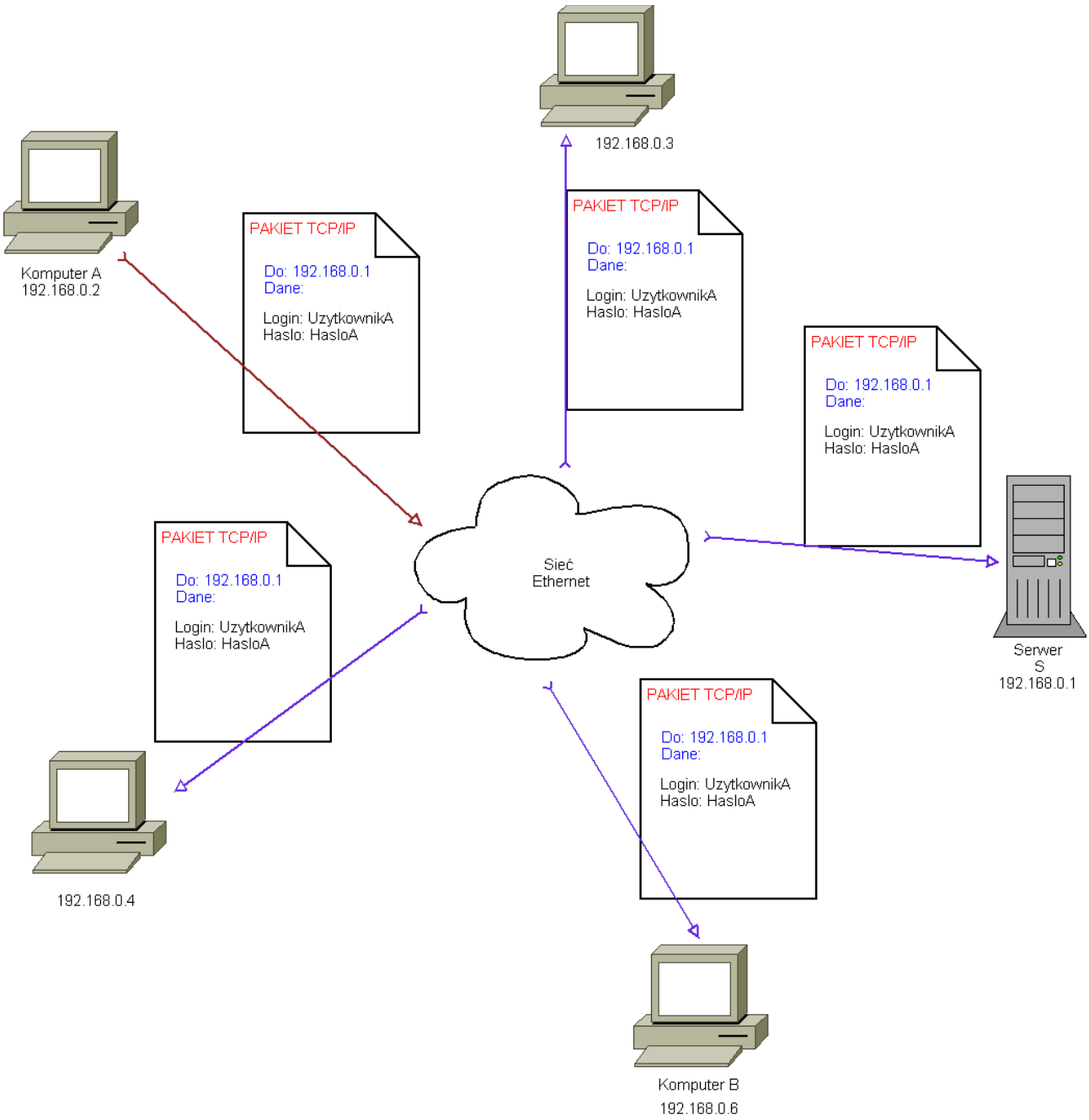
2.1.1 Ogólny scenariusz

Zobaczmy co się dzieje, kiedy ktoś chce uzyskać dostęp do pewnej usługi udostępnionej w sieci.



Użytkownik komputera A chce zalogować się do jakiejś usługi na serwerze (S). W tym celu wysyła

do serwera swój identyfikator oraz hasło. Dane te są umieszczone w pewnym pakiecie TCP/IP (dla uproszczenia, bez straty ogólności można złożyć, że znajdują się zawsze w jednym). W pakiecie zawierającym dane autoryzacji, zapisywany jest między innymi adres IP serwera. Pakiet jest następnie transmitowany za pomocą łącza Ethernet.



Pakiet trafia do wszystkich komputerów znajdujących się w sieci lokalnej. System operacyjny na wszystkich komputerach, poza serwerem (S) ignoruje pakiet, jako nie adresowany do tej właśnie maszyny. System operacyjny na maszynie S odczytuje pakiet, wyciąga przesyłane w jego wnętrzu dane i przekazuje je do odpowiedniej aplikacji, która ma dokonać weryfikacji danych użytkownika.

Przedstawiony powyżej scenariusz zmodyfikujmy nieco. Przyjmijmy, że komputer B posiada zmodyfikowany system operacyjny, który zamiast odrzucać nie adresowane do niego pakiety,

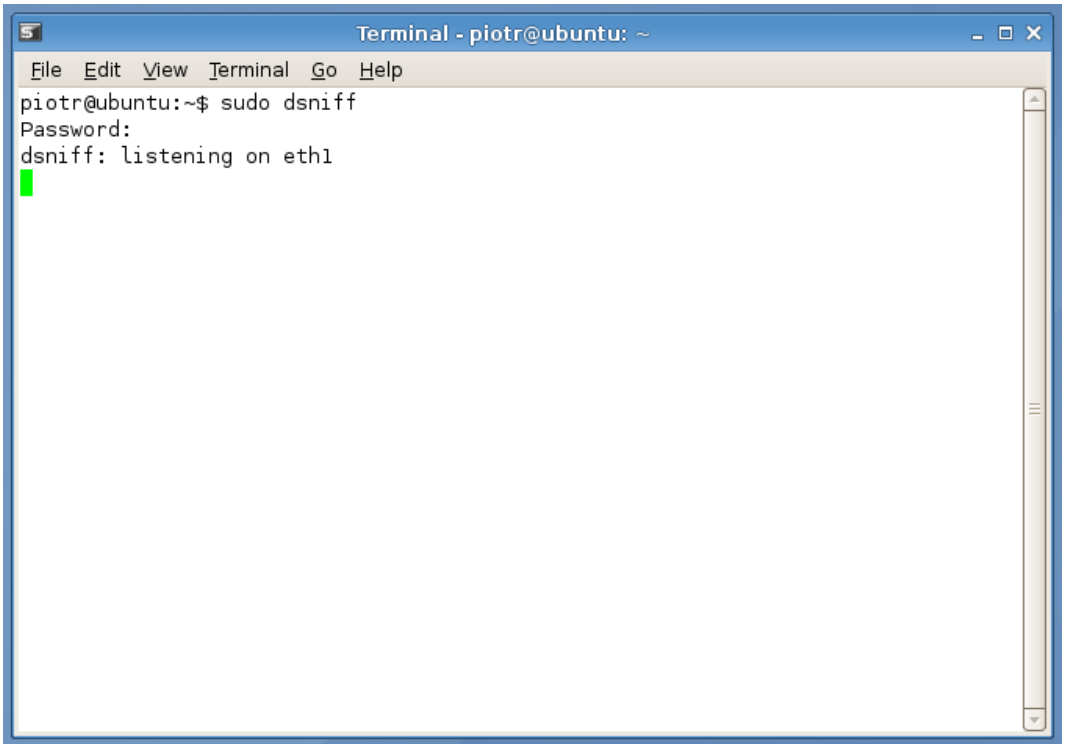
przekazuje ich zawartość do specjalnego programu zwanego *snifferem*. Program ten jest w stanie odczytać wszystkie poufne informacje zawarte w pakiecie. Dzięki temu użytkownik komputera B jest w stanie zalogować się teraz do usługi serwera S, jako użytkownik komputera A.

2.1.2 Przykład przechwycenia danych logowania przy wykorzystywaniu protokołu Telnet

Telnet jest popularnym protokołem stosowanym do zdalnego przeprowadzania sesji na innym komputerze. W tym miejscu pokażę, jak wykorzystać słabość tego protokołu (brak szyfrowania przesyłanych danych) w celu przechwycenia loginu i hasła użytkownika. Za serwer usługi Telnet posłuży komputer z zainstalowanym Windows XP. Komputer z którego loguje się użytkownik działa pod kontrolą systemu Linux, podobnie jak komputer intruza.

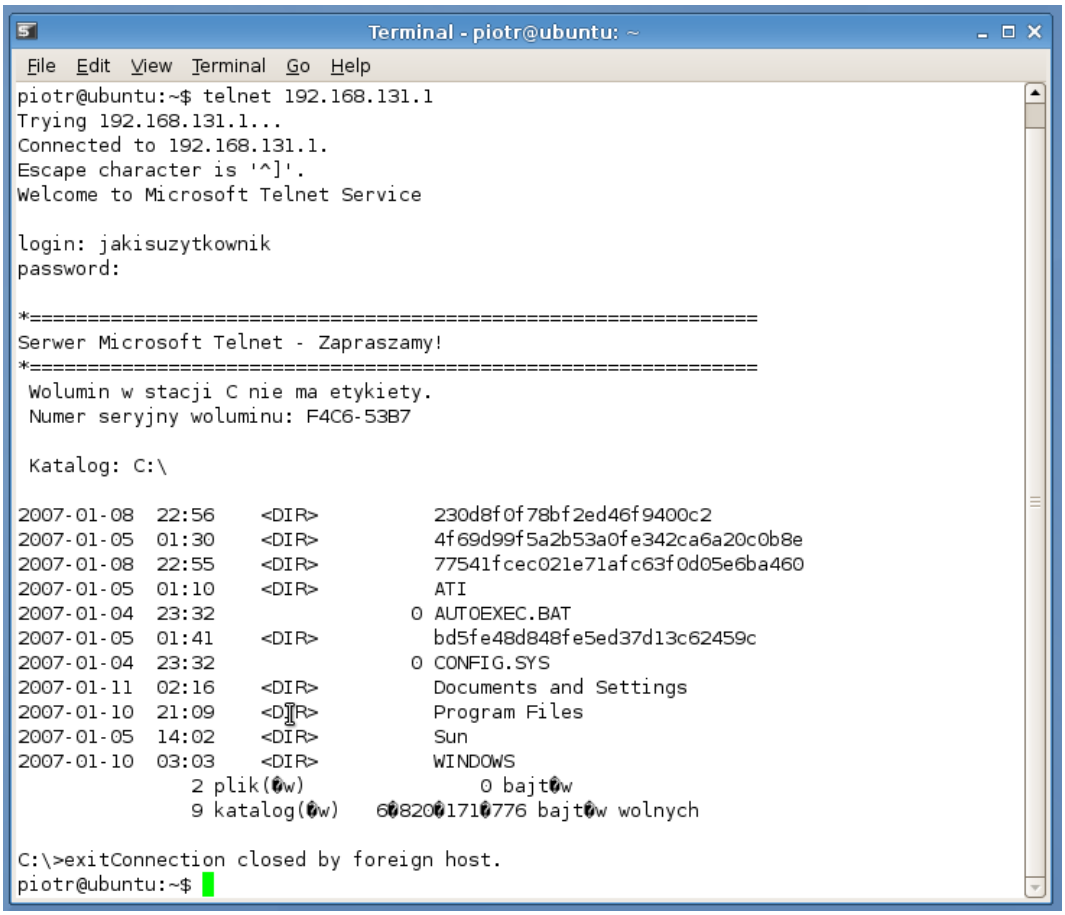
Do przechwytywania ruchu sieciowego można użyć pod Linuxem programu *dsniff*. Nie dość, że przechwytuje on wygenerowany w sieci ruch, lecz również analizuje zawartość pakietów TCP wyświetlając ich zawartość z czytelnym komentarzem.

Chcąc przechwycić dane logowania, zacznijmy od uruchomienia sniffera na komputerze intruza.

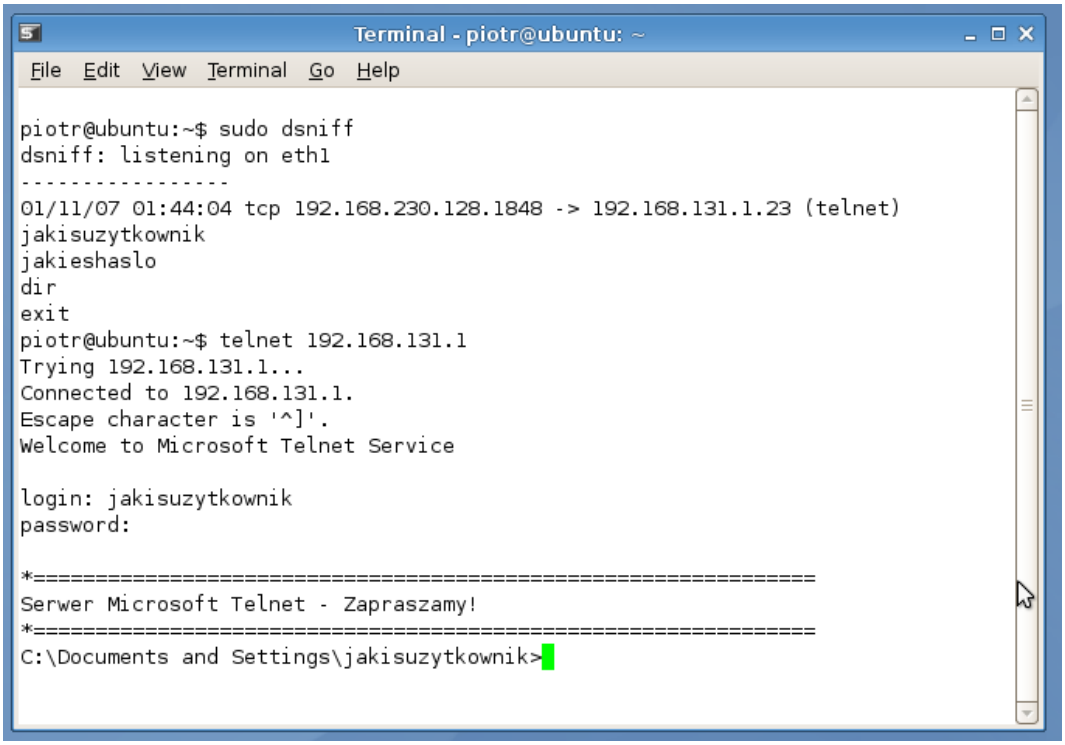


Program *sudo* pozwala na uruchomienie innego programu z prawami administratora. *Dsniff* został, jak widać, uruchomiony. Teraz wystarczy czekać na pierwszych nie podejrzewających niczego użytkowników.

W międzyczasie uytkownik postanowił zalogować się do serwera Telnet i przeprowadzić zdalną sesję:



Intruz przechwycił login i hasło a następnie wykorzystał je celem uzyskania dostępu do serwera Telnet:



W identyczny sposób można podsłuchiwać wiele innych protokołów. Wśród nich:

- FTP
- HTTP
- POP
- SMTP
- Microsoft SMB (W nowszych wersjach Microsoft Windows, loginy i hasła są szyfrowane)
- IMAP
- IRC

Jak widać, możliwości wynikające z podsłuchiwania są dość szerokie. Od czytania wysyłąnej przez inne osoby poczty, po przechwytywanie wszelkiego rodzaju haseł. Szczególnie ważna jest zatem obrona przed tego rodzaju atakami.

2.2 Sposoby obrony przed podsłuchem w sieci

2.2.1 Dobre projektowanie infrastruktury sieciowej

Najprostszym sposobem zabezpieczenia się przed podsłuchem w sieci jest jej odizolowanie. Aby móc podsłuchiwać ruch sieciowy potrzebne są duże uprawnienia w systemie operacyjnym na jednym z podpiętych do niej komputerów. W jakiś sposób uniemożliwione powinno być podłączanie własnych komputerów bezpośrednio do sieci lokalnej. Pomiędzy miejscem wpięcia zewnętrznych komputerów lub Internetu, powinien znajdować się *firewall,* lub przynajmniej ruch powinien być filtrowany względem adresów IP komputerów znajdujących się w poszczególnych podsieciach.

Dobrze jest, jeśli sieć lokalna podzielona jest na kilka podsieci, pomiędzy którymi ruch jest filtrowany. Nie dość, że zapobiega to nadmiernemu obciążeniu sprzętu sieciowego, ale również zwiększa bezpieczeństwo przesyłania danych.

2.2.2 Stosowanie protokołów obsługujących kryptografię

Większość nieszyfrowanych protokołów posiada swoje szyfrowane odpowiedniki. Chociaż nie zapewnia to całkowitego bezpieczeństwa (o czym później), dobrym zwyczajem jest stosowanie ich zamienników. Oto niektóre z nich

- HTTP - HTTPS
- FTP - SCP, SFTP
- TELNET - SSH

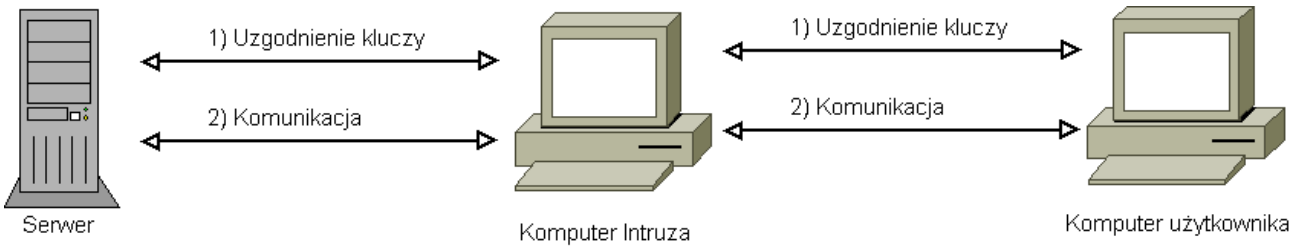
2.2.3 SSL

SSL (*ang. Secure Socket Layer*) jest technologią umożliwiającą przesyłanie innego protokołu przez szyfrowany kluczem asymetrycznym kanał. Pierwotnie został opracowany przez firmę Netscape Communications Corporation. Obecnie rozwijana jest jego pochodna o nazwie TSL (*ang. Transport Layer Security*).

2.3 Słabości kryptografii asymetrycznej.

Pomimo, że w przypadku kryptografii stosującej algorytmy asymetryczne (Np. RSA, którego opis znaleźć można między innymi na wykładzie pierwszego roku), odszyfrowanie danych nie wchodzi w grę (poza wersją 1 protokołu SSL, gdzie intruz miał możliwość wymuszenia na stornach transmisji najsłabszej możliwej kryptografii). Możliwy jest jednak inny scenariusz, nazywany atakiem *Monkey in The Middle* (lub *Man in The Middle*).

Rozważmy następującą sytuację:



1) Intruz w jakiś sposób przejmuje kontrolę nad komputerem przez który przechodzi ruch sieciowy. (Albo przez fizyczne podłączenie się do sieci, albo przez włamanie się na odpowiedni router).
2) Intruz przechwytuje żądanie nawiązania szyfrowanego połączenia, które komputer użytkownika nadał do serwera.
3) Intruz nawiązuje szyfrowane połączenie z serwerem, wymieniając się z nim kluczami publicznymi.
4) Intruz generuje swoją własną parę kluczy (publiczny/prywatny) i wymienia się z klientem kluczami publicznymi.

5) Cała transmisja pomiędzy komputerem użytkownika a serwerem jest po drodze odszyfrowywana, a następnie znów szyfrowana (innym kluczem, będącym w posiadaniu intruza) i przesyłana dalej.

W ten oto sposób, bez łamania szyfru, intruz jest w stanie odczytać całą „zabezpieczoną" transmisję.

2.3.1 PKI (Public Key Infrastructure)

Lekarstwem na powyższy sposób przechwytywania poufnych danych, ma być tzw. Infrastruktura Kluczy Publicznych. Jest to system szeroko pojętego zarządzania kluczami. Do najważniejszych jego zadań należą:

- Generowanie kluczy kryptograficznych
- Weryfikacja tożsamości stron
- Wystawianie i weryfikacja certyfikatów
- Podpisywanie transmisji
- Szyfrowanie transmisji
- Znaczenie danych aktualnym czasem

Pakiet *dsniff* jest wyposażony w narzędzie do przeprowadzania ataków MITM. Nazywa się ono webmitm.

3 Robaki sieciowe

Robak komputerowy jest programem bardzo podobnym do wirusa. Różnice jest jednak dość zasadnicza. Wirus potrzebuje nosiciela, którym może być plik wykonywalny, dokument zawierający skrypty itp, zaś robak jest samodzielnym bytem. Również rozprzestrzenianie się robaków ma inny charakter niż przypadku wirusów. Najczęściej wykorzystują one znane dziury bezpieczeństwa, a do rozprzestrzeniania się wykorzystują sieć. Z tego powodu szybkość infekcji kolejnych systemów jest bardzo duża.

3.1 Rodzaje robaków sieciowych

Robaki sieciowe można sklasyfikować ze względu na sposób rozprzestrzeniania się. Najważniejsze z rodzajów to:

3.1.1 Robaki E-mailowe

Przesyłane razem z wiadomościami e-mail. Zazwyczaj, użytkownik jest przekonywany do uruchomienia programu zawierającego robaka za pomocą treści e-maila. (Zaskakująco wiele osób nabiera się na proste sztuczki socjotechniczne). Po uruchomieniu, robak rozsyła się zazwyczaj do osób zapisanych w książce adresowej programu pocztowego.

3.1.2 Robaki komunikatorów internetowych

Rozprzestrzeniają się poprzez wysyłanie linku do zainfekowanego programu, do wszystkich osób znajdujących się na liście kontaktów osoby, która uruchomiła robaka.

3.1.3 Robaki IRC

Podobnie, jak w przypadku robaków komunikatorów internetowych. Rozsyłają się do wszystkich osób z danego kanału irc. Albo bezpośrednio przez protokół Irc, albo przez podanie linka do programu.

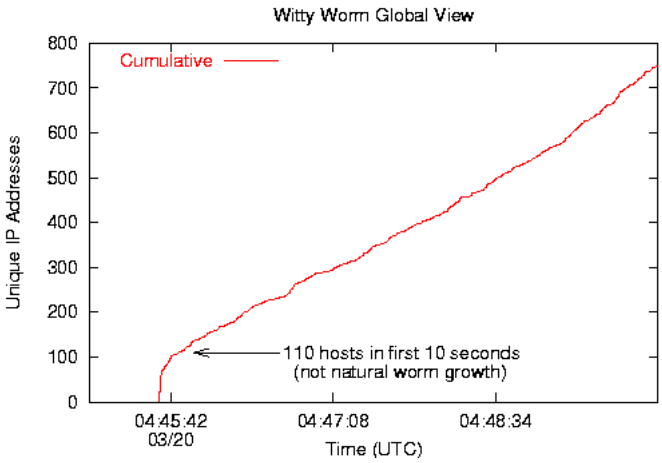
3.1.4 Robaki sieci wymiany plików

Robak kopiuje się do dzielonego katalogu klienta peer to peer. Zazwyczaj ukrywa się pod niewinnie brzmiącą nazwą, co zwiększa szanse pobrania go przez innych użytkowników sieci P2P.

3.1.5 Robaki internetowe

Programy tego typu wykorzystują niskopoziomowe błędy, np. W obsłudze protokołu TCP/IP. To one właśnie stanowią największy problem ze względu na szybkość rozprzestrzeniania się.

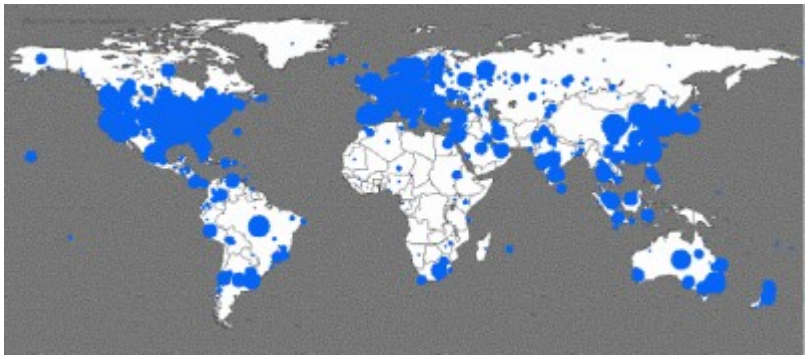
Ilustruje to poniższy wykres:



(Pobrane z adresu www.caida.org)

Już w ciągu pierwszych dziesięciu sekund, robak Witty zainfekował 110 maszyn. Tempo rozprzestrzeniania się robaków sieciowych ma zazwyczaj aż do momentu nasycenia sieci, charakter wykładniczy. (Proces rozprzestrzeniania się robaka można opisać prostym systemem dynamicznym zgodnym z modelem populacji). Tempo infekcji jest więc zastraszające.

Na poniższej mapce zaznaczone są rejony świata dotknięte robakiem slammer, 30 minut po wypuszczeniu opublikowaniu robaka. Są to praktycznie wszystkie dobrze rozwinięte regiony!



(Obraz pobrany z www.ahpcrc.org)

Zazwyczaj głównym problemem nie jest destrukcyjny kod zawarty w robakach, lecz ruch sieciowy generowany przez ich rozsyłanie.

3.1.5.1 Blaster

W sierpniu 2003, komputery z zainstalowanym systemem operacyjnym Windows XP lub Windows 2000, zostały zaatakowane przez robaka o nazwie Blaster (znanego również jako Lovsan).

Do rozprzestrzeniania się robak wykorzystywał błąd w podsystemie RPC

3.1.5.2 Slammer

Robak wykorzystujący błędy w Microsoft SQL Server

3.1.5.3 Zotob

Wypuszczony w 2005 roku robak wykorzystujący błędy w Microsoft Windows.

3.2 Walka z robakami sieciowymi

3.2.1 Instalowanie najnowszych uaktualnień systemu

Bardzo ważnym elementem walki z zagrożeniami opisywanego tutaj typu jest posiadanie najnowszych uaktualnień bezpieczeństwa dla posiadanego systemu operacyjnego. Po wykryciu błędu, zazwyczaj w ciągu kilku dni producent oprogramowania publikuje łatę, która usuwa problem. W przypadku administrowania serwerem, warto również śledzić listy dyskusyjne poświęcone bezpieczeństwu.

3.2.2 Nieinstalowanie podejrzanego oprogramowania z sieci

Niezwykle prosta metoda pozwalająca zapobiec większości zagrożeń. Tylko nieliczna kategoria robaków sieciowych wykorzystuje do rozprzestrzeniania się, błędy w systemie operacyjnym. Znaczna większość bazuje na naiwności bądź niewiedzy użytkownika.

3.2.3 Nachi – Robak, który zwiększa bezpieczeństwo systemów

Ciekawym przykładem wykorzystywania robaków sieciowych w dobrym celu, jest rodzina „Nachi", znana również pod nazwą „Welchia**".** Do rozprzestrzenia się wykorzystywały one błąd w podsystemie wywoływania zdalnych procedur (RPC) w Microsoft Windows. Program po zainfekowaniu komputera, ściągał z Internetu łaty, których celem było wyeliminowanie błędu dzięki któremu dostał się do systemu.

3.2.4 Network Intrusion Detection Systems

Systemy typu IDS mają na celu wykrycie wszelkich niepożądanych manipulacji na systemie komputerowym. W szczególności, rozwiązani NIDS koncentrują się na bezpieczeństwie sieciowym. Wśród zagrożeń przed którymi mają chronić NIDS'y wyróżniamy:

- Ataki DOS (Denial Of Service)
- Skanowanie portów
- Włamania do systemu za pomocą monitorowania ruchu sieciowego.

Ponieważ programy tego typu nie ograniczają się do prostego filtrowania danych, lecz dokonują również ich analizy, spełniają nieco inną funkcje niż *firewall'e*. Na przykład duża ilość wywołań TCP odwołujących się do wielu różnych portów, nadchodząca w bardzo krótkim czasie, zostanie zakwalifikowana jako próba skanowania portów.

Często analizowany jest nie tylko ruch przychodzący, ale również i ten, który opuszcza sieć lokalną (lub komputer). Czasami systemy NIDS połączone są ze sobą, dynamicznie tworząc różnego rodzaju czarne listy dla *firewall'ów*.

3.2.4.1 Snort

Jednym z trybów działania programu Snort jest praca jako system NIDS. Zaprezentuję tutaj przykład zastosowania tej aplikacji w celu wykrycia próby skanowania portów TCP/IP.

Skanowanie portów jest potencjalnie niebezpiecznym zdarzeniem. Większość ataków na systemy

informatyczne rozpoczyna się od rozpoznania, w jaki sposób system ten jest zbudowany. Dzięki skanowaniu portów można stwierdzić, jakie usługi są na serwerze uruchomione. Czynność ta polega na nawiązywaniu dużej ilości połączeń z kolejnymi portami danego serwera. Charakterystyczne jest to, że duża większość odwołań kończy się błędem.

Załóżmy, że posiadamy wersję programu udostępnianą razem z dystrybucją Linuxa Ubuntu 6.10. W przeciwnym wypadku należy zadbać, aby plik konfiguracyjny znajdował się w odpowiednim miejscu (W Ubuntu jest to /etc/snort/snort.conf), oraz aby była w nim włączona opcja wykrywania prów skanowania portów. Domyślnie jest ona włączona.

Najpierw stwórzmy katalog log, w którym będą przechowywane wyniki działania Snort'a.

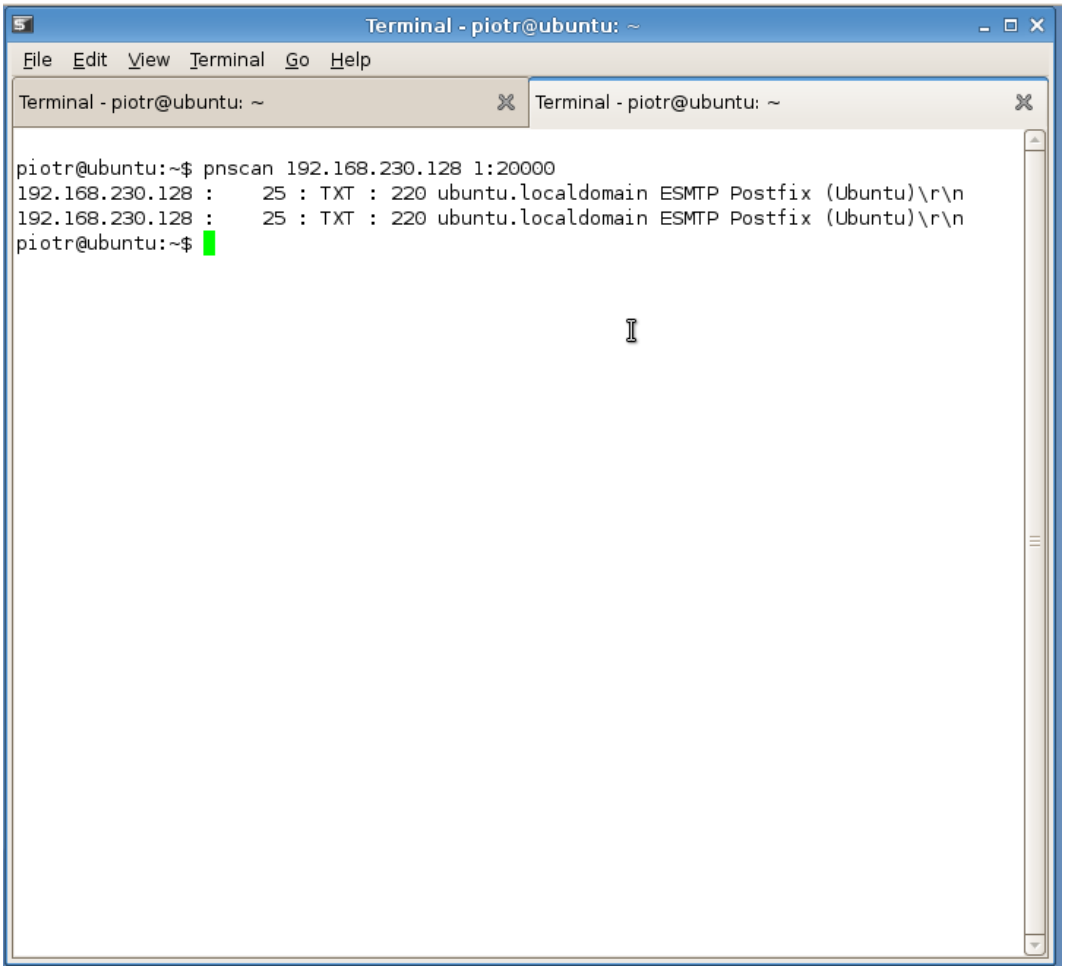
Aby uruchomić program jako NIDS, należy wykonać polecenie:

sudo snort -d -h 192.168.130.0/24 -c /etc/snort/snort.conf -l ./log

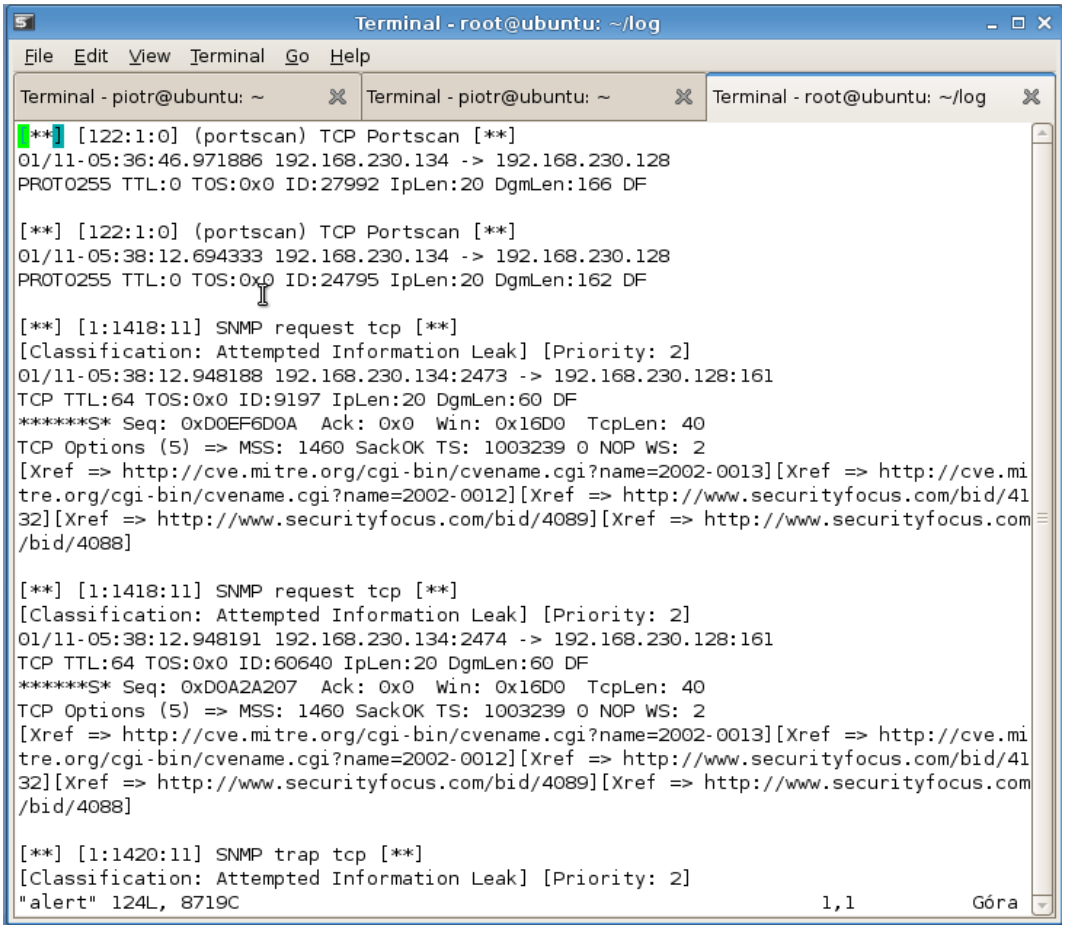
Od tego momentu wszystkie podejrzane zdarzenia są rejestrowane.

Teraz uruchommy z innej maszyny (znajdującej się w podsieci 192.168.230.255) skaner portów.

Posłużę się do tego celu programem *pnscan.*



Zobaczmy, czy Snort cokolwiek zauważył. W tym celu otwórzmy plik ./log/alert:



Jak widać, próba rozpoznania systemu nie uciekła uwadze naszego monitora bezpieczeństwa.

Wykrywanie prób skanowania portów jest tylko jedną z funkcji programu Snort. System ten jest bardzo rozbudowany, więc nie sposób opisać go dokładniej w tym miejscu. Ciężko byłoby przy małych zasobach sprzętowych wykonać symulację wykrycia ataku przez robaki sieciowe, w związku z tym w tym miejscu ograniczę się do skanowania portów. Bardziej szczegółowe informacje można znaleźć pod adresem http://www.snort.org.

3.2.5 Intrusion Prevention Systems

Wadą systemów IDS jest to, że jedynie wykrywają próby włamania, wszelkie akcje pozostawiając w gestii administratora. Podejście takie jest bardzo mało skuteczne np. W przypadku robaków sieciowych. Na wcześniejszych rysunkach widoczne było, jak szybko rozszerza się zagrożenie z nimi związane. Ręczna reakcja administratora była by zawsze zbyt późna. Również opracowanie nowych technik ataku sprawiało, że tradycyjne podejście nie sprawdzało się. Z tego powodu zaczęto rozwijać systemy IPS (ang. *Intrusion Prevention Systems*).

Zgodnie z informacjami zawartymi w dokumencie http://www.mcafee.com/us/local_content/white_papers/wp_host_nip.pdf, System IPS można zdefiniować jako taki, który chroni:

- **Poufność informacji** zapisanej na elektronicznym nośniku. (Zabezpiecza przed nieautoryzowanym kopiowaniem i otwieraniem, w tym przed programami czytającymi klawiaturę, back-dorami etc.)
- **integralność** informacji (zabezpiecza przed modyfikacją danych)
- **dostępność** informacji (dla autoryzowanych użytkowników)

Istnieją dwa podejścia mające na celu osiągnięcie powyższych celów.

- Pierwsze z nich polega na zainstalowaniu specjalnego oprogramowania bezpośrednio na chronionym komputerze.
- Drugie polega na podłączeniu do sieci lokalnej hosta pełniącego funkcję IPS

Oba z powyższych podejść mają swoje zalety. Ograniczę się tutaj jedynie do wymienienia ich.

3.2.5.1 Zalety systemu IPS działającego bezpośrednio na chronionym komputerze

- Oprogramowanie zainstalowane bezpośrednio na chronionym systemie chroni nie tylko przed atakiem, lecz również przed jego skutkami. (Poprzez zablokowanie możliwości zapisania pliku przez program etc..)
- Chroni laptopy nawet kiedy nie są podłączone do bezpiecznej sieci
- Chroni przed atakami, w których intruz ma dostęp bezpośrednio do maszyny.
- Stanowi ostatnią deskę ratunku, jeśli wszystkie sieciowe mechanizmy obronne zawiodły.
- Chroni przed atakami z sieci lokalnej
- Chroni przed zaszyfrowanymi atakami (kiedy kod przeznaczony do dokonania włamania jest odszyfrowywany dopiero na lokalnej maszynie)
- Nie jest zależny od architektury sieci lokalnej.

3.2.5.2 Zalety systemu IPS działającego w sieci lokalnej

- Jedna instalacja systemu może chronić bardzo dużą liczbę maszyn.
- Łatwe wdrożenie (Nie potrzeba powtarzać tej samej czynności na wielu komputerach)
- Monitorując ruch w całej sieci lokalnej jednocześnie, IPS działający na dedykowanym serwerze może dokonywać szerszej analizy danych niż z punktu widzenia konkretnego hosta.
- Może chronić zasoby nie będące komputerami (jak drukarki etc.)
- Jest neutralny ze względu na platformę. Chroni maszyny niezależnie od tego jakie oprogramowanie, w tym system operacyjny, są na nich uruchomione.
- Chroni przed atakami polegającymi na sparaliżowaniu działania sieci przez wywołanie nadmiernego ruchu.

Jak widać, najlepszy efekt można osiągnąć łącząc zalety obu tych rozwiązań.

4 Źródła dalszych informacji

Przedstawione tutaj problemy i sposoby radzenia sobie z nimi stanowią jedynie pobieżny przegląd najważniejszych zagadnień związach z bezpieczeństwem. Nie jest on ani kompletny, ani wyczerpujący. Poniżej przedstawiam kilka adresów internetowych, pod którymi można znaleźć przydatne informacje dotyczące tego tematu.

1) http://www.linuxsecurity.com/
2) http://www.linux-sec.net/
3) http://tldp.org/HOWTO/Security-HOWTO/
4) www.phrack.org
5) http://www.mcafee.com/us/local_content/white_papers/wp_host_nip.pdf