Nearchitekturní útoky

Martin Očenáš

Vysoké učení technické v Brne, Fakulta informačních technologií
Božetěchova 1/2, 612 66 Brno - Královo Pole
iocenas@fit.vutbr.cz

Nearchitekturní útoky

Co jsou to nearchitekturní útoky?

- Útoky, které nejsou založené na architektuře CPU.
  - Nelze je vyčíst z dokumentace nebo specifikace instrukcí CPU.
- Úzce souvisí s postranními kanály.
- Často zneužívají optimalizací procesoru.
- Jsou založeny na detailní znalosti systému.

Motivace

- Nearchitekturní útoky jsou těžko odhalitelné.
- Mohou ukrást důvěrné informace, např. šifrovací klíče.
- Zpravidla nejsou v rozporu s implementací bezpečnostní politiky.
- Je obtížné se proti nim bránit.

Bezpečnost procesorů

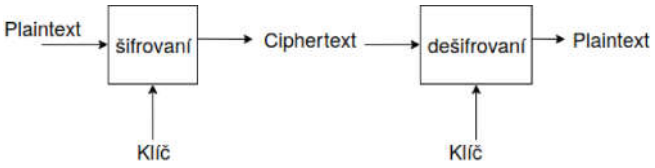
- Procesory dnes nepovažujeme za zcela bezpečné.
- Procesor by měl, společně s OS, zajišťovat oddělení procesů.
- Vykonávání na procesoru zanechává měřitelné vedlejší efekty.

Co se dnes dozvíte

- Rychlokurz kryptografie
- Útoky orientované na data.
  - Datová cache
- Útoky orientované na tok programu.
  - Instrukční cache.
  - Predikce skoků.
  - Sdílené funkční jednotky
- Nedávné reálné útoky.
  - Rowhammer.
  - Spectre & Meltdown.

Rychlokurz kryptografie

Obecný princip šifrování



RSA

- Asymetrický šifrovací algoritmus.
- Založený na faktorizaci prvočísel.
- Pracuje s velkými čísly, i zprávu chápe jako číslo.
- Důležité prvky v šifrování:
  - n - modulus.
  - e - veřejný exponent.
  - d - soukromý exponent.
  - Soukromý klíč - (n,d).
  - Veřejný klíč - (n,e).
- Šifrování:

$$c = m^e \% n$$

- Dešifrování:

$$m = c^d \% n$$

AES

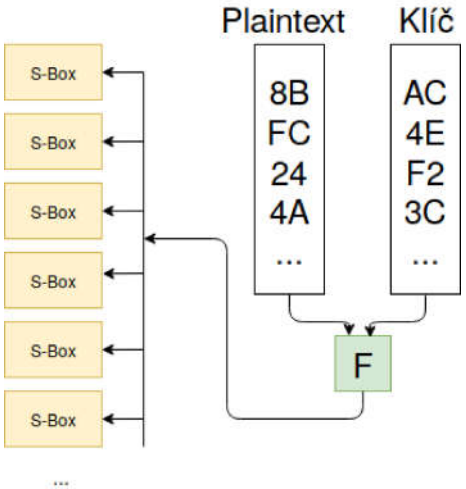
- Symetrický, blokový šifrovací algoritmus.
- Skládá se ze 4 hlavních fází:
  - Substituce skrz S-boxy.
  - Maticový posuv bajtů.
  - Maticové násobení bajtů.
  - XOR se subklíčem.
- Tyto se opakují v 10-14 kolech.

Substitution box (S-box)

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Obrázek: `https://en.wikipedia.org/wiki/Rijndael_S-box`

Útok na S-box



Útoky orientované na data

Útoky orientované na data

- Uvažujme 2 procesy: *útočník* a *oběť*.
- Útočník analyzuje přístupy oběti do paměti.
- Zneužívají pozůstatky předchozího vykonávání v procesoru.
- Místa v paměti, na která přistupuje kryptografický algoritmus, jsou odvozena od klíče a plaintextu.
- Pokud zjistím, do kterých částí paměti šifrovací proces přistoupil a znám plaintext, mohu zjistit klíč nebo jeho část.

Cache

- Optimalizační prostředek pro načítání dat z hlavní paměti.
- Po prvním načtení dat z paměti jsou tato uložena v cache.
- Data jsou blíže procesoru a jsou načítání rychleji.
- Při načítání dat z paměti se nejprve hledají v cache, teprve když tam nejsou, tak se hledají v paměti.
  - *Cache hit* - načítaná data jsou v cache.
  - *Cache miss* - načítaná data nejsou v cache a musí se načíst z paměti.

Mapování do cache

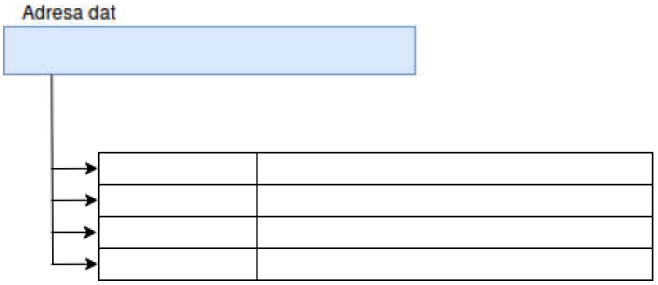
- Přímé.
- Plně asociativní.
- Skupinově asociativní.

Přímé mapování cache

- Adresa v cache je přímo dána částí adresy dat.
- Jeden blok dat se vždy mapuje do stejné buňky v cache.

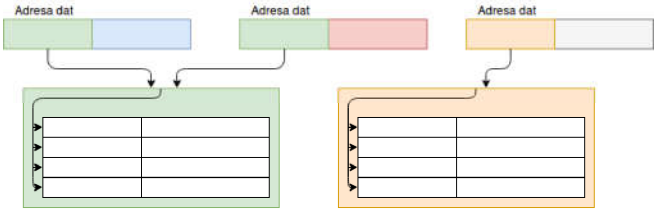
Plně asociativní cache

- Každý blok dat se může mapovat do libovolného řádku cache.



Skupinově asociativní cache

- Adresa dat určí skupinu v cache.
- V rámci skupiny se mapuje plně asociativně.



Uvolňování cache

- Cache má omezenou velikost, typicky výrazně menší než hlavní paměť.
- V případě že nově načtená položka vyžaduje místo v cache, které je již obsazeno, je stará položka vyřazena.
- Existuje více strategií jak určit položku pro vyřazení.
- Do jedné buňky cache se reálně načítá celý blok dat. Zde budeme pro zjednodušení uvažovat, že je to pouze jedna položka.

Útok na cache

- Cílem útočného procesu je zjistit, do jakých částí paměti přistupuje proces oběti.
- Z doby přístupu do paměti je možné změřit zda nastal cache-hit nebo cache-miss.
- Jsou různé typy útoků, podle toho jaké informace dokáže útočník zjistit.
  - Útoky orientované na čas.
  - Útoky orientované na trasu.
  - Útoky orientované na přístup do paměti.

Útoky orientované na čas

- Útočný proces měří celkovou dobu běhu procesu oběti.
- Je schopen odvodit kolik nastalo celkově cache-hit a cache-miss.
- Podle statistického měření, pro různé hodnoty plaintextu můžeme odvodit část klíče.

Jak zjistit klíč?

Předpokládejme že:

- Šifrovací proces přistupuje k S-boxům v paměti.
- Proces šifruje po jednotlivých bajtech.
- Adresa v paměti je dána xorem plaintextu a klíče.
- Před začátkem šifrování je cache prázdná.

Pak

- Přístup k prvnímu S-boxu bude vždy cache-miss.
- Pokud při přístupu k druhému S-boxu nastane cache-hit, pak víme:

$$P1 \oplus K1 == P2 \oplus K2$$

- Můžeme vyzkoušet všechny hodnoty P2, pouze pro jednu by měl nastat cache hit.

Postup útoku

- Vygeneruj náhodný plaintext.
- Vytvoř modifikace plaintextu pro všechny možné hodnoty P1 $\oplus$ P2.
- Spočítej průměrnou dobu šifrování všech plaintextů.
- Pouze pro jednu hodnotu P2 bude staticky kratší doba výpočtu.
- Ze znalosti:

$$P1 \oplus K1 == P2 \oplus K2$$

Jsme schopni odvodit výsledek:

$$K1 \oplus K2$$

- Obdobně postupuj pro další bajty.
- Teoreticky takto dokážeme zjistit vztah mezi všemi bajty klíče.

Reálný útok

- Do jednoho bloku cache se vejde víc než jedna položka.
- Při načtení jedné položky z paměti se jich do cache načte víc.
- Cache-hit pro šifrování P2 nastane pro více hodnot P2.
- Nezjistíme přesný vztah mezi K1 a K2, ale pouze vztah mezi některými jejich bity.
- Útok je však možné provádět i vzdáleně.

Příklad útoku:

- V roce 2004 byl prezentován útok na 128 bitový AES [1].
- Útok dokázal zkrátit efektivní délku klíče na 32 bitů.

[1] ACIIÇMEZ, Onur; SCHINDLER, Werner; KOÇ, Çetin K. Cache based remote timing attack on the AES. In: Cryptographers' track at the RSA conference. Springer, Berlin, Heidelberg, 2007. p. 271-286.

Útoky orientované na trasu

- Trasa *(trace)* je posloupnost cache-hit a cache-miss, které udělal proces oběti.
- např.

HHMMHMHM

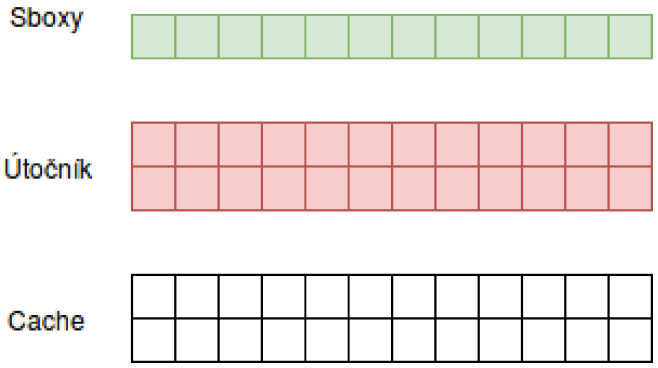
- Z trasy je možné odvodit v kterých částech programu proces přistupoval na stejná místa do paměti.
- Získání trasy je v praxi velmi problematické.

Útoky orientované na přístup do paměti

- Zjišťují do kterých částí paměti přistupuje šifrovací proces.
- Předpokládá že adresa bloku v paměti je dána hodnotu plaintextu a klíče.
- Pokud zjistíme, do kterých částí přistupoval šifrovací proces, můžeme odvodit hodnotu klíče.

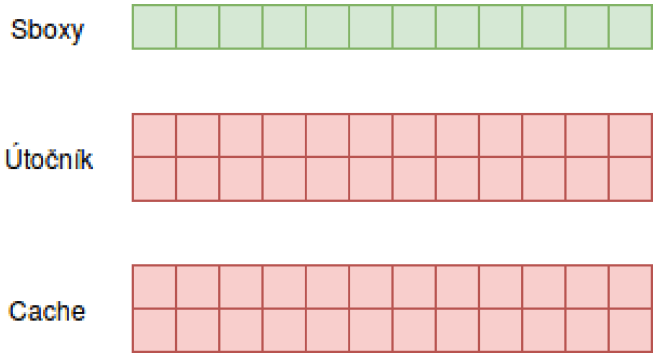
Popis útoku

- Uvažujeme skupinově asociativní cache.



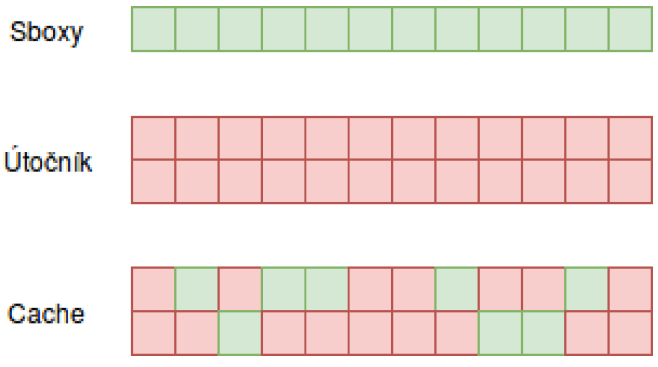
Popis útoku

- Útočník nejdříve naplní cache svými daty.
- Poloha dat musí být zvolena tak, aby v cache kolidovala s polohou S-boxů.
- Následně spustí šifrovací proces.



Popis útoku

- Šifrovací proces využije určité S-boxy.
- Útočník je schopen, dle doby čtení svých dat, zjistit, jaké S-boxy byly využity.



Výsledek útoku

- Pokud útočník zjišťuje stav v cache až po dokončení šifrování, nezíská mnoho informací.
- Pravděpodobně budou využity všechny S-boxy.
- Pokud nějaký S-box nebyl využit, můžeme vyřadit některé hodnoty klíče.

Pokud však máme paralelismus

- Útočník může opakovaně číst cache i v průběhu šifrování.
- Může zjistit které S-boxy jsou využity opakovaně a přibližně kdy.
- Přidáme-li hyperthreading, pak může útočník získávat tato data v reálném čase.
- S těmito informace je teoreticky možné zjistit celý klíč po jednom běhu algoritmu AES.

Percivalův útok na RSA

- Demonstrován v roce 2005 [2]
- Založen na přístupech do paměti.
- Vyžaduje hyperthreading.
- Zkoumá posloupnost násobení a umocňování v šifrování.
- Při útoku na OpenSSL a 512b RSA klíč, dokázal zrekonstruovat cca 200b soukromého exponentu.

[2] PERCIVAL, Colin. Cache missing for fun and profit. 2005.

Útoky orientované na tok program

Útok na tok programu

- Cílem útočníka je zjistit tok programu oběti.
  - Tedy jaké instrukce byly vykonávány, kdy a kolikrát.
- Blokové šifrovací algoritmy mají tok programu většinou konstantní.
- U RSA se tok programu výrazně liší v závislosti na klíči.
  - U RSA hodnota klíče slouží jako exponent pro umocňování. Pravděpodobně bude určovat počet iterací nějakého cyklu.
  - I z toku programu jednoho běhu RSA je možné odvodit celý soukromý klíč.

Instrukční cache

- Mnoho procesorů má oddělené cache pro data a instrukce.
- Instrukční cache funguje obdobně jako datová, pouze uchovává instrukce.

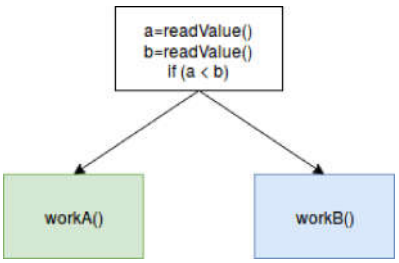
Útok na I-cache

- Princip stejný jako při útoku na datovou cache.
- Útočník provádí velké množství zbytečných instrukcí, aby vyprázdnil i-cache.
- Následně sleduje zda-li oběť naplnila i-cache svými instrukcemi.
  - Pokud ano pak se určité instrukce útočníka vykonají pomaleji.
- Z adres instrukcí, o které útočník v i-cache přišel, je možné odvodit adresu instrukcí použitých v procesu oběti [3].

[3] ACIIÇMEZ, Onur. Yet another microarchitectural attack:: Exploiting i-cache. In: Proceedings of the 2007 ACM workshop on Computer security architecture. ACM, 2007. p. 11-18.

Predikce skoků

- Moderní procesory načítají instrukce ještě v době vykonávání předchozích instrukcí.
- Problémem jsou podmíněné skoky v programu.

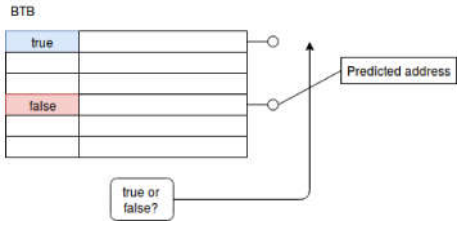


Prediktor skoků

- Jednotka uvnitř procesoru.
- Odhaduje cílovou adresu podmíněného skoku.
- Skládá se ze dvou hlavních částí:
  - Branch Target Buffer (BTB).
  - Logika predikce skoku.
- Po predikci procesor zahájí tzv. spekulativní vykonávání.

Branch Target Buffer

- Obsahuje adresy na které mohou skočit instrukce podmíněného skoku.
- Slouží jako cache na tyto adresy, mapování adres do BTB není náhodné.
- BTB má omezenou velikost.
- Aby mohlo začít spekulativní vykonávání, je třeba aby cílová adresa byla v BTB.
- Pokud dojde k BTB-miss, pak se do BTB vloží skutečná cílová adresa.



Útok na predikci skoků

- Velmi podobný útoku na i-cache.
- Útočník provádí velké množství podmíněných skoků aby naplnil BTB.
- Proces oběti do BTB následně vloží adresy míst, na která podmíněně skočil.
- Útočník poté měří dobu vykonávání svých podmíněných skoků, aby zjistit která z jeho položek z BTB vypadla.
- Jelikož adresy se do BTB vkládají podle určitého klíče, útočník dokáže odvodit, kam oběť skočila. [4]

[4] ACIIÇMEZ, Onur; KOÇ, Çetin Kaya; SEIFERT, Jean-Pierre. On the power of simple branch prediction analysis. In: Proceedings of the 2nd ACM symposium on Information, computer and communications security. ACM, 2007. p. 312-320.

Sdílené funkční jednotky

- Vyskytují se u procesorů s hyperthreadingem.
- Hyperthreading umožňuje dvěma procesům běžet současně na stejném jádře procesoru.
- Oba procesy mají vlastní kontext avšak sdílejí většinu komponent daného jádra.

Útok na sdílené funkční jednotky

- Na rozdíl od předchozích útoků nepracuje s uloženou informací.
- Vyžaduje aby útočník i oběť běželi na stejném jádře.
- Útočník neustále vytěžuje určitou funkční jednotku a sleduje dobu odezvy.
- Pokud je odezva dlouhá, tak ví že oběť právě využívá stejnou funkční jednotku.

Příklad útoku na RSA

- Útočník sleduje vytížení násobičky.
- Je schopen zjistit kolikrát a jakých časových intervalech oběť násobí.
- Množství násobení je v RSA závislé na hodnotě klíče.
- Z vytížení násobičky dokáže útočník vydedukovat část klíče. [5]

[5] ACIICMEZ, Onur; SEIFERT, Jean-Pierre. Cheap hardware parallelism implies cheap security. In: Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2007). IEEE, 2007. p. 80-91.

Ochrana

Na úrovni implementace SW:

- Zahřívání cache (cache warmup) - naplnění cache před začátkem šifrování.
- AES - změna pořadí S-boxů v paměti.
- RSA - konstantní tok instrukcí při násobení.
- ...

Na úrovni HW a OS:

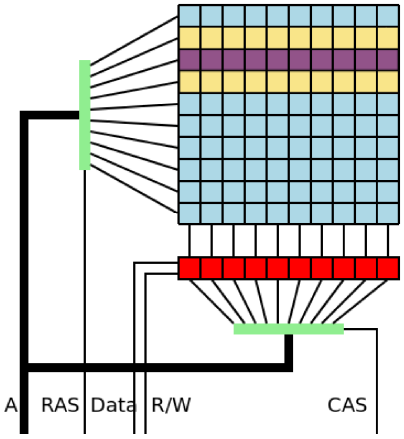
- Nepoužívat hyperthreading.
- Oddělení cache pro různé procesy.
- Invalidace cache při přepnutí kontextu.

Nedávné útoky

Rowhammer

- Obsažena v DRAM.
- DRAM ukládá bity v kondenzátorech.
- Kondenzátory si vybíjí a je třeba jejich hodnotu pravidelně obnovovat.
- Řadič paměti ví, jak často je třeba tuto hodnotu obnovovat.
- Každé čtení z paměti způsobí elektromagnetickou interakci s okolím.
- Časté čtení z jedné buňky paměti může způsobit rychlejší vybití kondenzátoru v sousední buňce - změnit její hodnotu.

Rowhammer



Obrázek: `https://en.wikipedia.org/wiki/Row_hammer`

Rowhammer

```
1 code1a:
2 mov (X), %eax    // read from address X
3 mov (Y), %ebx    // read from address Y
4 clflush (X)      // flush cache for address X
5 clflush (Y)      // flush cache for address Y
6 jmp code1a
```

Zneužitelnost:

- Teoreticky umožní přepsat libovolná data v paměti (i ty co jsou označeny jako read only).
- Eskalace práv.
- ...

Obrana:

- Častější obnova buněk v paměti.
- Používat ECC (Error-correcting code) paměť. Avšak také nechrání dokonale [6].

[6] https://www.vusec.net/projects/eccploit/

Rowhammer 2.0

- Stejný princip jako původní rowhammer.
- Zveřejněn v srpnu 2017.
- Tentokrát uplatněn na SSD disky.

Zneužití:

- Vyžaduje vytvoření velkého souboru na disku.
- Umožňuje např změnit i-node tak, aby byl vlastník nastavený na roota a nastavený SUID bit.
- Demonstrován útok s úspěšností 99,7%.

Obrana:

- Problematická.
- Šifrování disku.

Throwhammer

- Vzdálený Rowhammer na RAM. [7]
- Stačí upravené síťové pakety.
- Vyžaduje 10Gb/s spojení a RDMA (Remote DMA).

[7] https://www.bleepingcomputer.com/news/security/researchers-come-up-with-a-way-to-launch-rowhammer-attacks-via-network-packets/

Útoky na spekulativní vykonávání

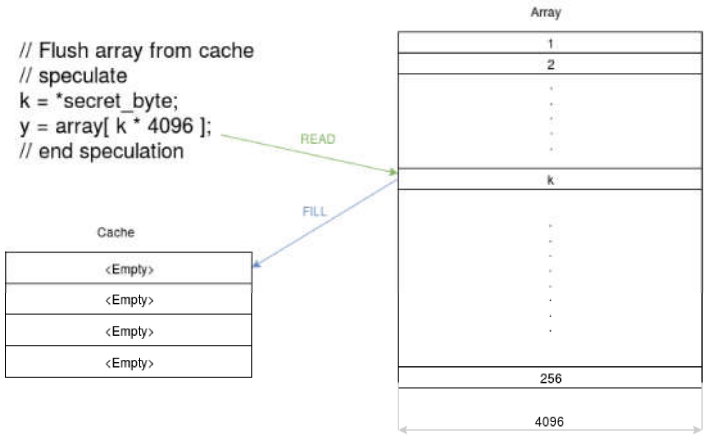
- V lednu 2018 zveřejněny útoky Spectre v1,v2 a Meltdown, zneužívající spekulativního vykonávání [8].
- Po nezdařené spekulaci existují měřitelné výsledky toho, jak spekulace probíhala.
- Pokud ve spekulaci použijeme nepřístupná data, nedojde k SEGFAULT nebo page fault.
- Můžeme takto přistoupit k paměti cizího procesu či jádra.

Jak to funguje:

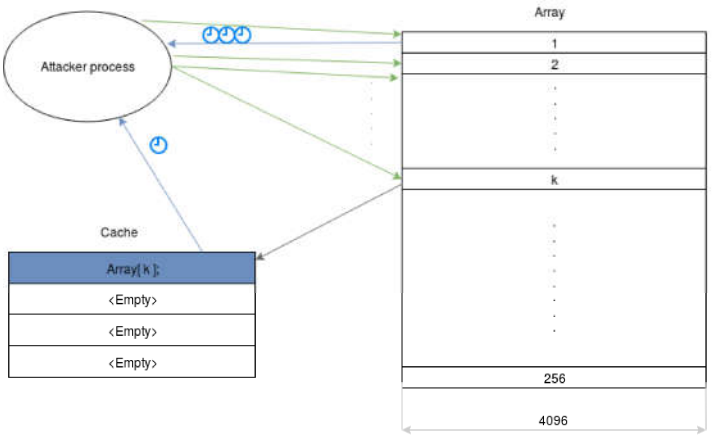
- Ve spekulaci načteme tajný bajt a použijeme jej jako index k naplnění cache.
- V normálním kódu pak pomocí měření času zjistíme, která adresa byla načtena do cache.

[8] https://spectreattack.com/

Podstata útoku



Podstata útoku



Spectre v1 - CONDITIONAL BRANCH MISPREDICTION

Útočný kód:

```
1 if (x < array1_size)
2 y = array2[array1[x] * 4096];
```

Předpokládáme že:

- Útočník má k dispozici *array1*, *array2* a může ovládat hodnotu **x**.
- Hodnota *array1_size* není načtená v cache a vyhodnocení podmínky bude trvat dostatečně dlouho.
- Útočník natrénoval prediktor skoků, aby spekulativně vykonal tělo podmínky.
- Hodnota **x** ukazuje mimo pole *array1* tak, aby výsledná adresa **array1 + x** ukazovala na cílené místo v paměti.

Meltdown

- Paměť jádra bývá mapována do paměti procesu.

```
1 kernel_byte = *kernel_address;
2 dummy = array[kernel_byte * 4096];
```

- Přístup do paměti jádra způsobí page fault.
- Zachycení chyby přístupu do paměti.
- Je možné jen na procesorech Intelu, protože příliš pozdě kontrolují supervisor bit.

Pokračování spekulativních chyb

- Původně nalezeny spectre v1,v2 a meltdown.
- Nalezena spousta dalších variant:
  - Foreshadow (L1 Terminal Fault).
  - Spectre NG.
  - NetSpectre.
  - BranchScope.
  - ...
- Chyby jsou v CPU, GPU, hypervizorech, ...