Metody włamań do systemów komputerowych - bezpieczne programowanie i robaki

Paweł Kozioł

16 stycznia 2003

Spis treści

Rozdział 1

Bezpieczne programowanie

1.1 Wstęp

W pierwszej części referatu chciałbym zwrócić uwagę na błędy w programach, które prowadzą do obniżenia ich bezpieczeństwa, a przez długi czas mogą pozostać niewykryte. Omówię najważniejsze typy błędów popełniane przez programistów i analityków systemowych, a także zaprezentuję kilka krytycznych błędów popełnionych przez programistów. Omówię również techniki wykorzystywania błędów w programach w celu włamania się do systemu.

Aby uświadomić sobie rangę problemu należy zwrócić uwagę na istnienie wielu list dyskusyjnych poświęconych błędom w oprogramowaniu takich jak Bugtraq czy Microsoft Security Bulletin. Na tych listach pojawiają się regularnie powiadomienia o kolejnych błędach i uaktualnieniach oprogramowania. Czasami ilość takich powiadomień dochodzi do kilkunastu dziennie.

Najczęściej błędy prowadzą do nieprawidłowego wykonania programu czyli awarii, ale możliwe jest też wykorzystanie ich do włamania się do systemu, czyli uzyskania uprawnień większych niż przewidziane przez administratora systemu. W tym referacie skupię sie na błędach, które są niebezpieczne ze względu na możliwość wykorzystania ich w celu włamania sie do systemu.

1.2 Najczęstsze źródła błędów w programach

1.2.1 Błędy implementacyjne

Błędy implementacyjne powstają na skutek niuważnego kodowania, niedostatecznego zrozumienia wymagań lub braku umiejętności programisty.Takie błędy

mogą pozostać ukryte w programach z powodu zastosowania niedokładnych procedur testowania i przeglądania kodu.

PRZEPEŁNIENIE BUFORA

Jest to najważniejszy błąd, którego w zasadzie nie da się wyeliminować. Zostanie omówiony w dalszej części referatu.

Synchronizacja wątków wykonywanych współbieżnie

Bląd ten jest bardzo trudny do przewidzenia i do wyśledzenia. Powstaje, gdy kilka wątków używających tego samego obiektu (struktury danych) albo pozostawia go w stanie niespójnym albo kiedy spreparowany kod wykorzystuje niedostatecznie chronione zasoby, używane przez inny wątek. Rozwiązaniem problemu jest dodanie prawidłowej synchronizacji.

Taki błąd został popełniony przez programistów z firmy Sun w wirtualnej maszynie Javy w programie JDK. W lutym 2001 roku Sun opublikował następujące zawiadomienie: „Błąd w niektórych wersjach Java Runtime Environment pozwala spreparowanemu programowi w Javie na wykonanie w systemie dowolnego polecenia, do którego wykonania użytkownik może nie być upoważniony. Jednakże użytkownik musi być uprawniony do wykonania przynajmniej jednego polecenia w tym systemie”. Błąd polegał na operowaniu na globalnych zmiennych, które były dostępne dla wielu wątków. Polecenie do wykonania, zapisane w globalnej tablicy, było sprawdzane przez menedżera bezpieczeństwa, a następnie wykonywane. Programiści nie przewidzieli jednak, że w środowisku współbieżnym istnieje przeplot, w którym „złośliwy” wątek może zmienić polecenie między instrukcjami sprawdzenia i wykonania.

Tak wygląda błędny kod, który został nieznacznie skrócony w celu uwydatnienia błędu

```
public Process exec(String [] arstringCommand,
                    String [] arstringEnvironment)
                    throws IOException
  {
    //Upewnij się, że argumenty funkcji nie są nullami i
    //ich elementy nie są nullami, itd.

    // Wykonaj jakieś mało ważne instrukcje
```

```
    // Pobierz Menadżera Bezpieczeństwa

    SecurityManager securitymanager = System.getSecurityManager();

    // Sprawdź pierwszy element tablicy, który zawiera nazwę
    // polecenia do wykonania
    // Upewnij się, że posiadamy uprawnienia do wykonania tego
    // polecenia

    if (securitymanager != null) {
       securitymanager.checkExec(arstringCommand[0]);

    // Skoro już jesteśmy pewni, że możemy wykonać
    //polecenie, wykonujemy je
    //W tym miejscu "złośliwy" wątek może zmienić
    //polecenie na dowolne inne

    return execInternal(arstringCommand, arstringEnvironment);
  }
}
```

Niedostateczne sprawdzenie danych wejściowych

Nie wszystkie dane wejściowe programu pochodzą z zaufanych źródeł i program powinien w każdej sytuacji sprawdzać, czy dane mają właściwy format i czy są poprawne.

Nieskuteczny algorytm generowania liczb losowych

Obecnie wiele programów polega na algorymach kryptograficznych w celu zabezpieczenia swoich danych. Jeśli stosowane są proste algorytmy szyfrowania są one bardzo łatwe do złamania. Wiele wczesnych wersji przeglądarki Netscape Navigator miało łatwe do złamania zabezpieczenia z powodu wykorzystania zbyt krótkiego klucza szyfrującego. Powodem błędu był nieodpowiedni algorytm szyfrujący

1.2.2 Błędy związane z projektowaniem

Błędy projektowania są jeszcze bardziej poważne niż błędy implementacyjne. Mogą wynikać z krótkowzroczności analityków systemowych lub niezrozumienia języka albo cech stosowanych bibliotek, kompilatorów itp. Usunięcie takich błędów jest bardzo kosztowne, gdyż często ich konsekwencje są na tyle duże, że zmiana wiąże się z przebudową całego programu.

Niewłaściwe korzystanie z mechanizmów języków obiektowych

W językach obiektowych (tu na przykładzie Javy, która postrzegana jest jako bardzo bezpieczny język w porównaniu z C++) możliwe jest ominięcie mechanizmów związanych ze sprawdzaniem bezpieczeństwa klas przez zastosowanie deserializacji oraz klonowania obiektów [Javaworld]. Stosowany w Javie mechanizm bezpieczeństwa zakłada sprawdzenie przez menedżera bezpieczeństwa każdego nowotworzonego obiektu. Serializacja jest mechanizmem, który zamienia obiekt na ciąg bajtów, deserializacja jest mechanizmem odwrotnym. Znając algorytmy stosowane przez menedżera bezpieczeństwa, można dokonać takich zmian w obiekcie poddanym serializacji, że po deserializacji powstanie obiekt, który jest akceptowany przez menedżera bezpieczeństwa, jednakże może wykonać niebezpieczny kod.

Nieprawidłowe korzystanie z mechanizmów dziedziczenia (np. umożliwienie przeciążania metod w podklasach) i enkapsulacji (atrybuty publiczne i prywatne klas) jest również potencjalnym źródłem błędów. Są to błędy dość subtelne, ale mogą być niebezpieczne np. w przypadku gdy klasa udostępnia atrybuty, które powinny być prywatne.

Wbudowywanie w program informacji poufnych

Wydaje się być oczywiste, że w program nie powinny być statycznie wkompilowane żadne poufne informacje takie jak hasła dostępu do usług czy nazwy użytkowników. Jednak i takie błędy się zdarzają. Przykładem jest produkt InterBase firmy Borland, w którym znaleziono ten błąd w 2001 roku. Program używał na stałe wbudowanych identyfikatorów i haseł, aby uzyskać dostęp do bazy danych haseł w procesie uwierzytelniania użytkownika. Błąd został popełniony w 1994 roku i pozostawał przez 7 lat niezauważony!

1.3 Omówienie przepełnienia bufora

Błąd ten omówię na przykładzie języka C, ale może się on zdarzyć się w każdym języku, który pozwala na zapis w pamięci, do której program nie powienien mieć normalnie dostępu.

Istnieje wiele wariantów tego błędu, ale generalnie polega on na skopiowaniu danych z jednego obiektu do innego, bez sprawdzenia czy obiekt nadpisywany jest wystarczająco duży, żeby zmieścić wszystkie kopiowane dane. Do kopiowania używana jest funkcja taka jak sprintf, która nie kontroluje ilości przepisywanych danych względem wielkości miejsca przeznaczenia.

Przykład programu, który zawiera ten błąd:

```
void main (int
argc, char *argv[]){  p(argv[1]);
}

void p (char *str){
char bufor[16];

strcpy(bufor,str);
}
```

Błąd ten można wykorzystać, kiedy błędny program wykonuje się w atakowanym systemie. Program można wykorzystać do uruchomienia dowolnego kodu z uprawnieniami użytkownika, który ten program uruchomił. Jeśli jest to administrator, atakujący uzyskuje pełną kontrolę nad systemem.

Wykorzystywany jest mechanizm zapisywania argumentów funkcji i adresu powrotnego na stosie w momencie wywołania funkcji. Dokładniej, w momencie wołania funkcji na stosie zapisywane są: argumenty funkcji, adres powrotu, wskaźnik ramki stosu (SFP) i miejsce na zmienne lokalne funkcji.

Na przykład w momencie wywołania funkcji:

```
void function(char *str) {    char bufor[16];
   strcpy(bufor,str);
}
```

Stos będzie wyglądał następująco:

Łatwo zauważyć, że przepełnienie bufora danymi zamaże adres powrotny funkcji (ret). Atakujący może to wykorzystać wstawiając tam dowolny adres w pamięci. Pod tym adresem zostanie zamieszczony wcześniej spreparowany kod maszynowy, który np. uruchamia proces shella.

Błąd ten pojawia się bardzo często i mimo faktu, że jest powszechnie znany i wiele jego wariantów zostało dogłębnie przeanalizowane. W zeszłym roku w produktach firmy Microsoft zostało zgłoszonych oficjalnie 28 błędów typu BO, z czego ponad połowa o najwyższej randze (Critical). Najświeższy z nich z 18 grudnia pozwalał użytkownikowi dowolnej wersji Windows XP uruchomić shella z uprawnieniami administratora. [Microsoft] Tylko w grudniu na liście Bugtraq pojawiło się 18 zgłoszeń takich błędów.

1.4 Literatura

1. [Javaworld] www.javaworld.com/javaworld/jw-07-2001/jw-0713-howto.html

2. [Microsoft] Microsoft Security Bulletin MS02-072 www.microsoft.com/technet/treeview/?url=/techn
072.asp

3. [Bugtraq] online.securityfocus.com/archive/1

4. [C Guidelines] ouah.sysdoor.net/mixtercguide.html

5. Dokladny opis wykorzystania przepełnienia bufora (angielski) skywalker.mis.boun.edu.tr/ murat/bof-eng.txt

6. Organizacja CERT zajmująca się bezpieczeństwem systemów komputerowych - www.cert.org

Rozdział 2

Robaki

2.1 Wstęp

Druga część referatu poświęcona jest robakom komputerowym. Zawiera omówienie definicji robaka, przedstawia historię pierwszych robaków, prezentuje ich sposoby działania oraz charakteryzuje kilka współczesnych robaków.

2.2 Definicja

Nazwa robak pochodzi od angielskiego terminu określającego tasiemca (tapeworm). Robak jest to program, który umie replikować działające kopie samego siebie wykorzystując do rozprzestrzenia się sieć komputerową. Program ten dokonuje włamań do innych systemów i uruchamia się na tych systemach. Może powodować na nich różne szkody takie jak usunięcie plików czy wykorzystanie całych zasobów komputera, co prowadzi do awarii systemu.

2.3 Historia

Koncepcja robaka została opisana po raz pierwszy w 1975 r. przez Johna Brunnera w powieści SF „The Shockwave Rider”. W latach 1979-1981 badacze z firmy XEROX stworzyli eksperymentalne programy, które nazwali robakami. Ich robaki zostały zaprojektowane, aby „wędrować” po sieci i wykonywać użyteczne zadania w środowisku rozproszonym. Nie były to więc programy destruktywne, jak współczesne robaki.

2.4 Historia pierwszego robaka

Robak nazwany "The internet worm"(zwany również Morris Worm od nazwiska autora) zaatakował w 1988 r. Są różne opinie na temat tego w jaki sposób robak wydostał się poza system, na którym został napisany. Mógł on zostać wypuszczony celowo lub przypadkowo wydostać się z systemu w czasie testów. Robak ten w krótkim czasie doprowadził do blokady ówczesnego internetu, uruchamiając się na wielu serwerach i zużywając wszystkie ich zasoby przez uruchamianie kolejnych procesów shella.

Robak składał się z dwóch części. Jedna z nich była odpowiedzialna za włamywanie się do kolejnych systemów, a druga za prowadzenie destruktywnej działalności na tych systemach. Co ciekawe kod części robaka, która dokonywała włamania miał 99 linijek w języku C.

Robak włamywał się do systemów używając:

- błędu przepełnienia bufora w demonie fingerd.
- włączonej opcji DEBUG w programie sendmail, która pozwalała na uruchamianie dowolnych komend, jeśli do programu sendmail podane zostały odpowiednie parametry wywołania. Opcja ta była włączona na wielu serwerach, ponieważ wielu administratorów uznało ją za bardzo użyteczną.
- algorytmów słownikowych do łamania haseł użytkowników

Po włamaniu się do systemu robak atakował wszystkie serwery, o których system przechowywał informacje w plikach konfiguracyjnych. Na zaatakowanym systemie uruchamiał wiele procesów shell. Poces robaka cały czas zmieniał swój identyfikator przez co trudno było go znaleźć i zabić.

Działalność tego robaka została dokładnie zbadana i opiana. Więcej informacji na jego temat można znaleźć w [PSO] oraz [NetSec].

2.5 Współczesne robaki i ich sposoby działania

Obecnie istnieje ogromna ilość robaków. Większość z nich iększość atakuje systemy operacyjne firmy Microsoft, ale istnieją też robaki przeznaczone dla Linuxa.

2.5.1 Robaki dla Linuxa

Robak Ramen

Robak ten atakowal na przelomie 2000/01 systemy RedHat w wersji 6.2 i 7.0. Istnialy co najmniej dwie wersje tego robaka. Wersja „A” zastępowała w systemie wszystkie strony internetowe na spreparowaną stronę. Wersja „B” instalowała w systemie „tylne drzwi” oraz program umożliwiający dokonywanie z tego serwera ataków DOS (Denial of Service).

Dodaktkowo Ramen wyłączał usługi firewallowe, kasował pliki konfiguracyjne takie jak /etc/hosts.deny, co czyniło zainfekowany system łatwym do zaatakowania przez inne programy.

Robak wykorzystywał trzy błędy: w demonach FTPd, RPC.statd i LPRing. Dwa z nich były błędami przepełnienia bufora.

Co ciekawe, Ramen na zainfekowanym systemie instalował mini-serwer HTTP, który po nawiązaniu połączenia wysyłał kopię źródeł robaka. Można więc stwierdzić, że robak ten sam przygotowywał dla siebie środowisko do atakowania kolejnych serwerów.

2.5.2 Robaki atakujące systemy Microsoft Windows

Worm.Kurnikova i „I love you”

Robaki te wykorzystują ścisłą integrację komponentów Windows i Office oraz ogromną ilość błędów luk bezpieczeństwa w programie Outlook Express i systemie Active X.

Mechanizm działania robaka polega na przesłaniu użytkownikowi skryptu lub programu typu exe w załączniku listu. Plik może mieć podwójne rozszerzenie np. AnnaKournikova.jpg.vbs. Rozszerzenie .vbs jest domyślnie ukrywane, więc plik udaje zwykły obrazek. Dodatkowo list z robakiem przychodzi zwykle od znajomej osoby, co zmniejsza czujność użytkownika. Po otwarciu go, uruchamiany jest skrypt, który instaluje program robaka w systemie i wysyła robaka pod wszystkie adresy z książki adresowej.

Powstał nawet program Alamar’s Vbs Worms Creator, który pozwala na automatyczne tworzenie nowych wersji robaków typu „I love you”.

Gnutella-Worm.Mandragore

Atakuje systemy Windows. W celu rozprzestrzeniania się robak wykorzystuje system wymiany plików Gnutella. W zainfekowanym systemie rejestruje się jako węzeł sieci Gnutella, oczekuje na żądania pobierania plików i odpowiada na nie pozytywnie. Szkodnik oferuje plik o nazwie wyszukiwanej przez użytkownika systemu Gnutella, z rozszerzeniem EXE. Jeśli zdalny użytkownik pobierze ten plik, otrzyma kopię robaka.

2.6 Literatura

1. [NetSec] netsecurity.about.com/gi/dynamic/offsite.htm?site=http://world.std.com/ franl/worm.html
2. [PSO] Silberschatz, Galvin „Podstawy systemów operacyjnych” wyd. 3 rozdz. 20.5.1
3. The future of inernet worms - www.crimelabs.net/
4. The internet worm - netsecurity.about.com/gi/dynamic/offsite.htm?site=http://world.std.com/ franl/worm.html%23f1