Hledání hesla metodou pokus a omyl

* ruční nebo automatizovaný pokus uhodnout platné jméno a heslo

* obrana:
 - všechna defaultní hesla by měla být po nainstalování systému změněna
 - uživatelé by měli být nuceni zvolit obtížně uhodnutelná hesla
 - pokud je některý účet delší dobu nepoužívaný, přihlašování na něj by mělo být zakázáno
 - všechna úspěšná i neúspěšná přihlášení by měla být zaznamenána
 - při přihlašování by systém měl ohlásit datum, čas a stroj ze kterého proběhlo poslední přihlášení (= možnost detekce neoprávněného přístupu) a případné neúspěšné pokusy o přihlášení

Útoky typu DoS

* denial-of-service attacks = cílový systém nebude použitelný
 - pokud je cílový systém kritický, může být značný problém (systém pro e-komerci, řídící systémy apod.)
 - někdy se používá jako součást útoku proti síti (např. při převzetí relace)
* DoS útoky většinou dvou typů: zaplavení systému nesmyslným provozem nebo zaslání nesprávně vytvořeného paketu

* zaplavení systému nesmyslným provozem (packet flood)
 - např. SYN flood - poslat systému velké množství SYN paketů - jsou interpretovány jako žádost o navázání TCP spojení, pošle se odpověď SYN-ACK
  . správně by strana zahajující spojení měla odpovědět ACK, pak se mohou přenášet data
  . DoS útok - všechny žádosti mají falešné zdrojové IP adresy, na SYN-ACK nikdo neodpoví
  . časem se vyčerpá kapacita serveru pro zahájená spojení - další pokusy o spojení (i od legálních uživatelů) jsou odmítnuty
 - další typ - smurf attack - zaplavení sítě ICMP pakety s využitím sítí třetích stran
  . ping na broadcast adresu sítě třetí strany, zdrojová adresa nastavena na cílový stroj
  . pokud konfigurace sítě třetí strany dovoluje průchod ICMP broadcastů, všechny stroje odpoví ICMP ECHO_RESPONSE na adresu cíle; může vyčerpat kapacitu připojení cíle
  . podobným způsobem pracují tzv. distributed DoS attacks (DDoS)

* zaslání nesprávně vytvořeného paketu (malformed packet attack)
 - paket formátován neočekávaným způsobem, OS při zpracování paketu havaruje
 - typický příklad: IP fragmenty segmentu se překrývají

* obrana proti DoS útokům:
 - sledovat nové patche výrobce OS, instalovat (po ověření fčnosti) co nejdříve
 - nevpouštět do sítě broadcast a pakety tvrdící že pocházejí z vnitřní sítě
 - vyfiltrovat odchozí pakety s podvrženou zdrojovou IP adresou (všechny odchozí pakety musejí mít IP adresu podcházející z naší podsítě)

Útoky zevnitř systému

* když se cracker dostane do systému, může začít škodit
 - pokud je systém bezpečný, může uškodit pouze uživateli na jehož účet se dostal
 - v mnoha případech je to pouze vstupní bod pro získání přístupu k dalším účtům
 - nyní se podíváme na některé útoky, které může provádět buď cracker který se dostal na účet nelegálně, případně i legitimní uživatel (útoky na základě přetečení bufferu na zásobníku lze provádět z vnějšku systému i zevnitř)

* popíšu nejznámější chyby ve třech OS: UNIX, TENEX a OS/360, a popíšu

  principy návrhu bezpečných OS
* pak se budeme věnovat trojským koňům, zadním vrátkům do systému apod.

Známé chyby v bezpečnosti systému UNIX
.....

* utilita ”lpr” tiskne zadaný soubor na tiskárnu
 - ”lpr -r soubor” po vytisknutí soubor zruší
 - ve starších verzích systému bylo možné, aby kdokoli vytisknul a zrušil
   /etc/passwd

* pokud je v systému povoleno vytváření souborů core (”ulimit -c unlimited”),
  systém při ukončení procesu signálem zapíše obsah paměti procesu do souboru core
 - vytvořit link s názvem core na soubor /etc/passwd (”ln /etc/passwd core”)
 - spustit SUID program a vynutit si vytvoření core (např. zasláním SIGQUIT)
 - nový /etc/passwd mohl obsahovat několik řetězců zvolených útočníkem
   (např. argumenty příkazového řádku)

* útok pomocí mkdir
 - ve starších verzích UNIXu (SVR3.2) neexistovalo volání mkdir(), adresáře se
   vytvářely SETUID programem ”mkdir” pomocí volání mknod() (vytvořilo
   adresář vlastněný rootem) a chown() (změna vlastníka na uživatele, který
   program spustil)
 - útok - pokusit se mezi mknod() a chown() provést:
   rmdir foo; ln /etc/passwd foo
 - příkazový soubor který provádí opakovaně dokud se nepovede

Známá chyba v bezpečnosti systému TENEX
.....

* svého času populární OS na počítačích DEC 10
* zajímavý díky následující chybě (uváděna v knihách o OS jako typický příklad)

* OS podporující stránkovanou virtuální paměť
 - aby uživatel mohl monitorovat chování programů, mohl si objednat vyvolání
   fce při výpadku stránky
* soubor chráněné heslem
 - při otevření souboru musel program předložit správné heslo
 - OS kontroloval heslo znak po znaku a kontrolu hesla ukončil, jakmile
   zjistil chybný znak
* útok
 - pečlivě umístit heslo na hranici stránky tak, aby první znak hesla byl na
   jedné stránce, ostatní znaky na následující stránce (heslo např. ”aaaaaa”)
 - zajistit, aby druhá stránka nebyla v paměti (např. způsobit tolik odkazů,
   aby druhá stránka musela být vyhozena)
 - pokusit se otevřít soubor
  . pokud heslo nebude začínat znakem ”a”, skončí kontrolu a oznámí chybu hesla
  . pokud heslo bude začínat ”a”, přejde na kontrolu dalšího znaku => výpadek
    stránky
 - tj. místo očekávaných N^128 pokusů pro N-znakové heslo v ASCII kódu bude
   zapotřebí maximálně N*128

Známá chyba v OS/360
.....

* OS přístup k souboru ve dvou krocích
 - kontrola zda je přístup k souboru (použito jméno souboru uvedené v datové
   struktuře uložené v paměti procesu)
 - pak přístup k souboru (opět použito jméno souboru z datové struktury)
* OS umožňoval asynchronní čtení, např. spustit čtení ze souboru a pokračovat
  v další práci zatímco se zapisují data do určené oblasti paměti
 - trik - systému ukázat soubor, ke kterému máme právo přístupu
 - nechat přepsat asynchronním čtením, pokud se povedlo mezi dvěma částmi
   operace pak přístup k souboru podle volby útočníka

Návrh bezpečných systémů
.....

* obvykle najmout skupinu expertů (tiger teams, penetration teams)
* během let se objevilo několik oblastí, kde bývaly často chyby (Linde 1975)

1. požadujte paměťový prostor, diskový prostor apod. - pokud systém obsah neruší před jeho alokací, můžete tam najít zajímavé informace zanechané předchozím uživatelem
2. pokuste se provádět nelegální systémová volání, legální systémová volání s nelegálními parametry nebo nesmyslnými legálními parametry (jméno souboru několik KB dlouhé apod.); pokud systém chybně ošetřuje vstupní parametry, můžete ho zmást
3. při přihlašovací sekvenci stiskněte klávesu generující událost se speciálním významem, např. Break; v některých systémech tím zrušíte přihlašovací program a přihlášení bude považováno za úspěšné
4. pokud OS uchovává složité datové struktury v adresním prostoru uživatele (občas OS pro mainframy), pokuste se je modifikovat; např. pokud systém uchovává informace o otevřeném souboru v adresním prostoru uživatele a datovou strukturu mění při čtení a zápisu souboru
5. podívejte se do manuálu; pokud se někde píše ”nedělejte X”, vyzkoušejte tolik variant X, kolik vás napadne
6. přesvědčte systémového programátora, aby do systému přidal zadní vrátka
7. pokud vše selže, představte se administrátorovi jako ubohý uživatel, který zapomněl své heslo a potřebuje ho rychle; alternativní možnost - podplacení sekretářky (bývá málo placená a má přístup k všelijakým zajímavým informacím); tomuto poslednímu způsobu se říká ”sociální inženýrství”

* návrháři nových systémů kontrolují

* je zřejmé, že navrhnout bezpečný OS není snadné
* existuje několik principů, které by návrháři bezpečných systémů měli dodržet - platí i pro jiné SW systémy než OS
  - design systému by měl být veřejný; předpoklad, že oponent nepřijde na to jak systém pracuje (”security through obscurity”) slouží pouze k oklamání návrhářů systému
  - defaultní nastavení by mělo být ”přístup odepřen”; chyby kdy je legitimní přístup odepřen bývají ohlášeny podstatně rychleji než opačná situace
  - systém by měl vždy kontrolovat současná přístupová práva; mnoho systémů kontroluje přístupová práva pouze při otevření souboru => uživatel může otevřít soubor a mít ho otevřený, i když původní vlastník změnil přístupová práva nebo se pokusil soubor zrušit
  - každý proces by měl mít nejmenší možná privilegia; pokud může textový editor přistupovat pouze k editovanému souboru, jeho ”trojská” verze nám nemůže příliš ublížit (to UNIX nedodržuje: mnoho serverů má privilegia roota)
  - mechanismy ochrany mají být jednoduché, uniformní a mají být zabudovány do nejnižších vrstev systému; bezpečnost - podobně jako správnost - nelze k systému ”přidat”
  - zvolené schéma musí být psychicky akceptovatelné; pokud si uživatelé budou myslet, že chránit soubory dá moc práce, nebudou to dělat

Nejdůležitější pravidlo: KISS = ”Keep It Simple, Stupid”
* pokud je systém jednoduchý a elegantní, byl navržen jedním architektem, má několik základních pravidel která určují ostatní - pak má šanci být bezpečný
* pokud systém naopak musí dodržovat kompatibilitu se starými systémy které nejsou bezpečné, nebo je navrhován komisí která do návrhu začlení oblíbené vlastnosti každého svého člena (chybí architektonická koherence) pak budou pravděpodobně bezpečnostní problémy

Instalace pomocných programů útočníkem

* mezi pomocné programy patří trojští koně, falešné přihlašovací programy, logické bomby, zadní vrátka
* nejčastější nástroje mají útočníci v tzv. rootkitech, o kterých řeknu něco na závěr

Trojští koně

* trojský kůň = zdánlivě neškodný program, obsahující nečekanou nebo nežádanou fci (poškození nebo modifikaci souborů uživatele, zkopírování na místo odkud je útočník snadno získá apod.)
* je zapotřebí, aby si ho oběť spustila => buď si ho oběť sama nainstaluje nebo je nainstalován tak, aby byl obětí neúmyslně spuštěn

* první případ - oběť si trojského koně sama nainstaluje
  . například útočník umístí trojského koně na WWW jako program, který si pravděpodobně získá pozornost (hra, program který tvrdí že z CD ROM čtecí mechaniky udělá zapisovací, apod.)
  . nebo pošle oběti mailem jako spustitelnou ”pohlednici k narozeninám” apod.

* druhý případ - program do systému nainstaluje útočník
 - v systémech UNIX i Windows proměnná PATH definuje cesty, která jsou prohledávány při spuštění programu (např. PATH=.:/usr/bin:/bin:/usr/bin/X11)
 - pokud je některý z adresářů zapisovatelný, útočník tam může zanechat program
 - program může mít jméno odpovídající chybně zadanému příkazu, např. ”la” (místo ”ls”)
 - po spuštění programu provede nekalou činnost a pak vypíše: bash: la: command not found
 - další možnost - legální uživatel může zanechat trojského koně s názvem ”ls” ve svém domovském adresáři a udělat něco podezřelého; administrátor se podívá co dělá, spustí ”ls”; pokud se příkazy vyhledávají v aktuálním adresáři (adresář ”.” je v cestě), spustí se trojský kůň
  . to je důvod, proč příkazové interprety v UNIXu defaultně nehledají spustitelné soubory v aktuálním pracovním adresáři

* možné činnosti trojského koně
 - vytvořit SUID shell pro pozdější práci (na to postačují dvě volání: chown(”sh”, 0, 0); chmod(”sh”, 04555);)
 - pokud je nainstalován program pro on-line banking, může pomocí něj provést nějakou finanční transakci (Denningová 1999) apod.

* obrana
 - nespouštět neprověřené programy
 - některé tradiční trojské koně pro Windows jsou detekovány antivirovými programy

Falešné přihlašovací programy
....

* myšlenka příbuzná trojským koňům: program, který vytvoří obrazovku stejnou jako skutečný přihlašovací program, ale zaznamenává kombinaci jméno/heslo

* jediná obrana je zahájit přihlašování kombinací kláves, kterou uživatel nemůže zachytit, např. Ctrl-Alt-Del ve Windows NT/2000/XP

Logické bomby
....

* kód tajně vložený programátorem do firemního programu, spouštění v případě nějaké události (např. vyhození programátora z práce)
 - například pokud je programátor na výplatní pásce, nedělá nic
 - pokud se dva měsíce za sebou neobjeví, aktivuje se (Spafford et al. 1989)
 - příklad činnosti: náhodné poškozování souborů, zašifrování důležitých souborů (aby byla firma nucena zaměstnat jako ”konzultanta” pro vyřešení problému)

Zadní vrátka
....

* angl. backdoors, trapdoors
* umožňuje útočníkovi získat přístup k systému bez nutnosti standardního přihlášení

* klasická zadní vrátka - např. modifikace přihlašovacího programu tak, aby dovolil přihlášení komukoli, pokud se přihlásí jako ”zzzzzz”

```
while (TRUE) {                          while (TRUE) {
```

```
  printf(”login: ”);
  get_string(name);
  disable_echo();
  get_string(password);
  enable_echo();
  v = check_validity(name, password);
  if (v)
    break;
}
start_shell(name);
// normální kód
```

```
  printf(”login: ”);
  get_string(name);
  disable_echo();
  get_string(password);
  enable_echo();
  v = check_validity(name, password);
  if (v || strcmp(name, ”zzzzzz”))
    break;
}
start_shell(name);
// kód se zadními vrátky
```

* obrana v tomto případě + v případě logických bomb
 - code review - např. po otestování modulu programátor vysvětlí co program dělá

Otázka: co když bude upraven překladač tak, aby při překladu nezměněného zdrojového kódu programu ”login” vložil zadní vrátka a zároveň bude upraven tak, aby při překladu překladače z nezměněných zdrojových textů vložil kód pro upravený překlad překladače? (”Reflections on trusting trust.”)

* současné programy implementující zadní vrátka do systému většinou naslouchají na určeném portu a čekají na příchozí spojení

* obrana proti zadním vrátkům
 - administrátor by měl vědět co spouští na svém systému
 - pokud se najednou objeví proces poslouchající na portu nebo existující proces poběží jako root (v UNIXu i Windows NT zjistíme netstat -a), administrátor musí zjistit proč
 - některá zadní vrátka pro Windows NT mohou být zjištěna antivirovými programy

* odposlouchávající zadní vrátka - aby nebyl detekovatelný programem netstat, odposlouchává provoz, reaguje na provoz určený pro něj
 - některé přepínají do promiskuitního režimu, reagují na provoz určený pro jiný stroj => obtížnější detekce

* obrana proti odposlouchávajícím zadním vrátkům
 - administrátor musí vědět které procesy mohou běžet jako root, jiné procesy nutné přezkoumat
  . útočníci je pojmenovávají ”nevinnými” jmény, např. ”scsi”, ”ups” apod.

❇