- da preprečimo deadlocke imamo v operacijskih sistemih kritična območja, da dva procesa ne moreta hkrati dostopati do istega vira
- če imamo globalno spremenljivko `x`, ki jo v funkciji povečamo in poženemo 10000 instanc procesa, se lahko zgodi, da dva procesa prebereta isto vrednost, če je context switch pred incrementom in bosta shranila isto vrednost `x`-a
- ko proces pokliče nekaj v kernelu, se tisto običajno izvaja asinhrono od ostalih stvari, zato so tu časovno odvisni napadi najpogostejši

Časovno odvisni napadi:

- sistem privede v stanje, kjer lahko pridobimo neavtoriziran dostop ali izvedemo operacijo.
- ko npr. odpiramo file, moramo počakati, da je file prost, da res dobimo dostop do njega

```
if (isopen("test")) return;
FILE f = open("test", "rw");
```

- ^ open še vedno lahko ne dela, ker je time of check (TOC) drugačen od time of use (TOU) in se vmes lahko file odpre od nekega drugega procesa
- npr. ko koda preverja ali je admin field v JWT nastavljen, vmes probamo context switch narediti, da se noter napiše admin=true
- običajno pridemo samo do DoS oz. DDoS

Hipotetični primer:

- dvakratni klik na gumb za nakup
- če nimamo locka, da za en id lahko naredimo samo en nakup, nam lahko šteje kot dva nakupa
- podatkovne baze: ko se spreminja neka vrstica, se locka, dokler ni pisanje končano - so atomarne

Primer volitve:

```
voli(uid, cid) {
    if (se_ni_volil(uid)) // TOC
        error;
    insert_volilec(uid, cid); // TOU
}
```

- ^ to ima dve atomarni operaciji (`se_ni_volil` in `insert_volilec`), zato jiju rabimo dati v isto transakcijo za bazo, da je vmes baza lockana in nič drugega ne more vmes brati/pisati stanja v bazo
- če večkrat pišemo v isti del baze, lahko pridemo v nek corrupted state
- ponavadi rabiš za te stvari nek dodaten dostop do interface-a, ne samo prek web

Primer Laravel:

- v local storage prepiše file z istim imenom in če daš dva fila z istim imenom, se lahko en checka in drugi izvede
- problem: če se nek vir spremeni med TOC in TOU (če imamo različna TOC in TOU)

Primer prepisovanje canary:

- problem je, ker je TOC za canary šele na koncu funkcije

Primer:

-
```
if ( a2 == 26214 )
{
 printk("Your flag is at %px!\n", flag);
 result = 0LL;
}
else if ( a2 == 4919 && is_userspace(*v5)
 && *(v5 + 8) == strlen(flag) )
{
 for ( i = 0; i < strlen(flag); ++i )
  if ( *(*v5 + i) != flag[i] )
    return 22LL;
 printk("%s\n", flag);
 result = 0LL;
}
else
 result = 14LL;
return result
```
- imamo naslov do flaga, flag je v kernel space
- v5 je nek naslov, želimo si da bi dali sem nalov do flaga
- v else if se preverja, da je v5 pointer na userspace, potem pa pred for loopom lahko z race conditionom spremenimo v5 na naslov flaga v kernel space
- v Javi si lahko s `synchronised` zagotovimo, da ne bo problemov, da se med TOC in TOU vir spremeni (med threadi se ne more spremeniti - bomo dobili vedno konsistenten rezultat); to je tradeoff s hitrostjo

Primer:

Victim	Attacker

`if (access("file", W_OK) != 0) {`
`    exit(1);`
`}`	

	After the access check, before the open, the attacker replaces `file` with a symlink to the Unix password file `/etc/passwd`: `symlink("/etc/passwd", "file");`

`fd = open("file", O_WRONLY);`
`write(fd, buffer, sizeof(buffer));`
Actually writing over `/etc/passwd`	

- ko damo symlink na /etc/passwd bomo lahko brali to datoteko
- običajno to delamo s programi, ki imajo drugačen privilege level

Side-channel napadi:

- okoli dodatnih ovinkov probamo dobiti informacijo, ne preko glavnega kanala
- lahko iz npr. clock speeda, časa za operacijo, porabe elektrike (količine toka), vrtenja diskov
- iz nekih informacij probaš izvedeti tisto informacijo, kar te res zanima; npr. veš, da je nekdo v službi, če gori luč; veš kakšno je geslo, če so neke črke bolj zlizane
- dobimo nove informacije brez tega, da spreminjamo proces, ampak gledamo "residuale"
- najbolj pogosti side channel: čas za operacijo in poraba elektrike
- čas za operacijo:
  - za preverjanje gesel

```
for (i : 0..len) {
    if (password[i] != guess[i]) {
        err
    }
}
```

-
  - ko se ena črka ne ujema, bo bilo prej konec - lahko si narišemo, katere črke rabijo več časa kot ostale
  - če vedno preverjamo vse znake ali pa damo nek random timeout, bomo še vedno lahko videli, katere črke rabijo več časa
  - rešitev: hashamo geslo s soljo (da se ista gesla ne preslikajo v iste hasha), kot user ne poznamo hasha - ne vemo v kaj se je naš `guess` spremenil in ne vemo, kaj smo napisali prav in kaj ne

Meltdown in Spectre:

- dela glede na branch prediction - imamo prediktorsko tabelo, ki nam napove kdaj bo branch true, kdaj false
- lahko beremo podatke drugega procesa, do katerih ne bi smeli imeli dostopa, ker zaradi predikcije beremo podatke drugega procesa iz predpomnilnika in potem traja še dve urini periodi preden se discardajo (out-of-order execution, OOE)
- na nivoju cevovoda operacijski sistem ne more nadzirati, kateri proces ima do česa proces
- ukazi še vedno naložijo podatke v medpomnilnik - te lahko nato ”prenesemo” iz enega programa v drugega
- zakaj se OOE ne znebimo? ker bi potem počasneje delalo - tradeoff hitrosti in varnosti
- rešitev: posodobimo mikrokodo za procesor
- side-channel nam lahko pomaga, da dobimo neke informacije, da izvemo kako skočiti iz sandboxa brskalnika

Primer HTTP side channel attack:

- ko v bazi ni userja, v katerega se želimo loginati, bo trajalo različno časa kot v userja, ki obstaja - ko vemo, da user obstaja, se lahko nanj osredotočimo