Wstęp

Tematyka bezpieczeństwa w systemach i sieciach komputerowych jest obszerna i bardzo dynamicznie zmienna. Dynamika rozwoju sieci komputerowych stosunkowo najbardziej przypomina Wielki Wybuch, natomiast u podstaw tego rozwoju leżą koncepcje z lat 1970-1980-tych, kiedy powstawały elementarne technologie współczesnego Internetu, i kiedy nikt nie był w stanie przewidzieć ani stopnia rozwoju i zasięgu tych sieci, ani ich znaczenia dla funkcjonowania świata.

Dlatego w krótkim czasie i małej przestrzeni nie ma możliwości dokładnej analizy nawet najważniejszych zjawisk i metod związanych z tą tematyką. W dodatku, nie jest ona zbyt dobrze usystematyzowana. Wszelkie podziały na rodzaje zagrożeń, techniki ataków, i środki zabezpieczeń, są umowne, często nachodzą się i krzyżują.

W podręcznikach o systemach operacyjnych rozdział(y) dotyczące bezpieczeństwa zmieniają się najsilniej pomiędzy kolejnymi wydaniami, najszybciej się dezaktualizują, i ta tematyka jest powodem, dla którego potrzebne są stale nowe ich wydania.

Niemniej, bez powierzchownego przynajmniej poruszenia tej tematyki, wykład z systemów operacyjnych nie jest kompletny.

Zagrożenia

Zagrożenia bezpieczeństwa pracy systemów:

naruszenie poufności — pozyskanie poufnych danych np. w celu dokonania kradzieży, kompromitacji, itp.

uszkodzenie danych — wykasowanie plików, bądź ich modyfikacja

zakłócenie pracy systemów — np. doprowadzenia do zawieszenia się serwera, podmiana strony internetowej, itp.

kradzież zasobów lub usług — nieautoryzowany dostęp do usługi

zablokowanie świadczenia usługi — *Denial of Service* (DoS)

Niektóre zagrożenia należą do więcej niż jednej kategorii, bądź występują łącznie. Można również określić jeszcze inne zagrożenia.

Terminologia włamywaczy komputerowych

Określenia używane w języku potocznym często rozmija się z terminologią fachową.

Na przykład: określenie **haker** (*hacker*) w informatyce oznacza zdolnego programistę, członka zespołu programistycznego. Prasa popularna zawłaszczyła to określenie i nadała mu znaczenie włamywacza komputerowego. W odpowiedzi wprowadzono nowe określenie **kraker** (*cracker*) na określenie fachowego włamywacza komputerowego.

Krakerów odróżnia się od **skrypciarzy** (*script-kiddies*), którzy są włamywaczami-amatorami, często znudzonymi młodzieżowcami, posiadającymi komputer i wolny czas, oraz dostęp do internetu i narzędzi o dużych możliwościach.

Fachowcy od bezpieczeństwa komputerowego z konieczności zawodowo zajmują się wykrywaniem dziur w zabezpieczeniach, włamań, oraz innych nadużyć, jak również tworzeniem narzędzi zarówno do wykrywania jak i łatania dziur w zabezpieczeniach — całkiem podobnie do zwykłych włamywaczy. Z tego powodu określa się ich w tej roli jako **białe kapelusze** (*white hats*), w odróżnieniu od **czarnych kapeluszy** (*black hats*), którzy robią często podobne rzeczy, ale w złych zamiarach.

Istnieje cały asortyment narzędzi programowych do monitorowania i testowania zabezpieczeń systemów komputerowych, które posiadają **podwójne wykorzystanie**, tzn. mogą być skutecznym narzędziem w rękach obu rodzajów fachowców od bezpieczeństwa komputerowego: zarówno białych jak i czarnych kapeluszy.

Dygresja — bezpieczne systemy

Pojawia się pytanie, czy nie jest możliwe całkowicie szczelne i skuteczne zabezpieczenie systemów komputerowych, tak aby skuteczne ataki na nie nie były w ogóle możliwe?

Wbrew pozorom nie jest to całkiem bezsensowne pytanie, i bezpieczne systemy istnieją.

Jak to jest możliwe? Jednymi z głównych zasad budowania systemów niezawodnych i wbudowanych są: prostota i minimalizm. Jeśli system komputerowy jest prosty, to kompetentni fachowcy potrafią zbudować go tak by był bezpieczny. Jeśli system przekracza pewien poziom złożoności, to nie da się zidentyfikować i wyeliminować wszystkich słabych elementów, i aby uczynić go bezpiecznym, trzeba dodatkowo odizolować go od zagrożeń.

Niestety, dzisiejszy świat nie toleruje minimalizmu, ani ograniczeń w dostępie. Jeśli nawet jakiś system został zaprojektowany jako prosty i niewielki, to jeśli jest przydatny, z czasem będzie rozbudowywany o nowe funkcje. Niekiedy nowe funkcje mają sens, bo powstają lepsze albo innowacyjne rozwiązania. Jednak w większości są nikomu niepotrzebnymi bajerami, które narzuca przemysł i marketing. Ale w projektowanych w szalonym tempie i coraz bardziej złożonych systemach nie da się zachować prostej i logicznej struktury, niezbędnej do zapewnienia niezawodności i bezpieczeństwa.

Podobnie, wymaganie by każdy system był dostępny na wszelkie możliwe sposoby, z wielu urządzeń i przy użyciu różnych technologii, wyklucza bezpieczeństwo.

Źródła zagrożeń

Katalog zagrożeń można uporządkować według ich źródeł:

- błędy w programach, błędy użytkowników
- złośliwe oprogramowanie (*malware*): konie trojańskie, wirusy, robaki, rootkity
- ataki lokalne
- ataki sieciowe

Wykorzystanie błędów w programach

- ataki z wykorzystaniem przepełnienia bufora — najpopularniejsze
- wykorzystanie łańcuchów formatujących
- wykorzystanie „wiszących” lub pustych wskaźników
- wykorzystanie przepełnienia arytmetyki liczb całkowitych
- wstrzykiwanie kodu
- wykorzystanie wyścigów

Ataki z wykorzystaniem przepełnienia bufora

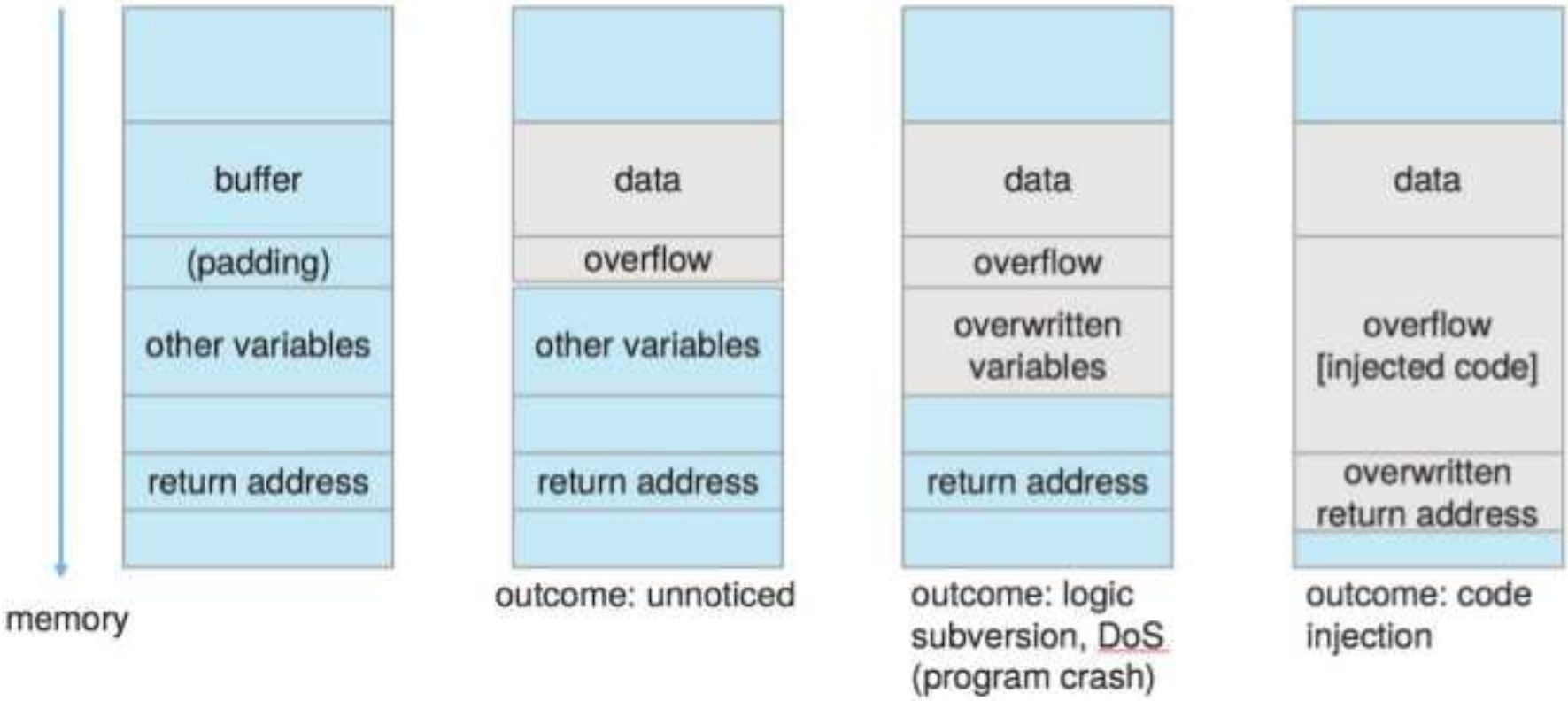
```
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
  int j = 4095;
  char buf1[16];
  char buf2[2];
  char buf3[16];
  int k = 4095;
  if (argc < 2) {return -1;}

  strcpy(buf1,"Ala ma kotaXXXX");
  strcpy(buf2,argv[1]);
  strcpy(buf3,"Jola ma kanarka");
  printf("J=%d, buf1=%s, buf2=%s, buf3=%s, K=%d.\n",
         j,buf1,buf2,buf3,k);
  return 0;
}
```

Ataki z wykorzystaniem przepełnienia bufora (2)

Skutkiem ataku na przepełnienie bufora może być napisanie zmiennych lokalnych procedury, i w efekcie błędy w działaniu programu. W przypadku napisania adresu powrotu z procedury na stosie, program może być przekierowany na określoną lokalizację w programie, na stosie, stercie, lub nawet w obrębie nadpisanego obszaru.



Złośliwe oprogramowanie — konie trojańskie

Konie trojańskie: programy, które na pozór wykonują bezpieczne i pożyteczne funkcje, jednak w rzeczywistości wykonują jakiś rodzaj ataku na system komputerowy.

Koń trojański może mieć postać ekranu logowania użytkownika — na stronie internetowej, ekranie komputera, terminalu komputerowym — albo wygaszacza ekranu, który żąda hasła użytkownika do odblokowania jakichś funkcji. Po odczytaniu i zarejestrowaniu danych uwierzytelniających użytkownika, program „znika", ustępując rzeczywistemu programowi logowania. Użytkownik często ma wrażenie, że musiał(a) wpisać błędnie hasło, bo ten sam program drugi raz o nie prosi.

Innym mechanizmem podkładania użytkownikom konia trojańskiego może być pokusa zainstalowania darmowej apki, atrakcyjnej wtyczki, albo paska narzędziowego. Jeszcze innym mogą być programy przesyłane w załącznikach mailowych, zainstalowane na odwiedzanych stronach internetowych (ściągane programy binarne lub kontrolki ActiveX), makra w dokumentach (Visual Basic), programy podłożone w omyłkowo otwartych katalogach w systemie (np. program `ls` gdy ścieżka `PATH` użytkownika jest `.:/bin:/sbin:...`, albo program `la` w którymś z niepilnowanych katalogów systemowych w ścieżce), i wiele innych.

Złośliwe oprogramowanie — wirusy

Metoda konia trojańskiego jest ogólnym określeniem całej grupy metod, za pomocą których złośliwe oprogramowanie może sforsować zabezpieczenia systemu i przedostać się do innego systemu. Jednak ze względu na sposób w jaki dane złośliwe oprogramowanie funkcjonuje w systemie komputerowym, wyróżnia się: wirusy, robaki, i rootkity.

Wirusem nazywa się fragment kodu przyklejający się do jakiegoś programu wykonywalnego. Podobnie jak jego biologiczny odpowiednik, wirus komputerowy nie jest w stanie żyć sam.

Można wyróżnić kilka rodzajów wirusów komputerowych, w zależności od środowiska, w jakim wirus się zagnieżdża:

- pliki programów wykonywalnych
- boot sektory dysków
- makrodefinicje (makra)

Złośliwe oprogramowanie — robaki

Robaki to samodzielne programy rozprzestrzeniające się metodami konia trojańskiego, lub przez bezpośrednie łączenie się przez sieć, wykorzystując znane (lub niedawno odkryte) błędy i luki w oprogramowaniu sieciowym systemów operacyjnych.

Złośliwe oprogramowanie — rootkity

Rootkit to złośliwe oprogramowanie instalujące się głęboko w systemie operacyjnym, w oprogramowaniu firmware komputera, w boot sektorze dysku, w jądrze, w sterownikach urządzeń, w bibliotekach, a nawet w systemie operacyjnym gościa na maszynie wirtualnej, itp. Dzięki pełnemu dostępowi do systemu rootkity często maskują się modyfikując wywołania systemowe które pozwalałyby je wykryć; jedną z metod rozpowszechniania rootkitów jest umieszczanie go w skryptach *autorun* na płytach CD/DVD (rootkit firmy Sony 2005).

Ataki lokalne

- wykorzystanie błędów/luk w zabezpieczeniach, np. odczytanie lub zniszczenie pliku, który nie został poprawnie zabezpieczony
- ataki „sfrustrowanych" administratorów i/lub programistów (bomby logiczne, tylne drzwi)
- atak na niezabezpieczone lub słabo zabezpieczone konta, łamanie haseł
- wyłudzanie dostępu lub hasła przez fałszowanie ekranu logowania
- socjotechnika — wykorzystanie psychologii w celu uzyskania dostępu do chronionego obiektu lub systemu przez manipulację osobami stojącymi na straży takich obiektów lub systemów

Ataki sieciowe

- podsłuchiwanie, skanowanie portów
- podszywanie się (*masquerading*, *spoofing*)
- atak pośrednika (*man-in-the-middle*)
- DoS, DDoS
- ...

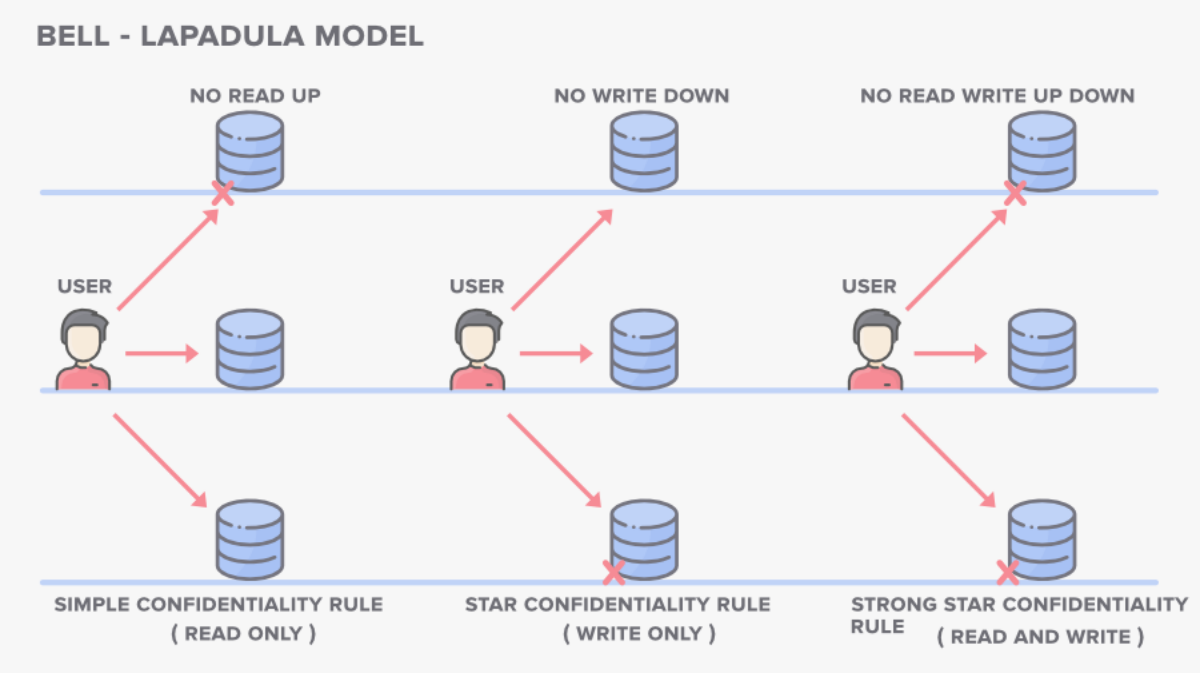
Bezpieczeństwo systemów komputerowych

Ogólnie, nie ma 100%-owego bezpieczeństwa w systemach komputerowych, a tworzenie wszelkich zabezpieczeń jest trudne i kosztowne. Wynika to z faktu, że zabezpieczeniami muszą być objęte wszystkie elementy i mechanizmy systemów i oprogramowania. Poza koniecznym dużym kosztem pracy programistów, powoduje to wielokrotne zwiększanie całego systemu, i w efekcie zwiększenie zawodności i powstawanie dalszych luk w zabezpieczeniach. Dodatkowo, nakłada to wymagania na użytkowników, którzy zmuszeni są do poddania się trudniejszym, bardziej uciążliwym procedurom systemowym. Często ci użytkownicy, ze względu na lenistwo lub swoje przyzwyczajenia, próbują dróg „na skróty" i sami obchodzą zabezpieczenia systemowe, tworząc kolejne luki.

Dla odmiany, atakujący ma łatwiej, wystarczy jedna mała luka w ogólnie dobrze zabezpieczonym systemie, i można ją wykorzystać do uruchomienia wielotorowych ataków na taki system. Ten sam atak można powtórzyć wielokrotnie w krótkim czasie, korzystając z dużej wydajności komputerów i przepustowości sieci. Opracowany skrypt ataku można rozpowszechniać w sieci, dając do ręki wielu użytkownikom narzędzia do nękania poprawnie pracujących, ale być może nie w pełni zabezpieczonych systemów (*script-kiddies*).

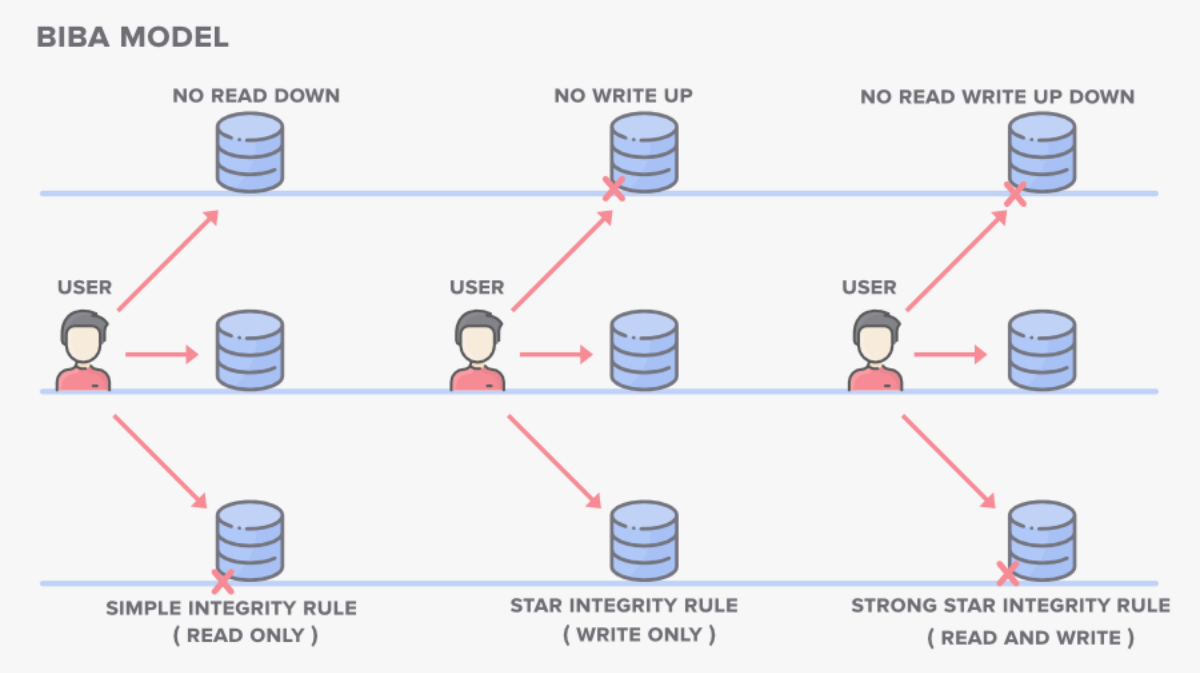
Ogólne modele systemów bezpieczeństwa

Powstało wiele modeli bezpieczeństwa, o charakterze teoretycznym i praktycznym. Znaczącym przykładem jest **model Bell-LaPaduli**, osadzony w wojskowości, gdzie istnieje hierarchia klas bezpieczeństwa dokumentów: Unclassified, Confidential, Secret, i Top Secret. Jednocześnie istnieje analogiczna hierarchia uprawnień użytkowników.



Użytkownik z uprawnieniami Secret ma dostęp „read” do dokumentów Secret i niższych, ale nie ma do nich dostępu „write”. Jednocześnie posiada dostęp „write” do dokumentów Secret i Top Secret, aczkolwiek do tych ostatnich nie ma dostępu „read”. Inaczej mówiąc, użytkownik może tworzyć dokumenty o klasie bezpieczeństwa wyższej od swoich uprawnień, ale nic co stworzy nie może mieć niższej klasy bezpieczeństwa.

Model Bell-LaPaduli dobrze zabezpiecza poufność dokumentów, lecz nie chroni integralności, ponieważ pozwala użytkownikom o niższej klasie uprawnień modyfikować dokumenty powyżej swojej klasy. Dlatego formułuje się analogiczny — ale dualny — **model Biby**, który interpretuje uprawnienia w sposób dokładnie odwrotny.



Modele LaPaduli i Biby są abstrakcjami, i często nie odpowiadają wymogom świata rzeczywistego. Na przykład, użytkownik może potrzebować w swojej pracy dostępu do pewnego fragmentu dokumentu, do którego nie ma pełnego dostępu. Poza tym, klasyfikacja dokumentów może ulegać zmianom, i ktoś musi te zmiany wprowadzać. Dodatkowo, odchodząc od wojskowości do świata biznesu i komercji, właściciele dokumentów sklasyfikowanych na danym poziomie bezpieczeństwa mogą nie chcieć, aby mieli do nich dostęp inni użytkownicy na tym samym poziomie uprawnień.

Zabezpieczenia

Aby przeciwdziałać zagrożeniom, zabezpieczenia mogą być budowane na wielu poziomach:

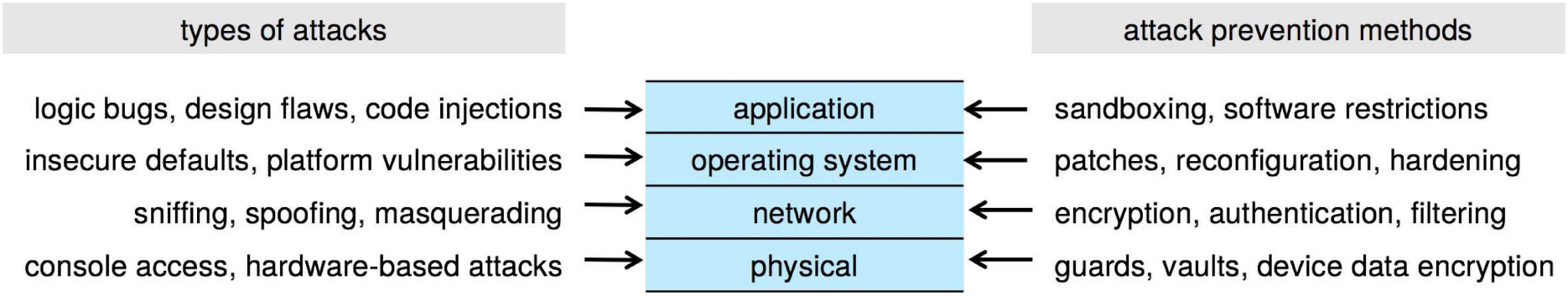
fizycznym — zabezpieczenia sprzętowe, ograniczenia dostępu

sieciowym — urządzenia z dostępem do sieci komputerowych muszą mieć zabezpieczenia zarówno dostępu jak i komunikacji

systemu operacyjnego — usługi systemu operacyjnego a definicji oferują pewien zbiór mechanizmów zabezpieczających

aplikacji — systemy oprogramowania aplikacyjnego mogą mieć zarówno defekty programistyczne, jak i błędy koncepcyjne, logiczne

języka programowania — mechanizmy bezpiecznego programowania wbudowane w język programowania



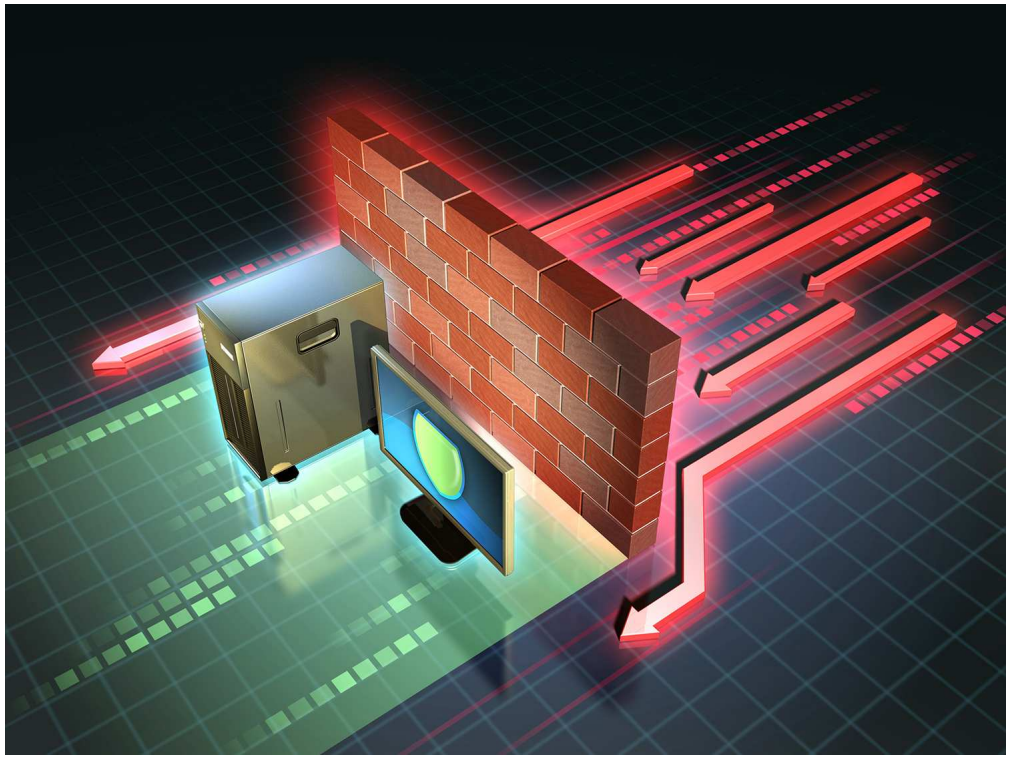
Zabezpieczenia sieciowe

- zapory sieciowe (*firewall*)
- systemy wykrywania włamań (*Intrusion Detection Systems* IDS) — próba rozpoznania zachowania użytkowników lub aplikacji jako włamania
- logowanie i monitorowanie

Zapory sieciowe

Zapora sieciowa (*firewall* — mur ogniowy) jest narzędziem cyberbezpieczeństwa, którego zadaniem jest ochrona komputera lub sieci lokalnej przed zagrożeniami przychodzącymi z sieci, przy zachowaniu komunikacji z tą siecią.

Zapory sieciowe zawierają programowalny zestaw reguł, określających czy dany wychodzący lub przychodzący pakiet sieciowy należy przepuścić, czy zablokować.



Klasyfikacja zapór sieciowych ze względu na sposób działania:

- zapory filtrujące
- zapory pośredniczące
- inne mechanizmy (pseudo-zapory sieciowe)

Filtrujące zapory sieciowe

Filtrujące zapory sieciowe monitorują przepływające przez nie pakiety sieciowe i przepuszczają tylko zgodne z regułami ustawionymi na danej zaporze (Wikipedia).

Zapory tego typu różnią się jednak co do głębokości analizy, jakim poddawane są poszczególne pakiety sieciowe, i nakładu pracy na analizę jego roli i ocenę rzetelności. Starsze zapory sprawdzają tylko podstawowe dane z nagłówka warstwy sieciowej (IP): adresy IP nadawcy i odbiorcy pakietu, numery portów, typ pakietu. Taka analiza jest prosta, i decyzja może być podjęta szybko, ale nie jest w stanie wykryć wielu schematów włamań, bo decyzja o uznaniu pakietu za prawidłowy lub nieprawidłowy podejmowana jest na podstawie bardzo cząstkowej informacji.

Pewien wariant filtracji pakietów sprawdza elementy negocjacji połączeń TCP, co wymaga sprawdzenia nagłówka TCP, wewnątrz pakietu sieciowego (IP).

Nowsze i bardziej zaawansowane zapory sieciowe głębiej analizują treść każdego pakietu, z analizą stanu połączeń przepływających przez zaporę (*stateful inspection*). Tego typu zapora jest w stanie śledzić stan istniejących połączeń i realizować o wiele bardziej rozbudowaną politykę bezpieczeństwa, oraz podejmować znacznie bardziej prawidłowe decyzje zgodne z tą polityką. Jednak z jednej strony konfiguracja takiej zapory jest trudniejsza, a jej praca wymaga większych nakładów obliczeniowych, i mimo wszystko może istotnie spowalniać ruch przepływający przez zaporę.

Pośredniczące zapory sieciowe

Pośredniczące zapory sieciowe nie ograniczają się do sprawdzania pakietów, ale aktywnie uczestniczą w komunikacji sieciowej pełniąc rolę serwerów *proxy*. Takie zapory, określane również jako **proxy poziomu aplikacji**, izolują aplikacje pracujące za zaporą od możliwych ataków ze strony sieci próbujących znaleźć dziury w zabezpieczeniach chronionych systemów. Te ataki przejmuje na siebie zapora, z założenia zbudowana na prostszym systemie i lepiej dopracowana pod względem bezpieczeństwa, niż dynamicznie zmieniane i stale aktualizowane systemy.

Zdolność analizowania treści pakietów sieciowych, również składania wielu z nich dla odtworzenia treści komunikacji, pozwala na przykład na analizę typu załączników zawartych w komunikatach e-mail, i odrzucenie e-maila, bądź wycięcie załącznika, który zgodnie z polityką bezpieczeństwa został uznany za niebezpieczny (np. zawiera wirusa).

W parze z większymi możliwościami idą jednak: bardziej skomplikowana konfiguracja, wymagająca stałego nadzoru, monitorowania, aktualizacji oprogramowania, a także bardziej intensywne obliczenia. Rozbudowana zapora sieciowa może mieć reguły zabezpieczeń obejmujące wiele różnych warstw oprogramowania sieciowego

Niektóre bardziej zaawansowane zapory tego typu są określane jako zapory **następnej generacji** (*next-generation*).

Pseudo-zapory sieciowe

Pewne systemy sieciowe niebędące zaporami sieciowymi pełnią funkcje, które do pewnego stopnia realizują również funkcje zapory sieciowej.

Router NAT (*Network Address Translation*) jest urządzeniem, lub programem, który pozwala komputerom w sieci lokalnej LAN wykorzystującej prywatny zakres adresów IP[1] komunikować się z Internetem. Router NAT sam posiada połączenie internetowe i pełni rolę *proxy* warstwy połączeniowej dla węzłów wewnątrz sieci LAN. Dzięki tej roli, dodatkowo ukrywającej rzeczywistą tożsamość tych węzłów, router NAT pełni część funkcji zapory sieciowej i skutecznie chroni sieć LAN przed wieloma zagrożeniami.

Wirtualna sieć prywatna VPN (*virtual private network*) jest tunelem pozwalającym łączyć ze sobą dwie sieci lokalne wirtualnym połączeniem przez sieć niezabezpieczoną (Internet), bez łączenia się z innymi systemami w Internecie. Ze względu na szyfrowanie połączenia i jego realizację typowo przez dobrze dopracowane oprogramowanie, lub urządzenia, wirtualne sieci prywatne pozwalają uniknąć komunikującym się partnerom zagrożeń pochodzących z sieci.

[1] Prywatne zakresy adresów IP zostały stworzone by umożliwić wykorzystywanie oprogramowania TCP/IP wewnątrz organizacji, które nie są podłączone do Internetu. Ruch sieciowy wykorzystujący takie adresy nie może wypływać poza sieć lokalną. Zakresy adresów prywatnych IPv4: 10.0.0.0–10.255.255.255, 172.16.0.0–172.31.255.255, 192.168.0.0–192.168.255.255.

Realizacja zapory sieciowej

Ze względu na miejsce/sposób realizacji, zapory sieciowe można podzielić na: (i) sprzętowe, (ii) programowe, i (iii) w chmurze.

Sprzętowa realizacja zapory ma sens zwłaszcza jeśli jakaś sieć lokalna wykorzystuje do połączenia z Internetem: (a) sprzętowy router sieciowy, (b) sprzętowy modem DSL/ADSL, (c) sprzętowy router NAT, lub (d) urządzenie, które łączy więcej niż jedną z tych funkcji. Urządzenie może dodatkowo realizować funkcje zapory sieciowej pozwalając skonfigurować pożądaną politykę bezpieczeństwa. Ale sprzętowe zapory istnieją również jako samodzielne urządzenia.

Zapora zaimplementowana programowo na komputerze może służyć do ochrony sieci lokalnej analogicznie do zapory sprzętowej. Dodatkowo, niektóre systemy operacyjne implementują indywidualną zaporę sieciową chroniącą sam pojedynczy komputer.

Zapora może być zrealizowana jako *proxy* zlokalizowane w chmurze. Ma to zwłaszcza sens jeśli infrastruktura danej organizacji jest zlokalizowana również w chmurze. Zapora nie chroni już wtedy fizycznie sieci przed wpuszczeniem do niej złośliwych pakietów, natomiast chroni wszystkie elementy infrastruktury firmowej pośrednicząc w komunikacji sieciowej tych elementów ze światem zewnętrznym.

Zapora sieciowa jądra Linuksa: iptables

Jądro Linuksa posiada wbudowaną, konfigurowalną warstwę filtrowania pakietów sieciowych. W nowych jądrach (≥ 2.4) za konfigurację odpowiedzialny jest program `iptables`.

System iptables zawiera **łańcuchy** reguł, które zebrane są w **tabele**. Można tworzyć nowe reguły, i dodawać je do konkretnych łańcuchów konkretnych tabel. Jak również modyfikować istniejące w nich reguły. Podstawowe tabele iptables:

filter
: To jest domyślna tabela, zawiera łańcuchy: INPUT, FORWARD, OUTPUT.

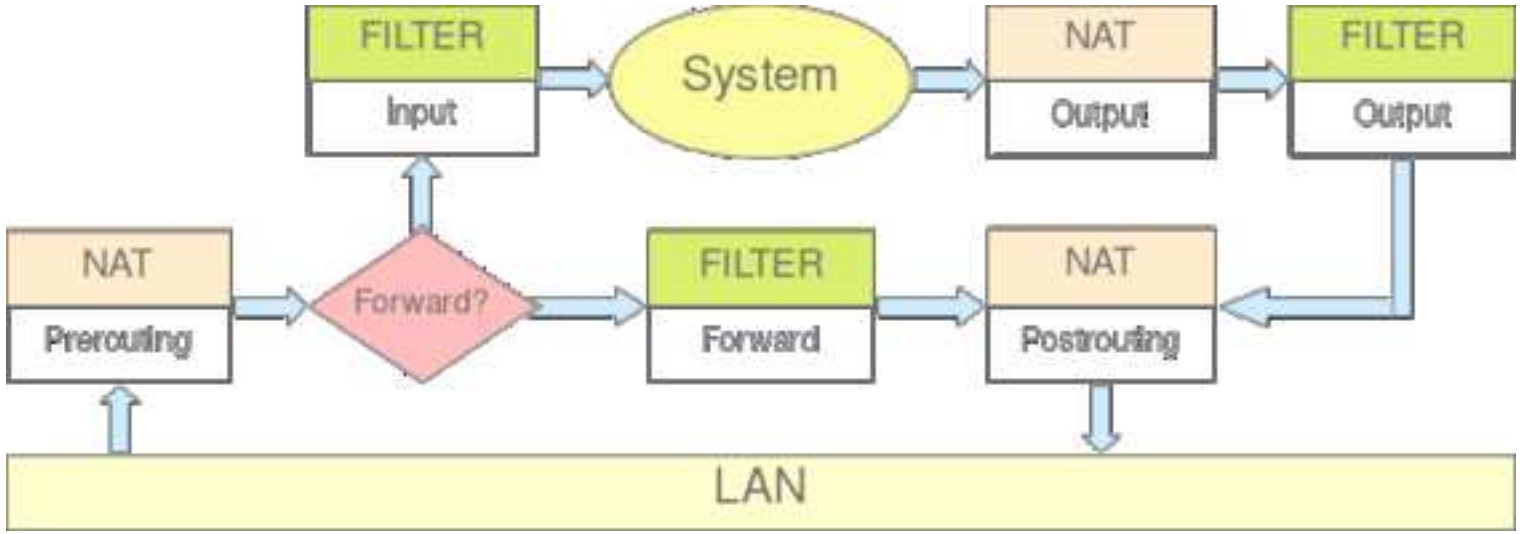
nat
: Trafiają do niej pakiety nawiązujące nowe połączenia, zawiera łańcuchy: PREROUTING, OUTPUT, POSTROUTING.

mangle
: Tabela dla wyspecjalizowanych konwersji pakietów. Zawiera łańcuchy: PREROUTING, INPUT, FORWARD, OUTPUT, POSTROUTING.

raw
: Tabela stosowana przed wszystkimi innymi, zawiera łańcuchy: PREROUTING i OUTPUT.

Przepływ pakietów przez iptables

Poniższy diagram ilustruje przepływu pakietów przez łańcuchy iptables. Dla uproszczenia pominięte w nim zostały tabele raw i mangle.



Każdy pakiet sieciowy jest przetwarzany po kolei przez wszystkie reguły każdego łańcucha, do momentu, w którym reguła „pasuje" do pakietu. Wtedy wykonywana jest akcja określona przez daną regułę. Gdy żadna reguła łańcucha nie pasuje do pakietu, wtedy wykonywana jest domyślna akcja łańcucha.

iptables: przykłady konfiguracji

Domyślnym przeznaczeniem dla wszystkich pakietów w tabeli filter w łańcuchach FORWARD, INPUT, OUTPUT jest ACCEPT. Załóżmy, że chcemy zezwolić na otwieranie dowolnych połączeń wychodzących (łańcuch OUTPUT), ale zabronić wszelkich pakietów przychodzących, jak również przepływu obcych pakietów:

```
iptables -P FORWARD DROP
iptables -P INPUT DROP
```

Mogłoby się wydawać, że to jest minimalna sensowna konfiguracja. W rzeczywistości jednak nie jest przydatna prawie do niczego, ponieważ nie wpuszcza odpowiedzi na połączenia wychodzące; system może wysyłać polecenia, ale nie przyjmie odpowiedzi.

Załóżmy, że jako minimalną konfigurację sieciową chcemy mieć zdolność *browsowania* Internetu, czyli otwierania połączeń HTTP do zdalnych serwerów na porcie 80. Aby wpuścić do systemu pakiety odpowiedzi dotyczące tych połączeń, można dodać reguły:

```
iptables -A INPUT --protocol tcp --source-port 80 -j ACCEPT
iptables -A INPUT --protocol udp --source-port 53 -j ACCEPT
```

Druga reguła umożliwia uzyskiwanie odpowiedzi z DNS w przypadku użycia adresów symbolicznych, które masowo pojawiają się w dokumentach HTTP (np. obrazki). Klient może komunikować się z serwerem DNS protokołem TCP lub UDP. Powyższa reguła umożliwia tylko komunikację UDP.

iptables: podsumowanie — konfiguracja minimalna

Powyższe reguły stanowią niezłą minimalną konfigurację komputera osobistego:

```
# ustawienie polityki restrykcyjnej
iptables -P FORWARD DROP
iptables -P INPUT DROP

# umozliwienie translacji adresow
iptables -A INPUT --protocol udp --source-port 53 -j ACCEPT

# umozliwienie pracy z ssh i HTTP
iptables -A INPUT --protocol tcp --source-port 80 -j ACCEPT
iptables -A INPUT --protocol tcp --source-port 22 -j ACCEPT

# przyjmowanie przychodzacych polaczen ssh
iptables -A INPUT --protocol tcp --destination-port 22 -j ACCEPT

# przyjmowanie klientow X Window
iptables -A INPUT --protocol tcp --destination-port 6000 -j ACCEPT

# pingowanie innych
iptables -A INPUT --protocol icmp --icmp-type 0 -j ACCEPT
# mozliwosc pingowania nas - w zaleznosci od preferencji
###iptables -A INPUT --protocol icmp --icmp-type 8 -j ACCEPT
```

Zabezpieczenia na poziomie systemu operacyjnego

Systemy operacyjne, w celu realizacji swoich funkcji, muszą wykorzystywać dwa tryby pracy: tryb jądra i tryb użytkownika, wspierane przez procesor. Program (fragment programu) wykonujący się w trybie jądra może wykonywać instrukcje uprzywilejowane, i posiada zdolność pełnego zarządzania systemem komputerowym. Proces wykonujący się w trybie użytkownika może wykonywać jedynie nieuprzywilejowane instrukcje.

Jednak musi również istnieć mechanizm, dzięki któremu proces użytkownika może wykonać operację uprzywilejowaną.

Dodatkowo, system musi zapewniać zabezpieczenie procesów przed sobą nawzajem.

Zasada minimalnego przywileju (*principle of least privilege*) — aplikacjom, użytkownikom, oraz całym systemom należy przyznawać minimalną ilość przywilejów niezbędną do pracy.

Ta zasada jest realizowana przez różne systemy operacyjne w różny sposób i w różnym stopniu.

Zabezpieczenia na poziomie systemu operacyjnego — przykłady

Przykład 1: Unix. Istnieją konta użytkowników, które tworzą przestrzenie ochrony. Wszystkie procesy jednego użytkownika mogą sobą nawzajem zarządzać (np. wysyłać sobie sygnały), ale nie mogą zarządzać procesami innych użytkowników. Użytkownik numer 0 (root) jest uprzywilejowany, może zarządzać wszystkimi procesami w systemie i ma pełny dostęp do wszystkich zasobów.

Aby umożliwić procesom użytkowników wykonywanie operacji zastrzeżonych, istnieje mechanizm zwany **set-uid**. Jest on sterowany specjalnym bitem w bloku kontrolnym każdego pliku. Jeśli ten bit jest ustawiony, to wykonanie danego programu powoduje, że cały proces wykonuje się z uprawnieniami użytkownika root.

Ten mechanizm, w teorii prosty i skuteczny, w rzeczywistości może zostać nadużyty, zarówno przez błędy w administracji systemu (z jakichś powodów pozwalające zwykłemu użytkownikowi na utworzenie, lub przejęcie kontroli nad plikiem set-uid), jak i przez omówione wcześniej mechanizmy przepełnienia bufora, wstrzykiwania kodu, lub wyścigów.

Zabezpieczenia na poziomie systemu operacyjnego — przykłady

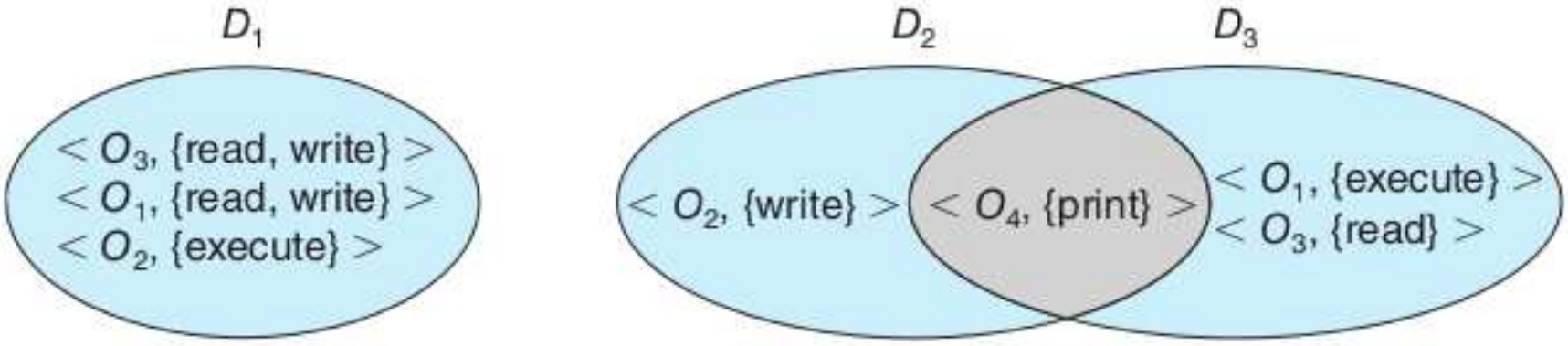
Przykład 2: Android. W tym systemie (opartym na Linuksie, który może być traktowany jako wariant Uniksa) różne aplikacje instalowane są z różnymi identyfikatorami użytkowników (UID) i grup (GID). Zatem aplikacje zainstalowane na danym urządzeniu korzystają ze zwykłych uniksowych zabezpieczeń użytkowników.

Dodatkowo, Android implementuje szczegółowe mechanizmy bezpieczeństwa. Na przykład, pewne operacje systemowe są dostępne tylko dla wybranych GID (np. gniazdka sieciowe dla grupy `aid_inet`, 3003). Pewne UID mogą być zdefiniowane jako izolowane, co ogranicza możliwości wywoływania przez nie usług RPC do minimum.

Dziedziny zabezpieczeń

Aby wprowadzić zunifikowany widok na zabezpieczenia istniejące w systemie operacyjnym można stosować pojęcie dziedzin zabezpieczeń. Dziedzina jest zbiorem par: obiekt-uprawnienia.

System komputerowy traktujemy jak zbiór procesów i obiektów. Obiektami są: obiekty sprzętowe (CPU, moduły pamięci, dyski, drukarki, itp.), oraz obiekty programowe (pliki, programy, blokady, semafory, itp.). Obiekty posiadają identyfikatory i są dostępne poprzez określony zbiór operacji, zależny od rodzaju obiektu. Np. procesor może jedynie wykonywać kod. Plik danych można tworzyć, zapisywać, odczytywać, albo kasować. Pliki wykonywalne można dodatkowo wykonywać.



Proces może wykonywać operacje na obiektach, do których posiada uprawnienia. Ponadto, zgodnie z zasadą minimalnego przywileju, powinien mieć prawo w danej chwili wykonywać tylko takie operacje, które są niezbędne w ramach wykonywanego zadania. Dziedzina, w której proces działa, określa te obiekty i operacje.

Tworzenie dziedzin zabezpieczeń

Związek procesu z dziedziną może być albo statyczny, ustalony na cały czas życia procesu, albo dynamiczny. W przypadku statycznym, zawartość dziedziny musi podlegać zmianom, aby cały czas odzwierciedlać zasadę minimalnego przywileju.

W szczególności, koncepcja dziedzin zabezpieczeń może być zrealizowana na następujących poziomach:

- Każdy użytkownik może być dziedziną.
- Każdy proces może być dziedziną.
- Każda procedura może być dziedziną.

Macierz dostępu

Zbiór uprawnień może być przedstawiony w postaci **macierzy dostępu**.

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Wiersze tej macierzy reprezentują dziedziny, kolumny — obiekty, a treścią macierzy są uprawnienia istniejące w danej dziedzinie dla danego obiektu.

Zauważmy, że domeny też zostały uwzględnione jako obiekty, na których można wykonywać operację `switch`. W powyższym przykładzie, proces wykonujący się w domenie D_1 może przejść do domeny D_2, a proces wykonujący się w domenie D_2 może przejść do domen D_3 oraz D_4.

Macierz dostępu (2)

System powinien zapewnić mechanizm tworzenia nowych praw dostępu. W tym celu może istnieć uprawnienie `owner`, które pozwala procesowi wykonującemu się w ramach uprawnionej domeny nadawać uprawnienia dla obiektu, dla którego posiada uprawnienie `owner`.

domain \ object	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

domain \ object	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

W powyższym przykładzie widać, że dzięki uprawnieniom `owner` istniejącym w macierzy po lewej stronie, zostały dodane dodatkowe uprawnienia w macierzy po prawej stronie.

(Uprawnienia z gwiazdkami odnoszą się do mechanizmu kopiowania uprawnień — te uprawnienia dla danego obiektu mogą być kopiowane do innych dziedzin, nawet bez uprawnienia `owner`.)

Macierz dostępu (3)

Poza możliwością modyfikacji uprawnień dla danego obiektu, niezbędna jest zdolność modyfikacji uprawnień konkretnej dziedziny. Pozwala na to inne uprawnienie `control`.

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

W powyższym przykładzie, proces działający w dziedzinie D_2 ma prawo modyfikować dowolne uprawnienia dla dziedziny D_4.

Implementacja macierzy dostępu

Macierze dostępu mogą być tworzone w systemie jako pełne tablice. Jest to rozwiązanie mało wygodne, ze względu na duże rozmiary i problemy z przechowywaniem ich w pamięci RAM.

Zamiast tego, poszczególne kolumny macierzy mogą być przechowywane jako **listy praw dostępu** (*Access Control Lists* ACL) dla obiektów. Lista pozwala pominąć puste pozycje macierzy. Dodatkowo, lista dla danego obiektu może zawierać domyślne prawa dostępu dla tego obiektu.

Alternatywnie, wiersze macierzy uprawnień mogą być przechowywane jako **listy uprawnień** (*Capability Lists*) dla dziedzin.

Możliwe są również inne rozwiązania. Większość nowszych systemów uniksowych stosuje ACL, jak również Windows. Standard POSIX określa system uprawnień, i Linuksy od jądra 2.6.24 posiadają system i listy uprawnień, które współistnieją z mechanizmem set-uid.

Uwierzytelnianie

Użytkownicy systemów komputerowych mają przyznane różne uprawnienia. Aby system udostępnił użytkownikowi dostęp do zasobów, do których ma uprawnienia, musi on/ona: (a) zidentyfikować się, to znaczy podać swoją tożsamość, i (b) uwierzytelnić się, to znaczy, potwierdzić tę tożsamość.

Problem identyfikacji i uwierzytelniania istnieje w wielu systemach, nie tylko komputerowych. Taka potrzeba zachodzi np. przy wypożyczaniu książki z biblioteki, wyłączania systemu alarmowego, albo wypłaty gotówki z bankomatu. Nie jest to również potrzeba związana ze współczesnymi czasami ani nowymi technologiami. Już w starożytności wysłannik jednego władcy, aby być jego uznanym reprezentantem w siedzibie innego, musiał się uwierzytelnić.

Tradycyjnie, istnieją trzy podstawowe sposoby uwierzytelniania:

- za pomocą czegoś, co użytkownik wie,
- za pomocą czegoś, co użytkownik posiada,
- za pomocą czegoś, co użytkownika charakteryzuje.

W praktyce te trzy sposoby z reguły przekładają się na użycie: haseł, tokenów (żetonów?) dostępu, i parametrów biometrycznych.

Uwierzytelnianie z użyciem haseł

Najstarszą, powszechnie stosowaną techniką uwierzytelniania użytkowników przy dostępie do zasobów komputerowych były i są hasła. Użycie haseł typowo jest dwustopniową procedurą odpowiadającą z grubsza krokom identyfikacji i uwierzytelnienia: login+hasło, albo numer karty+PIN, itp.

Niestety, idea uwierzytelniania oparta na hasłach kryje w sobie jedno istotne założenie, które okazuje się fałszywe. Ta idea polega na tym, żeby wymyślić hasło przypadkowe, trudne do zgadnięcia, i jednocześnie łatwe do zapamiętania. W praktyce okazuje się, że hasła łatwe do zapamiętania, typu Gosia123, są jednocześnie łatwe do zgadnięcia, natomiast trudne do zgadnięcia hasła typu !7Ug*4qN są też trudne do zapamiętania.

Powody, dla których hasła łatwe do zapamiętania okazują się również łatwe do zgadnięcia mogą nie być oczywiste, bo np. skąd atakujący może wiedzieć, że moja dziewczyna ma na imię Gosia? Bynajmniej nie dzieje się to na zasadzie drapania się w głowę i wymyślania kolejnych dobrych, łatwych do zapamiętania kandydatów. Łatwe do zapamiętania hasła są podatne na tzw. **ataki słownikowe**. Krakerzy budują listy wszystkich potencjalnych haseł złożonych ze wszystkich słów danego języka, nazw własnych: imion ludzi, popularnych nazwisk, nazw geograficznych, handlowych, itp. Plus ich wariantów z dodatkiem często używanych liczb (jak 1). Taki słownik wszystkich potencjalnych „łatwych" haseł zawiera setki tysięcy pozycji; jeśli kraker ma możliwość wypróbowania ich wszystkich przy użyciu komputera bardzo szybko, to ...

Środki obrony przed atakiem słownikowym

Niektóre systemy bronią się przed zgadywaniem haseł wprowadzając: (a) opóźnienia po każdej kolejnej nieudanej próbie podania hasła, (b) blokowanie konta na jakiś czas po kilku nieudanych próbach hasła, itp.

Jednak pozostają pewne problemy. W niektórych wczesnych systemach hasła przechowywane były w wersji źródłowej, co w oczywisty sposób tworzy zagrożenie w przypadku wycieku takiej bazy haseł. System Unix wprowadził metodę przechowywania haseł w formie zahaszowanej funkcją jednokierunkową, z jednoczesną zasadą dostępności wszystkich zahaszowanych haseł.

Niestety, ta metoda wymyślona i skuteczna w latach 1970-tych, przestała być bezpieczna w dobie masowych ataków i szybkich komputerów. Jawnie dostępne hasła mogą być poddane atakowi słownikowemu, który skutecznie pozwala znaleźć oryginalne hasło, o ile nie jest ono zbyt skomplikowane.

Zabezpieczono to już w połowie lat 1980-tych przez wprowadzenie w systemie SunOS bazy zahaszowanych haseł niedostępnych dla użytkownika (*shadow file*). W połączeniu z opóźnieniami i tymczasowym blokowaniem dostępu w przypadku podania nieprawidłowego hasła, zapewniło to skuteczną ochronę indywidualnych kont użytkowników.

Ataki słownikowe — dodatkowe zagrożenia

Opisane metody skutecznie chronią konta pojedynczych użytkowników w poprawnie zabezpieczonym systemie. Jednak sytuacja jest inna w przypadku wykradzenia całej bazy zahaszowanych haseł. Kraker może wtedy przeprowadzić atak słownikowy na taką bazę, co prowadzi do odkrycia haseł do wielu kont naraz. Te konta mogą być następnie wykorzystane do dalszych ataków. Dodatkowo, częsta praktyka używania jednego hasła w wielu systemach pozwala krakerowi na kontynuację ataku. (Krakerzy wykorzystują również tę ludzką słabość zakładając pozornie atrakcyjne strony webowe tylko po to, by skłaniać użytkowników do zakładania tam kont, i pozyskiwania ich ulubionych haseł!)

Co więcej, ponieważ w trakcie ataku słownikowego na całą bazę haseł, hasze wszystkich haseł ze słownika musiałyby być wielokrotnie obliczane, kraker może przygotować sobie słownik od razu w postaci zahaszowanej. Taka baza zahaszowanych haseł do wykorzystania w szybkim łamaniu haseł nazywana jest **tęczową tablicą** (*rainbow table*).

Atak z wykorzystaniem takiej tablicy jest bardzo efektywny, ponieważ polega jedynie na porównywaniu zahaszowanych haseł, bez konieczności obliczania dodatkowych haszy.

Środki obrony przed atakiem słownikowym — sól

Jako kolejne zabezpieczenie przed atakiem słownikowym wprowadzono technikę dodawania do haseł **soli** (*salt*). Każde hasło przed zahaszowaniem jest uzupełniane przez losowo wygenerowany ciąg bitów (sól), który jest pamiętany jawnie (niezahaszowany) razem z zahaszowanym hasłem. Przy uwierzytelnianiu użytkownika, podane przez niego hasło zostaje zahaszowane z solą i porównane.

Utrudnienie dla krakerów polega na tym, że nie mogą korzystać z gotowej listy haseł słownikowych. Dla każdego znalezionego zahaszowanego hasła lista słownikowa musi być indywidualnie budowana i haszowana wraz z indywidualną wartością soli.

W przypadku zastosowania tęczowych tablic musiałyby one być wygenerowane dla wszystkich możliwych wartości soli. Przy niezbyt długiej wartości soli, np. 12 bitów stosowane w starszych systemach Unix, powoduje to powiększenie tablicy 4096 razy, co jeszcze jest wyobrażalnie możliwe. Jednak zastosowanie dłuższych soli, typu 100 bitów lub więcej, skutecznie wyklucza możliwość zastosowania takich tablic.

Dygresja — ćwiczenia z haszowaniem haseł z solą

Haszowanie hasła domyślną metodą DES, hasz ma 13 znaków, dwa pierwsze to sól:

```
whistler-572> mkpasswd -m des basia123     # losowa wartosc soli
afRI7OBQ18nIc
whistler-573> mkpasswd -m des basia123     # losowa wartosc soli
UT/tZsoSy6ZVs
whistler-573> mkpasswd -m des basia123     # losowa wartosc soli
a1B4DTn8jirTs
whistler-573> mkpasswd -m des basia123 a1  # wymuszona ta sama wartosc soli
a1B4DTn8jirTs
```

Haszowanie hasła metodą MD5 (id1), hasz ma 22 znaki, sól 8 znaków:

```
whistler-592> mkpasswd -m md5crypt basia123
$1$CzEaBRPm$rbhG9s2Zm6UCvH3kmxELV0
whistler-593> mkpasswd -m md5crypt basia123
$1$aBGCR155$B9QwheDcsEA1DIza/bvDl0
whistler-593> mkpasswd -m md5crypt basia123
$1$R6O2m1DL$2WvgqQ870EU1RS9kzTWPv0
whistler-594> mkpasswd -m md5crypt basia123 R6O2m1DL
$1$R6O2m1DL$2WvgqQ870EU1RS9kzTWPv0
```

MD5-Crypt haszuje metodą MD5 1000 razy stringa: hasła+soli+poprzedniego haszu.

Środki obrony przed atakiem słownikowym — złożoność haszowania

Długa sól skutecznie uniemożliwia zastosowanie tęczowych tablic do efektywnego ataku na bazę haseł. Np. użycie soli 100-bitowej wymusiłoby zwiększenie tęczowej tablicy $2^{100} \approx 10^{30}$ razy. Jednak ponieważ sól jest jawna, nie utrudnia ona zwykłego ataku słownikowego na pojedyncze hasła, których zahaszowana postać jest znana.

Innym utrudnieniem może być stosowanie bardziej złożonych obliczeniowo metod haszowania. Popularne metody takie jak MD5, SHA1, SHA256, są bardzo efektywne, pozwalając na obliczenie milionów haszy na sekundę, dla typowej długości haseł. Nadają się one dobrze do zastosowań takich jak obliczanie sygnatur dokumentów, albo podpisów cyfrowych, ale do haszowania haseł lepsze są metody kosztowne obliczeniowo. Do takich metod należą: scrypt, bcrypt, Argon2 i PBKDF2. Gdy obliczenie haszu trwa milisekundy zamiast mikrosekund, sprawdzanie hasła rzeczywistego pojedynczego użytkownika wydłuża się niezauważalnie, natomiast atak słownikowy — o wiele rzędów wielkości.

Środki obrony przed atakiem słownikowym — pieprz

Kolejną techniką stosowaną dla obrony przed atakami słownikowymi jest **pieprz** (*pepper*). Pieprzeniem(!) nazywa się dodawanie do hasła przed zahaszowaniem dodatkowego losowego stringa, podobnie jak dodaje się sól. Jednak w odróżnieniu od soli, pieprz nie jest przechowywany jawnie razem z hasłem.

Istnieje szereg wariantów technik pieprzenia. Jednym z nich jest przechowywanie pieprzu w systemie w postaci tajnej. Pieprz może być indywidualnie wygenerowany dla każdego hasła, co prowadzi do tworzenia i przechowywania dodatkowej bazy pieprzu.[2] Innym wariantem jest zastosowanie do wszystkich haseł jednej losowej wartości pieprzu, która może być zakodowana na sztywno w programie obliczającym hasze.

Jeszcze innym wariantem techniki pieprzenia jest dodanie do każdego hasła pieprzu indywidualnie wygenerowanego, ale niezbyt długiego, i wyrzucenie go po zahaszowaniu hasła. Przy sprawdzaniu hasła trzeba sprawdzać wszystkie możliwe wartości pieprzu, co jest równoważne znaczącemu wydłużeniu tego procesu, z analogicznym wydłużeniem ewentualnego ataku.

[2] Jest to podobne do skrytek w banku, do których otwarcia potrzebny jest klucz posiadany przez klienta, i jednocześnie drugi klucz przechowywany przez bank.

Bezpieczeństwo haseł — z punktu widzenia użytkownika

siła hasła — W oczywisty sposób trywialne hasła (basia123) stanowią słabe zabezpieczenie i są podatne na złamanie. Jednak trudne do złamania hasła stanowią utrudnienie w codziennym użyciu, a przy wielu hasłach którymi użytkownik musi posługiwać się na co dzień, zmusiłyby go/ją do zapisywania sobie haseł w miejscu łatwo dostępnym (np. na żółtej karteczce przyklejonej do monitora). Jest to zatem kompromis — użytkownik musi rozważyć jaka jest uciążliwość stosowania bezpiecznych haseł w porównaniu z zagrożeniem (utrata zawartości komputera, pieniędzy, pracy, proces sądowy, kompromitacja zawodowa lub rodzinna, itp.).

bezpieczeństwo wpisywania — Hasło może zostać przechwycone w momencie jego wpisywania przez użytkownika. Na przykład: fizycznie na ekranie, jeśli jest wyświetlane. Albo przez specjalne urządzenia przechwytujące (tzw. *keyloggery*) wpięte fizycznie w kabel klawiatury, lub program zainstalowany na komputerze. Albo z kolei przez nasłuchiwanie transmisji sieciowej (kablowej lub WiFi), gdy użytkownik wpisuje hasło do zdalnego systemu, i jest ono przesłane otwartym tekstem przez sieć.

fizyczny dostęp do komputera — Hasło do konta ustawione na komputerze, do którego włamywacz uzyska dostęp fizyczny (bo dostanie się do pomieszczenia, lub ukradnie komputer), w efekcie będzie nieefektywne, o ile setup komputera nie został zabezpieczony hasłem, i/lub dysk nie został zaszyfrowany.

Bezpieczeństwo haseł — z punktu widzenia programisty/administratora

przechowywanie haseł — Rolą programistów i/lub administratorów systemów komputerowych jest bezpieczne przechowywanie haseł. W praktyce oznacza to przechowywanie tylko wersji zahaszowanej, z zastosowaniem dodatkowych technik, takich jak sól, pieprz, i/lub dostatecznie powolna metoda haszowania, oraz właściwym zabezpieczeniem dostępu do bazy haseł. Hasła powinny być dostępne tylko za pośrednictwem uprzywilejowanego programu, który pobiera pojedyncze zahaszowane hasło w celu jego porównania z wersją uzyskaną od użytkownika.

utrudnienia dla włamywaczy — Warto i należy stosować proste techniki utrudniające skanowanie kont i haseł: kilkusekundowe opóźnienia zapytania o konto i hasło skutecznie wyklucza większość programów skanujących, a blokowanie adresów skąd przychodzą takie próby skanowania zabezpiecza system przez ponawianymi atakami.

Bezpieczeństwo haseł — z punktu widzenia producenta/instalatora systemów

domyślne hasła do urządzeń — Wiele różnych produkowanych na świecie urządzeń, jak również systemów oprogramowania, dostarczanych jest z preinstalowanymi fabrycznymi hasłami, których wielu użytkowników nigdy nie zmienia. Otwiera to ogromne pole do działania czarnych kapeluszy.

Ma to szczególne znaczenie zwłaszcza w połączeniu z modnymi technologiami IoT (*Internet of Things*) instalującymi procesory z dostępem do internetu na wielu urządzeniach codziennego użytku. Urządzenia te mają powszechnie znane domyślne hasła, bardzo słaby ogólny poziom zabezpieczeń, i liczne dziury, również ogólnie znane. Niektóre zamontowane są w urządzeniach, które mogą być niebezpieczne.

Na przykład przeprowadzone w roku 2010 ataki za pomocą robaka Stuxnet były możliwe dzięki wykorzystaniu domyślnego hasła do przemysłowego oprogramowania bazy danych SCADA firmy Siemens. Wskutek tych ataków ucierpiał m.in. irański zakład wzbogacania uranu, gdzie uszkodzeniu uległy wirówki przyspieszone poza krytyczną prędkość obrotową przez oprogramowanie sterujące.

Uwierzytelnianie z żetonem

Uwierzytelnianie z żetonem nie jest nowym wynalazkiem. Od wieków stosowane były różne formy żetonów uwierzytelniania: wymyślny sygnet potwierdzający tożsamość osoby, pieczęć królewska uwierzytelniająca dokumenty, albo klucz umożliwiający dostęp do kufra z zasobami (lub do wieży z zamkniętą księżniczką),

Współczesne wersje tokenów uwierzytelniania (żetonów) to: karty bankowe, czyli karty pamięci z hasłem zapamiętanym w postaci magnetycznej lub w elektronicznej pamięci ROM, tokeny generujące hasła jednorazowe, albo znacznie bardziej zaawansowane inteligentne karty procesorowe.

Pewną wersją uwierzytelniania z żetonem jest technika haseł jednorazowych. Użytkownik posiada listę takich haseł pierwotnie wygenerowanych na serwerze, i wykorzystuje je po kolei, każde tylko jeden raz. W praktyce, takie hasła działają bardziej jak żeton uwierzytelniania niż hasło, i ta metoda eliminuje wiele wad tradycyjnego systemu haseł.

Alternatywą dla listy jednorazowo wygenerowanych haseł jednorazowych są elektroniczne tokeny generujące takie hasła zsynchronizowane z aktualnym czasem.

Uwierzytelnianie z inteligentną kartą procesorową

O ile karty pamięci, tak magnetyczne jak i elektroniczne, są podatne na możliwość skopiowania, to tej wady nie mają karty inteligentne, zawierające procesor i niewielką pamięć. Uwierzytelnianie z użyciem takiej karty przebiega w trybie **zapytanie-odpowiedź** (*challenge-response*), gdzie karta nie odpowiada na zapytanie hasłem, tylko używa go do zaszyfrowania i deszyfrowania na wewnętrznym procesorze.

Karty inteligentne zawierające procesor i niewielką pamięć w połączeniu z metodą zapytanie-odpowiedź zapewniają bardzo wysoki poziom bezpieczeństwa. Hasło po pierwotnym wygenerowaniu jest zapisywane w pamięci karty i nie jest nigdy nigdzie przesyłane. Uwierzytelnianie polega na wygenerowaniu losowego tekstu i przesłaniu go do procesora karty, a następnie uzyskaniu od karty wersji tego tekstu zaszyfrowanej kluczem prywatnym karty. Jest on następnie rozszyfrowywany pamiętanym na serwerze kluczem publicznym, i porównaniu z wersją oryginalną tekstu. Ponieważ tekst jest losowy i jednorazowy, to ani znajomość jego postaci oryginalnej, ani zaszyfrowanej, nie daje żadnej możliwości przyszłego ataku.

Podobne technologie można zastosować z wykorzystaniem smartfonów, jednak smartfony same nie są systemami bezpiecznymi. Wręcz przeciwnie, same są wdzięcznym obiektem do różnych metod ataku.

Uwierzytelnianie biometryczne

Uwierzytelnianie biometryczne jest niewątpliwie najstarszą techniką uwierzytelniania. Rozpoznawanie twarzy, sylwetki, i innych cech biometrycznych było stosowane przy wpuszczaniu naszych przodków do jaskini zamieszkałej przez dane plemię. Delfiny (i wiele innych gatunków zwierząt) stosują sygnatury dźwiękowe do wzajemnego rozpoznawania. Koty znaczą terytorium indywidualnym zapachem.

Technologie biometryczne takie jak: rozpoznawanie odcisku palca, obrazu twarzy, obrazu siatkówki oka, głosu, itp., są coraz częściej stosowane w systemach dostępowych (i komputerowych). Są one wygodniejsze od haseł, a nawet od kart inteligentnych, i w teorii zapewniają wysoki poziom bezpieczeństwa. Jednak wiele z nich jest jeszcze w fazie intensywnego rozwoju i w praktyce nie są jeszcze aż tak wygodne i niezawodne jak te stosowane przez naszych przodków.

Niektóre wady uwierzytelniania biometrycznego:

- system ustawiony na wysoki wymagany poziom zgodności może nie rozpoznać użytkownika w stanie zmęczenia, zapocenia, makijażu, użycia soczewek kontaktowych, itp.; natomiast ustawiony na niski wymagany poziom zgodności może uznawać podobne obrazy za zgodne
- system może budzić zastrzeżenia etyczne, gdy będzie różnie traktował użytkowników różnej płci, wieku, rasy, budowy fizycznej, itp.
- bezpieczne przechowywanie danych biometrycznych jest wysoce krytyczne; w przypadku ich wykradzenia nie ma możliwości zresetowania danych jak haseł

Zabezpieczenia na poziomie programów

skanery antywirusowe

weryfikatory integralności: sumy kontrolne, tripwire

podpisywanie kodu

„piaskownice”

unikanie wirusów — „zdrowe” zachowania użytkowników

Zabezpieczenia na poziomie języka programowania

Zabezpieczenia na etapie kompilacji: wymagane zabezpieczenia mogą być deklarowane w programie, i implementowane przez kompilator. Na przykład, kompilator może wyróżnić odwołania do pamięci całkowicie bezpieczne, które mogą być wykonane bez dodatkowego sprawdzania, oraz niekoniecznie bezpieczne, które mogą być wykonywane inaczej, z dodatkowymi zabezpieczeniami. Te dodatkowe zabezpieczenia mogą być zrealizowane ze wsparciem ze strony systemu operacyjnego i/lub sprzętu.

Zabezpieczenia na etapie wykonywania: niekiedy realizacja zabezpieczeń na etapie kompilacji nie jest możliwa. Na przykład, w Javie jest możliwe dynamiczne ładowanie klas, nawet z lokalizacji sieciowej. Takie klasy z definicji są traktowane jako niezaufane, i ich dostęp nawet do lokalnych plików musi być weryfikowany. Maszyna wirtualna Javy w czasie ładowania klasy przypisuje jej domenę zabezpieczeń określającą uprawnienia tej klasy. Następnie, w czasie wykonywania programu, wygenerowanie żądania dostępu do określonego zasobu powoduje badanie stosu wykonania programu, aby ustalić która klasa wygenerowała to żądanie, i podjąć decyzję, czy żądanie należy spełnić.

Mity dotyczące zabezpieczeń

Użytkownicy, ale także administratorzy i programiści, lubią ulegać mitom. Istnieje wiele popularnych mitów dotyczących systemów zabezpieczeń.

Na przykład, każdy wolałby kupić dobry program zabezpieczający, i zapomnieć o problemach bezpieczeństwa. W rzeczywistości, żaden pojedynczy program, ani wielkość firmy, która go stworzyła, nie gwarantują bezpieczeństwa.

Nie należy również ufać nadmiernie żadnej pojedynczej metodzie, ani technologii zabezpieczeń. Każda może zostać przezwyciężona w sposób, którego nie sposób przewidzieć przy projektowaniu systemu zabezpieczeń. Natomiast dobrym podejściem jest stosowanie wielu technologii zabezpieczających, na różnych poziomach.

Jednocześnie nic nie jest w stanie zastąpić podstawowych, codziennych praktyk każdego użytkownika: wiedzy o zagrożeniach, wiedzy o ich unikaniu i zapobieganiu, i zastosowania tej wiedzy w praktyce, poświęcenia czasu i wysiłku, wykazania cierpliwości i wytrwałości, tworzenia wielopoziomowych zabezpieczeń (włącznie z kopiami zapasowymi), i ćwiczenie istniejących mechanizmów zanim zdarzy się nieszczęście (np. czy kopia zapasowa jest kompletna, czytelna, i w pełni wystarczająca).

Referencje

w tej prezentacji wykorzystane zostały materiały:

1. Rozdział 9 podręcznika A.S.Tanenbaum, H.Boss: Systemy operacyjne, Wydanie 4, 2015.

2. Rozdział 17 podręcznika A.Silberschatz, G.Gagne: Podstawy systemów operacyjnych, Wydanie 10, 2018.