EHU/UPV

Ingenieritza informatikoa

Meltdown

Egilea:
Iker Sandoval

2025/05/10

Índice

1. Sarrera

Informatika-segurtasunaren arloan, ahultasunak ustiatzea ezinbesteko praktika da, sistemak nola konprometitzen diren eta, azken finean, nola babestu ikasteko.

Meltdown erasoak hainbat prozesadoretan dauden ahultasun bat ustiatzen du, batez ere Intel prozesadoretan. Ahultasun honek, erabiltzaile baten prozesu bat kernel memoriaren eskualde pribililegiatuetan irakurtzea. Meltdown, Out-of-order execution eta side-channel attack erabiltzen ditu erasoak egiteko.

2. Tresnak

- Linux bat x86-64 egiturarekin
- Kompiladore bat (gcc)
- Github

3. Ingurunea

3.1. Out-of-order execution

Optimizazio teknika bat da non CPU-aren exekuziorako erabiltzen diren unitateen erabilpena maximizatzea helburu dauka.

Horretarako, 1967. urtean Tomasulok sortu zuen algoritmoa erabiltzen du. Aginduak sekuentzialki exekutatu ordez, baliabideak eskuragarri dauden bezain laster, erabiltzen dira beste agindu bat exekutatzeko. Orduan, aginduak ordenetik kanpo exekuta daitezke. Honen adibide bat izan daiteke lehenengo agindu bat denbora asko irautea baliabide baten zain eta hurrengo aginduak aurrekoa baino lehen amaitzea zeren eta ez daude baliabideen zain. Dena ondo atera bada, exekuzioa jarraituko da. Baino arazoren bat baldin badago, arazoa sortu duen agindutik aurrera exekutatu diren aginduak exekuziotik kanporatzen dira, nahiz eta amaituta egonda. Berriro exekutatuko dira hasieratik behar bada.

3.2. Side channel attack

Flush+Reload teknika aplikatuko dugu, bertan cachean dagoen memoria partekatuaren informazioa eskuratzea helburu dauka.

Cache memoriaren garbiketa (Flush) bat egiten da (clflush instrukzioa bidez). Oso garrantzitzua da amaituta egotea, bestela, guk nahi ditugun neurketetan eragina izan dezakete daturen bat kargatuta badago.Behin katxe memoria garbituta dagoenean, biktima prozesua exekutatuko da eta helbide hoietako edukia erabilzten saiatuko da, berriro datu hori kargatzen saiatuko da eta denboraren arabera erasotzaileak helbidearen informazioa jakin dezake. Erasotzaileak, jakin dezake ea biktimak helbide hori erabili duen (katxean berriz kargatu delako, azkarrago irakurtzen delako) edo ez (memoria nagusitik datorrenean motelago doa).

4. Analisia.

4.1. Nondik dator

2018ko urtarrilean eraso berri bat publikatu zen Spectrerekin batera. Google ProyectZero, Cyberus Technology eta Graz University od Technology

4.2. Analisia

Lehen aipatu dugun bezala, orden kanpoko exekuzioa (out-of-order execution) erabiltzen du erabiltzaile ingurunetik kernelaren memoria irakurtzeko, nahiz eta baimenak ez izan.

Erasoak memoriaren sektore baten saitatuko da irakurtzen prozesadoreak hurrengo aginduak exekutatzen saiatzen ari den bitartean. Nola memoriaren zati pribilegiatu batera irakurtzen saiatuko da, salbuespen bat

sortuko du eta agindu horretatik aurrera exekutatu diren aginduak atzera botako ditu. Ateratako informazioa katxean gordeta geratuko da eta denbora limitatu batean ilegala den aginduaren emaitza erabili ditzakegu.

Behin espekulatiboki balore hori irakurrita izanda, 256 posizioko array batean gordeko da eta honek katxearen lerro konkretu bat kargatzen du. Erasotzaileak array-aren posizioa irakurtzeko denbora neurtzen du jakiteko katxearen zein lerrotan gorde den, filtratutako byte irakurtzeko.

Nola exekuzioan salbuespenak gerta daiteke eta oso garrantzitzua da kontrolatuta izatea katxearen irakurpena egiteko. Honen adibide bat izan daiteke memoria irakurtzerakoan salbuespen bat gertatzea eta horretarako SIGSEGV seinaleak inplementatuko dira, erasoaren faseak desberdintzeko.

Kernelaren ingurunean dauden memoriaren zatiak mapeatuta irakurri dezakegu, desplazamendu baten bidez memoria byteka irakurtzea lor dezakegu.

4.3. Eragina

Meltdown eragin kritiko duen eraso bat da non erasotzaileari memoria irakurtzeko ahalmena ematen dio. Baino, 2018 sortu zen eraso hau eta gaur egun hainbat mitigazio teknikak sortu dira.

Eragin kritiko bat du zeren eta kernelaren memoria irakurtzeko ahalmena lortzen dugu erabiltzailearen ingurunetik. Memorian dauden pasahitzak, autentifikaziorako tokenak, informazio pribatua, memoriaren fitxategi tenporalak edo sistema eragilearen memorian dauden daturen bat... lor dezakegu baimen nahikorik izan gabe.

Irakurtzeko ez ditu sistema eragilean deiak egiten, orduan, detektatzeko zailagoa da. Syscall bitartez ingurunetaz aldatzea mitigazio teknika bat izan dezake. Gainera, mitigazio teknikak konputagailuaren errendimenduan eragina daukate, ehuneko hamar baino gehiago errendimendu galera sor dezakeela estimatzen da.

4.4. Beharrezko baldintzak

Beharrezko diren baldintzak hauek dira:

- Erasotu ahal den CPU bat
- Mitigazio teknikak kendutak
- Erabiltzailearen eta kernel memoria partekatuta izatea
- Denbora neurtzeko ahalmena izatea
- Segurtasun konfigurazio batzuk kenduta izatea

Hasteko, CPU 2018 baino zaharragoak behar ditugu eta eraso hau Intel,AMD eta ARM prozesadoretan egin da batez ere. Prozesadore hauek, exekuzioa ordenetik kanpo egiteko ahalmena izan behar du eta exekutatzen diren agindu hoiek aislatu gabe izatea eta sortu ahal duten eragina kontuan hartu gabe.

Existitzen dira teknika batzuk eraso hau egitea saihesten dituztenak. Teknika hauek izan daitezke KPTI(Kernel Page Table Isolation)/KAISER, mikrokodean fabrikanteak sortutako partzeak (Microcode patches) edo kernelaren aktualizazioren bat non irakurketa espekulatiboa blokeatzen da.

Meltdown erabiltzailearen ingurunetik kernel memoria mapeatzen saiatuko da nahiz eta permisoa ez izan. Meltdown katxearen bitartez mapeatutako datuak irakurtzen saiatzen da. Errendimendu hobeagoa izateko kernelaren orriak mapeatutak dituzte prozesuen helbide birtualen ingurunean, horrela ez da beharrezkoa ingurune aldaketa oso bat egitea nahiz eta baimenak ez izan. Hau saihesteko, erabiltzailearen eta kernelaren ingurune aldaketa sistema deien bitartez egitea gomendatzen da.

Behin datuak katxean daudenean, denbora erabiltzen dugu jakiteko zein datu dagoen katxean neurtzeko. Orduan, ezinbestekoa da denbora neurtzea zehaztasunez eta horretarako rdtscp eta rdtsc funtzioak erabili ditzakegu.

Distribuzio eguneratu batean exekutatzen bada hainbat segurtasun konfigurazio desaktibatu behar dira erabiltzailearen ingurunetik zehaztasunez neurtzeko ahalmena izateko, katxea irakurtzeko ahalmena izateko (existitzen dira algoritmoak karga susmagarriak detektatzeko).

5. Arkitektura berriak

Alde batetik, exekuzio espekulatiboa kodearen hurrengo agindua zein izango den asmatzen saiatuko da,adibidez if/else batean.

Bestalde, Out-of Order Execution kodearen hurrengo agindua exekutatzen hasiko da hurrengo bukatu den ala ez jakin gabe.

Hauek, errendimendua hobetzen dute zeren eta itxarote denborak murrizten dituzte baino apostua txarra bada, egindako lana ezabatu behar da. Ezabatutako datuak txarto ezabatzen badira ala aztarnak geratzen baldin badira segurtasun arazoak sor ditzakete.

Prozesadoretan ikusi ditugu hainbat baliabide inplementatu dira memoriara ez sartzeko exekuzio espekulatiboaren bitartez. Gainera, arkitektura batzukin ezin da kerneralren orrietara sartzea erabiltzaile baten ingurunetik.

- Kernel Page Table Isolation(KPTI)
- Errendimendu kostea
- Hardwarean orientatutako soluzioak
- Sistema txertatuetarako eta legacy-etarako inplikazioak

6. Kodearen analisia

Hasieran hainbat liburutegi definitu ditugu eta Flush and reload egiteko funtzio bat sortu dugu non movq pointeraren memoria irakurtzen saiatuko da eta rax gordetzen. Hau memoria neurtzeko balio du.

```
1static inline void memoria_sarrera(void *p) {
  asm volatile("movq (%0), %%rax\n" : : "c"(p) : "rax");
}
```

cflush erabili da katxearen dagoen informazioa ezabatzeko.Honekin lortzen duguna da katxean informazioa ez egotea eta memoriara jutea behartzea aginduari. Hau interferentziak edo katxean agertu ahal diren datuak ez molestatzeko balio du. Horrela gero detektatu ahal dugu ea irakurri duguna katxean dagoen konfiantza tarte handiago batekin.

```
static void garbitu(void *p) {
  asm volatile("clflush 0(%0)\n" : : "c"(p) : "rax");

}
```

Tsx funtzioak erabili ditugu (Transactional Synchronization Extensions). Honek transakzioan erroreren bat gertatzen bada, memoria egoera originalera bueltatuko du. Gainera, memoria irakurtzeko espekulaziorako gure alde jolasten du.

```
static __attribute__((always_inline)) inline unsigned int xbegin(void) {
  unsigned egoera;
  asm volatile(".byte 0xc7,0xf8,0x00,0x00,0x00,0x00" : "=a"(egoera) : "a"(-1UL) : "mem
  return egoera;
}

static __attribute__((always_inline)) inline void xend(void) {
  asm volatile(".byte 0x0f; .byte 0x01; .byte 0xd5" ::: "memory");
}
```

Salbuespen tratamendua nahiko garrantzitzua da, kodea exekutatzen jarraitzeko eta programa ez gelditzeko.

Desblokeatu seinalea SIGSEGV seinalearen salbuespena kontrolatzen du zeren eta saiatzen ari gara datu bat irakurtzen kernel memoriatik erabiltzailearen ingurunetik, hau errorea bueltatuko du.

Bestalde segmentation fault detektatzen bada, adibidez memoria helbide bat irakurtzen saiatzen ari gara eta zati hori ez da baliozkoa.

```
static void desblokeatu_seinalea(int sig_zenbakia __attribute__((__unused__))) {
  sigset_t seinaleak;
  sigemptyset(&seinaleak);
  sigaddset(&seinaleak, sig_zenbakia);
  sigprocmask(SIG_UNBLOCK, &seinaleak, NULL);
}
static void segfault_kudeatzailea(int sig_zenbakia) {
  (void)sig_zenbakia;
  desblokeatu_seinalea(SIGSEGV);
  longjmp(buf, 1);
}
```

7. Sistema errealetarako plantamendua

7.1. Planifikazioa

Jakin behar dugu zer erasotzen ari garen eta zer segurtasun maila duen makina honek. Horretarako uname -i, uname -a, cat /proc/info... komandoak erabilgarriak dira.

Sistema eragilea eta distribuzioa zein den, erabiltzailearen baimenak jakitea, segurtasun konfigurazioak dituzten(adibidez KASLR,KPTI, ASLR, NX...).

Azkenean, makinaren analisi bat egiteko. Gero eta zehatzagoa izanda hobeto.

7.2. Erasoa

Erasoa egiten da, bertan jarraipen eta kontrol bat egiten da, non denboraren kontrola eramaten da, ateratako emaitzak aztertzen dira...

7.3. Arazo posibleak eta haien mitigazioa

KSSLR aktibatuta baldin badago arazo bat sor dezake zeren eta kernel memoriaren helbideak aleatorioki gertatzen dira, hau prozesuari sailtasun maila bat gehitzen dio. Honi aurre egiteko, baimenak baldin baditugu desaktibatu edo brute force bidez lortzea memoria irakurtzea.

Stack Canaries buffer overflow egitea sahieztu dezakete eta horren aurre egiteko Jump-Oriented Programing erabiltzen da.

NX kodearen exekuzioa debekatzen du memoria exekutagarria ez diren tarteetan. Horretarako Return Oriented Programing erabiltzen da.

Linuxen AppArmor-ek erasoa blokeatu dezakete eta horren aurre egiteko sistema eragilearen ahultasunak bilatzen lortzen da.

8. Ondorioak

Praktika honek erakutsi digu nola funtzionatzen duten meltdown motako arazoak. Erabiltzaile ingurune batetik nola kernelean dagoen memoria mapeatuta irakurri dezakegu.

Honek arrisku kritikoak sor dezakete zeren eta informazio pribatu asko filtratu dezake, adibidez pasahitzak edo fitxategiak.

Baita ikusi ditugu mitigazio teknikak eta haien eragina.

Praktikaren bidez ikasitakoak sistema errealetan ahuleziak identifikatzeko eta horiei aurre egiteko balioko du, baita programazio-praktika seguruak aplikatzeko ere. Babes-neurriak ezartzea eta pribilegioak behar bezala kudeatzea oinarrizko urratsak dira sistema seguruak izateko.

9. Github repositorioa

https://github.com/ikerSandoval003/MeltdownErasoa

10. Bibliografia

- Ehu.eus
- Meltdown Attack Website
- Meltdown Attack Blog 1
- Meltdown Attack Blog 2
- Meltdown Spectre GitHub - kianenigma
- Meltdown GitHub Script - heartever