Format

- Welcome
- Discuss Confidentiality
- Issue Clearing (if required)
- Meaningful Update & Follow-Up
- Review Next Papers
- Presentation
- Feedback

REVelry
Branch History Injection

Cross-Privilege Spectre-v2 Attacks & Mitigations

About Me | Jay Warne

Currently

- DARPA research
  - Side-channels
  - Processors
  - Hypervisors
  - Rowhammer Style Attacks
  - Program Analysis Methods
- Project Work
  - TL for Enclosed Training Environment
  - PO for Videographic Data Analysis
- Advisory
  - Product Development
  - Product Direction
  - Expert Reviewer

Previously

- Ran a Security Ops Team
- Occasional RE & Red Team
- Field Forensic Analysis & Tool Deployment

Things I Like

- Alpine Ski Racing
- Ski/ Alpine Mountaineering
- Ice/ Rock Climbing
- Surfing
- Backpacking

Branch History Injection (BHI) – tl;dr

- Branch Target Injection (BTI) aka Spectre-v2 exploits branch mispredictions
- Hardware and software mitigations exist, but to limited benefit
- BHI works with latest hardware mitigations in place, leaking kernel from userland

PoC Code Available At:

http://vusec.net/projects/bhi-spectre-bhb

What We Will Cover

- Related Concepts
  - Branch Prediction
  - Branch Target Injection (BTI)
  - uArch Leaking & Disclosure Gadgets
  - Transient Execution
- Mitigations for BTI
  - Retpoline (Intel/ AMD)
  - Workaround_1 (ARM)
  - IBPB
  - STIBP
  - eIBRS
  - FEAT_CSV2
  - CET-IBT
- Branch History Injection
  - Defeating Countermeasures
  - Attack Surface
  - Requirements
- Exploitation
  - Primitives
  - Exploitation w/ eBPF
  - Exploitation w/o eBPF
- Results

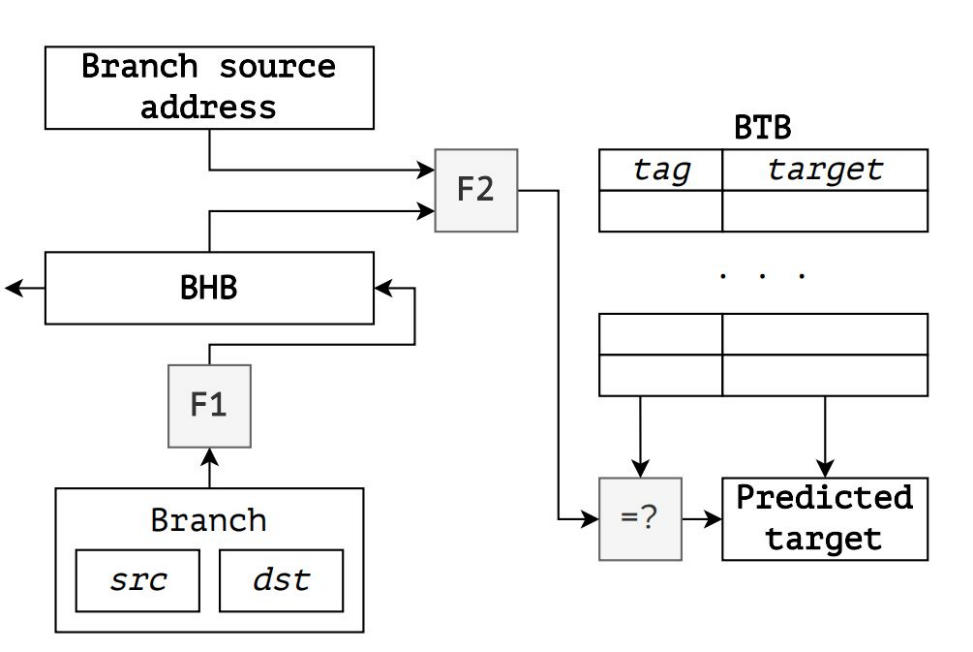
Related Concepts

Related Concepts

- Branch Prediction
- Branch Target Injection (BTI)
- uArch Leaking & Disclosure Gadgets
- Transient Execution

- Branch Prediction Unit (BPU)
- Predicts the targets of branches based on previous behavior to seculatively execute on upcoming instructions
- Direct Branches
- Indirect Branches

Related Concepts



- Branch Prediction Unit (BPU)
- Predicts the targets of branches based on previous behavior to seculatively execute on upcoming instructions
- Direct Branches
- Indirect Branches

Related Concepts

- Branch Prediction
- Branch Target Injection (BTI)
- uArch Leaking & Disclosure Gadgets
- Transient Execution

- Was one of the Spectre gadgets released
- Processors try to predict targets
- Mistrained processor speculatively fetches the wrong thing
- If the processor selects the wrong branch, we can arbitrarily execute code on the victim

Related Concepts

- Branch Prediction
- Branch Target Injection (BTI)
- uArch Leaking & Disclosure Gadgets
- Transient Execution

- This work is predicated in part on knowledge of side channel leaks and disclose
- “Fallout” toy example

Snippets

```
volatile char* lut = (volatile char *)mmap(NULL, STRIDE*256, PROT_READ|PROT_WRITE, MAP_SHARED|MAP_ANONYMOUS,
volatile char* v_pg = (volatile char *)mmap(NULL, PAGE_SIZE*50, PROT_READ|PROT_WRITE, MAP_SHARED|MAP_ANONYMO
volatile char* a_pg = NULL; //(volatile char *)mmap(NULL, PAGE_SIZE, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -
if (v_pg == MAP_FAILED || lut == MAP_FAILED || a_pg == MAP_FAILED) {
  perror("MMAP ERROR BOIS");
  return 1;
}
mprotect((char *)a_pg, PAGE_SIZE, PROT_NONE);
memset((void *)v_pg, 1, PAGE_SIZE*50);
memset((void *)lut, 4, STRIDE*256);

warmup_memory();
warmup_memory();
warmup_memory();
warmup_memory();
```

Snippets

```
clflush(&v_pg[offset]);

for (int i = 0; i < table_entries; i++) {
  clflush(lut + STRIDE * i);
  mfence();
}
```

```
cpuid(0);
unsigned status = _xbegin();
if(_XBEGIN_STARTED == status) {
  for (int x = 0; x < 45; x++) {
    v_pg[offset + x * STRIDE] = 42;
  }
  *(lut + STRIDE * a_pg[offset]);
  //*(lut + STRIDE * 5);
  _xend();
}
```

```
for (int i = 0; i < table_entries; ++i) {
  struct results & res = rarray[i];
  uint32_t aux_val = 0;
  mfence();
  cpuid(0);
  uint64_t perf = read_perf(0);
  uint64_t time = rdtscp(aux_val);
  lut[i * STRIDE];
  mfence();
  //cpuid(0);
  uint64_t time2 = rdtscp(aux_val);
  uint64_t perf2 = read_perf(0);
  res.dt = time2 - time;
  res.dp = perf2 - perf;
}
```

Mitigations for BTI

Mitigations for BTI

- Retpoline (Intel/ AMD)
- Workaround_1 (ARM)
- IBPB
- STIBP
- eIBRS
- FEAT_CSV2
- CET-IBT

Mitigations for BTI

- Retpoline (Intel/ AMD)
- Workaround_1 (ARM)
- IBPB
- STIBP
- eIBRS
- FEAT_CSV2
- CET-IBT

- Software Mitigation
- Code sequence that converts indirect branches to `ret` instructions (intel) or `lfence; jmp` sequences (AMD)
- Intel ensures use of the Return Stack Buffer (RSB) instead of BTB
- AMD ensures that the load retires before the jump, reducing the transient execution window
- Anything that would mispeculate just loops until the address is resolved
- No indirect branch prediction is performed

Mitigations for BTI

- Retpoline (Intel/ AMD)
- Workaround_1 (ARM)
- IBPB
- STIBP
- eIBRS
- FEAT_CSV2
- CET-IBT

- Software Mitigation
- Attempts to invalidate branch predictor entries by
  - executing `BPIALL` on mode switch
  - Disabling and re-enabling the MMU

Mitigations for BTI

- Retpoline (Intel/ AMD)
- Workaround_1 (ARM)
- IBPB
- STIBP
- eIBRS
- FEAT_CSV2
- CET-IBT

- Hardware Mitigation
- Indirect Branch Prediction Barrier
- Previous branches don’t affect the execution of subsequent branches

Mitigations for BTI

- Retpoline (Intel/ AMD)
- Workaround_1 (ARM)
- IBPB
- STIBP
- eIBRS
- FEAT_CSV2
- CET-IBT

- Hardware Mitigation
- Single Thread Indirect Branch Predictors
- Restrict sharing of branch prediction state across hyper-thread and cores

Mitigations for BTI

- Retpoline (Intel/ AMD)
- Workaround_1 (ARM)
- IBPB
- STIBP
- eIBRS
- FEAT_CSV2
- CET-IBT

- Hardware Mitigation
- Indirect Branch Restricted Speculation
- Allegedly prevents indirect branch prediction from being controlled by lower-privileged parts

Mitigations for BTI

- Retpoline (Intel/ AMD)
- Workaround_1 (ARM)
- IBPB
- STIBP
- eIBRS
- FEAT_CSV2
- CET-IBT

- Hardware Mitigation
- ARMs version of: Indirect Branch Restricted Speculation
- Supports two different solutions:
  - Prevent speculative control of indirect branch branch targets from different contexts
  - Enforce the above guarantees for the software context

Mitigations for BTI

- Retpoline (Intel/ AMD)
- Workaround_1 (ARM)
- IBPB
- STIBP
- eIBRS
- FEAT_CSV2
- CET-IBT

- Hardware Mitigation
- Control-Flow Enforcement Technology – Indirect Branch Tracking
- Prevents speculative and architectural execution of all indirect branch targets that don’t start with `ENDBR32/64`

Branch History Injection

Defeating Countermeasures

- Author’s thought was that indirect branch prediction is complex, so corners may have been cut to not lose performance
- Syscall

Defeating Countermeasures

- Author's thought was that indirect branch prediction is complex, so corners may have been cut to not lose performance
- Syscall

Algorithm 1 Experiment to verify BHB isolation between user and kernel mode. For the training phase, $n = 1$ is usually sufficient on the tested CPUs in Table 2.

```
1:  for i ← 1, n do                          ▷ Training
2:      set_history(H_load)
3:      syscall → load(mem)                  ▷ H_load → load(mem)
4:      set_history(H_dummy)
5:      syscall → dummy                      ▷ H_dummy → dummy
6:  end for
7:  flush(mem)                               ▷ Remove mem from cache
8:  set_history(H_load)
9:  syscall → dummy                          ▷ Misprediction to load(mem)?
10: reload(mem)
```

Defeating Countermeasures

- Author’s thought was that indirect branch prediction is complex, so corners may have been cut to not lose performance
- Syscall
- Success proves that the BHB is not isolated
- BHI can overcome defenses to divert execution, but **only to valid kernel targets**
- Their testing showed success on hypervisors as well

Attack Surface

- Out-of-place BTI is possible since BTB collisions can be created only by controlling BHB
- Out-of-place BTI not possible on ARM
- In-place BTI appears to be possible everywhere

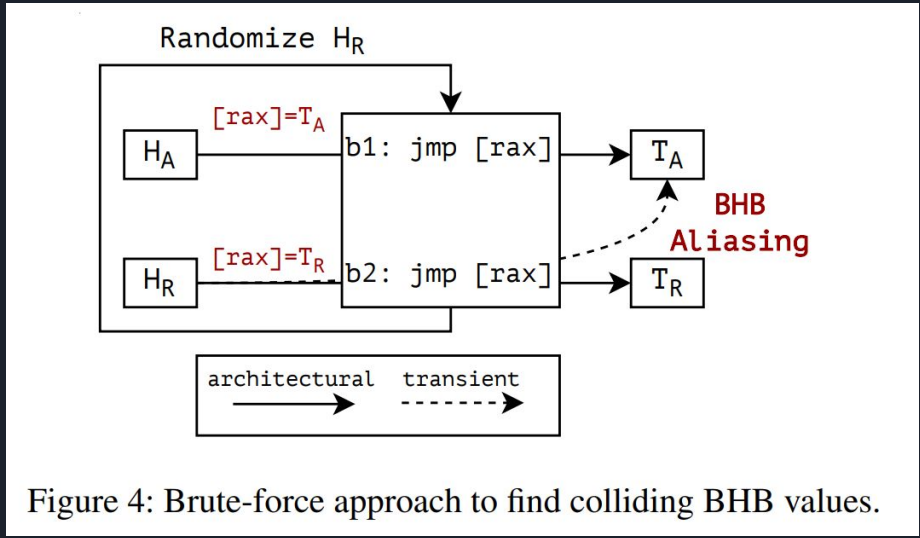


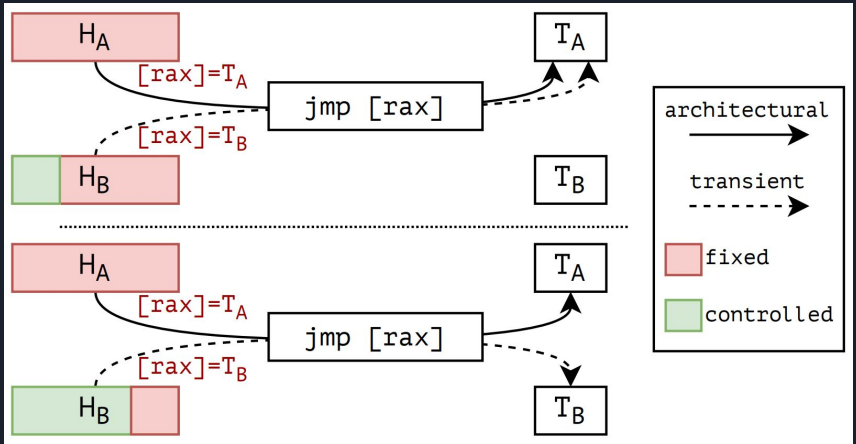
Figure 4: Brute-force approach to find colliding BHB values.

Requirements

- BHB Size
- History Brute-forcing
- History Controllability

Requirements

- BHB Size
- History Brute-forcing
- History Controllability



- Knowing the size informs how many indirect branches needed to cause mis-speculation
- Test for size loop
  - Given: Two colliding histories: Ha, Hb
  - Step 1: Alter oldest branch of Hb
  - Step 2: Test if Hb causes misprediction
  - Exit: When enough old branches of Hb have been altered to no longer cause misprediction

Requirements

- BHB Size
- History Brute-forcing
- History Controllability

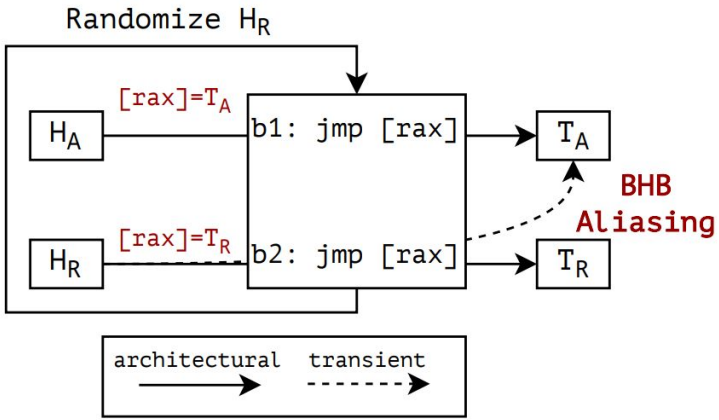


Figure 4: Brute-force approach to find colliding BHB values.

- Must be able to generate matching BTB tags to cause mispeculation
- Brute Force Method
  - Given: Fixed History and Target: Ha, Ta
  - Step 1: Generate Random History (Hr) ending at some Random Target (Tr)
  - Step 2: Test if Hr causes misprediction to Ta
  - Exit: When Hr causes BTB tag to match Ta and cause mispectulation

Requirements

- BHB Size
- History Brute-forcing
- History Controllability

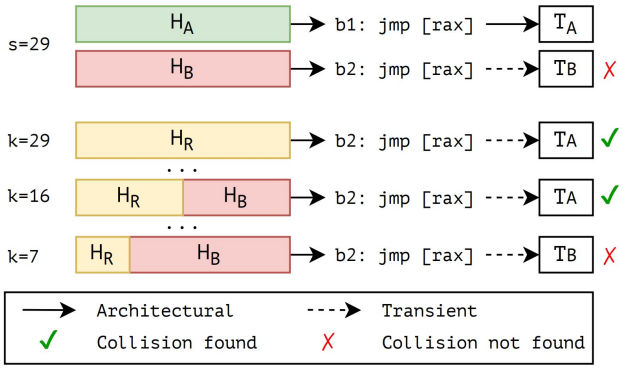


Figure 7: Minimum history control recovery for the Intel 10700K. Starting with two non-colliding histories (H_A and H_B), we aim to discover the minimum number of controlled branches in H_B to preserve collisions.

- How many branches must be controlled to generate arbitrary BTB tags
- Using Hr from History Brute-forcing
- Fix branches closest to Target jump leading to misprediction to a known bad history, Hb until misprediction fails

Exploitation

Full Attack

1. Disclosure Gadget (F+R) from eBPF
2. TAG Generated by Disclosure Gadget is kernel-tagged
3. Execute Jump Chain from ‘Requirements’ to cause BHB collision before syscall
4. Tag collision with disclosure gadget instead of syscall handler
5. “Transient Type Confusion” Gadget leaks into r12, recovered by F+R covert channel

Listing 2 JIT’ted code for the eBPF program serving as disclosure gadget.

```
1   push   rbp
2   mov    rbp,rsp
3   ;load er_buf base address
4   movabs rsi,0xffffc900028ff110
5   ;rdi+0x18 = &pt_regs.r12 transiently
6   ;         = &bpf_sock architecturally
7   mov    rax,QWORD PTR [rdi+0x18]
8   test   rax,rax
9   je     fail
10  ;Dereference of user r12 value transiently
11  mov    eax,DWORD PTR [rax+0x14]
12  ;extract the byte to leak
13  and    rax,0xff
14  shl    rax,0xc
15  add    rsi,rax
16  ;maccess(er_buf[byte_to_leak*0x1000])
17  mov    rsi,QWORD PTR [rsi+0x0]
18 fail:
19  xor    eax,eax
20  leave
21  ret
```