Architecture Security Advisory ASA

Version: 1.0

INCEPTION: Speculative Branch Type Confusion and Transient Training

Non-Confidential

Architecture Security Advisory

INCEPTION: Speculative Branch Type Confusion and Transient Training

Web Address

http://www.arm.com

Contact

arm-security@arm.com

Contents

1 Introduction.

Arm is aware of a research paper called INCEPTION that describes a new transient execution attack that can bypass software Branch Target Injection (BTI) countermeasures in CPUs of another architecture.

By combining multiple techniques and microarchitectural behaviors, an unprivileged/user-space adversary can poison the return predictor of a fully patched system and redirect the privileged/kernel speculative execution to an adversary-controlled location containing an exfiltration gadget.

The impact of a successful INCEPTION attack is similar to Spectre v2 [9]: breach of confidentiality, the kernel virtual address (VA) space can be read by an unprivileged adversary. However, the likelihood of this threat is significantly lower due to its requirements and complexity, therefore, Arm considers the overall risk of INCEPTION lower than Spectre v2.

1.1 Branch Target Injection (BTI) attacks.

We define Branch Target Injection (BTI) attacks as those where a hardware defined context (e.g., a process at EL0) poisons a branch predictor to force another context (e.g., the kernel at EL1) to mis-speculate into an adversary-controlled location.

Spectre v2 [1] was the first instance of this class and manipulated the Branch Target Buffer (BTB) to poison the targets of indirect branches. After that, similar attacks [2] were devised to manipulate the Return Stack Buffer (RSB) and control the destination of return instructions.

The recommended mitigation is the isolation of predictions across contexts. This solution has already been adopted by Arm in newer CPUs implementing FEAT_CSV2. In vulnerable CPUs without built-in protection, software took two different approaches to mitigate Spectre v2:

Retpoline: replace indirect branches in the kernel by a special code sequence that ensures that no code is speculatively executed on that branch.

Flush the BTB: invalidate or overwrite all predictions on context switch.

For variants involving the RSB, the general approach is “stuffing” the predictor on context switch by executing enough return instructions to overwrite any previous potentially-adversary-controlled return address.

Retbleed [3] demonstrates that BTI is still possible despite retpoline. The key observation is that when the RSB is empty, what naturally occurs on return from a deep-enough call path, the return predictor relies on the BTB. This means that by poisoning the BTB entry of a target victim return, the adversary can force the victim to mis-speculate despite no indirect-branch being involved and the RSB having been stuffed on context switch.

Arm has never recommended retpoline as a solution against Spectre v2, and thus Arm processors are not affected [4] by Retbleed.

Jmp2ret [6] was proposed as a Retbleed mitigation and involved replacing all returns in the kernel with direct jumps to a single return function. This reduces the attack surface and allows the kernel itself to mis-train the fallback BTB entry of a single return on a context switch, effectively removing the adversary control.

1.2 The INCEPTION attack.

INCEPTION is another form of BTI attack that, due to some interesting microarchitectural corner cases, bypasses both retpoline and jmp2ret mitigations. This section describes this attack on a vulnerable CPU.

One of the key elements of INCEPTION are PhantomJmps [6]. Branch prediction occurs early in the fetch stage. Before the decode stage, the CPU has almost no information about the instruction entering the pipeline, the fetch unit tries to predict the next instruction exclusively based on the current Program Counter (PC), and optionally the branch history. Therefore, it is possible that a non-branch instruction is mis-predicted to branch somewhere simply because the corresponding BTB entry is filled.

PhantomJmps could completely defeat retpoline as the mitigation only protects indirect branches and the adversary could poison the BTB to force any instruction, not only indirect branches, to branch anywhere. Fortunately, the speculation window is short enough to prevent the execution of leak gadgets containing two data-dependent loads, because the CPU can re-steer the control flow at decode stage, rather than delaying it until the execution stage.

In a way, PhantomJmps are the opposite of Straight-Line Speculation [5]; where a branch instruction that misses the BTB is mis-predicted as a non-branch instruction and fallbacks into the sequential path, i.e., fetch PC+4.

PhantomJmps have been only demonstrated in non-Arm architectures and only to perform a single load. This means that only data already present in the register file can be leaked via a gadget executed with a PhantomJmp, limiting its impact.

The INCEPTION attack describes another interesting behavior, the pushing of return addresses into the RSB occurs during the fetch stage and not during execution. This implies that the CPU relies on a prediction of whether the current PC is a call instruction or not, and therefore it is possible to inject PhantomCalls: any instruction mis-predicted as a call will push PC+4 into the RSB. Furthermore, PhantomCalls can happen during the short speculation window created by a PhantomJmp, even if the short speculation window prevents the PhantomCall from reaching the execution stage.

Now, with all the ingredients for the INCEPTION attack, suppose a victim in context B (e.g., kernel in EL1) executes some function at 0x100, the victim also has an exfiltration gadget at 0x204, as shown in Table 1.

Table 1. Victim Code.

```
0x100: nop // any instruction
…
0x110: ldp x29, x30, [sp], #16
0x114: b return_thunk
…
return_thunk: ret
```

```
0x200: add x2, x3, x4
0x204: ldr x1, [x0]
0x208: ldr x1, [x1]
```

The adversary in context A (e.g., user-space) needs to poison the BTB as described[1] in Table 2:

Table 2. BTB poisoned state left by the adversary.

```
0x100 is a br to 0x200
0x200 is a bl to 0x200
```

[1] The victim and adversary addresses do not necessarily need to be the same, they just need to be aliased/collide.

On context-switch, the BTB is left as it is except for the “return_thunk” entry used by jmp2ret, which is of no interest and thus not showing in Table 1.

When the victim fetches 0x100, the fetch unit will PhantomJmp into 0x200, which will then cause a recursive PhantomCall into 0x200, repeatedly pushing 0x204 into the RSB.

Note that this happens regardless of RSB stuffing on context switch, the victim context itself is acting as a confused deputy and filling the RSB with exfiltration gadget addresses.

Eventually, the execution is re-steered to properly execute 0x100 and continue through 0x104, but when the next RET instruction is found the fetch unit will pop the prediction from the RSB and speculatively return to the exfiltration gadget at 0x204.

This time the speculation window created by the ret instruction will be a long one, not just a PhantomJmp, and will be resolved at execution stage when all the data-dependencies (e.g., the address coming from the stack) are ready. If at that point the adversary happens to control register x0, the exfiltration gadget will be speculatively executed, and leakage will occur.

2 Are Arm cores affected by INCEPTION?

Arm cores implementing FEAT_CSV2 are not affected by INCEPTION or any[2] of the previously discussed BTI attacks.

On cores without FEAT_CSV2, mitigations against Spectre v2 involve flushing all branch predictors (via an implementation specific route) on every context switch, this mitigates INCEPTION and other BTI attacks.

2.1 Speculative Branch Type Confusion.

Speculative Branch Type Confusion (SBTC) causes an instruction to be incorrectly predicted as a branch of a certain type (e.g., direct, indirect, return, etc.). When the instruction is not even a branch, the mis-prediction is known as PhantomJmp.

Some Arm implementations prior to Armv8.5 may be susceptible to SBTC, but Arm recommendations against Spectre v2 on these cores effectively remove the adversary control from the predictor and mitigate any exploits relying on SBTC.

In more recent CPUs implementing FEAT_CSV2, without an adversary capable of poisoning the predictor from another context the risk of exploitation is very low.

Note that in a same-context attack, unless SCXTNUM is used, the indirect branches would be already unprotected and SBTC would only marginally increase the risk.

Given all this, Arm does not consider it necessary to add any new architectural restrictions on speculation.

2.2 Training in Transient Execution (TTE).

The INCEPTION research paper also describes different scenarios where it is possible to train a predictor during speculation. This behavior is well understood and until very recently has been considered appropriate, since the cost of buffering and rolling back microarchitectural changes under a transient branch are high.

The INCEPTION attack relies on TTE to force the victim context to mis-train the RSB during speculation, but without PhantomJmps/Calls, the benefit of TTE itself is unclear. We acknowledge that chaining "training gadgets" with leak gadgets can enable more complex speculative execution attacks that exploit otherwise unexploitable code paths, but the risk remains similar to speculative-type-confusion (which abuse the still unprotected directional predictor). Furthermore, in most cases where the duration of the speculation window is not a limitation, the adversary can directly trigger an exfiltration gadget instead of a training gadget.

Most Arm out-of-order implementations are not susceptible to TTE-RSB. However, Arm considers the risk of exploitation acceptable, and thus implementations should evaluate their own cost vs. security tradeoffs.

2.2.1 TTE Branch History.

An interesting question arises around Spectre-BHI [7] also known as Spectre-BHB. On affected cores, Arm recommendation [8] is to execute a series of branches on context switch to clear the branch history and remove

[2] Spectre-BHB [8] could count as an exception, although it involves target re-use rather than target injection.

control from the adversary, in a similar fashion as RSB stuffing works. Cores with FEAT_CLRBHB add an instruction to clear the history on context switch.

A natural question is whether it would be possible to coerce the victim to execute a sequence of conditional branches in a way that brings the branch history into the desired state.

Theoretically, because the directional predictor is not protected by FEAT_CSV2, an adversary could indeed try to poison the directional predictor from another context. However, the adversary first needs to find a reachable training gadget with enough "poisonable" branches to control the branch history. Note that in Arm, an adversary is not able to leverage PhantomJmps to inject the branch history training gadget, but even if it were, it is unlikely that the phantom speculation window would last long enough.

Given the high complexity of this attack, the high sensibility of the branch history, the uncertainty about the existence of such training gadgets, and the fact that Spectre-BHB only leads to speculative target reuse, compared to arbitrary BTI, we do not consider this a practical threat.

3 Recommendations.

We do not consider the attack or techniques described in the INCEPTION paper a security issue for Arm cores implementing FEAT_CSV2.

For other cores, following the existing recommendations [9] against BTI (a.k.a. variant 2) attacks is sufficient to mitigate INCEPTION.

4 References.

1. "Spectre Attacks: Exploiting Speculative Execution". P. Kocher et al. (2018)
2. "ret2spec: Speculative Execution Using Return Stack Buffers". Giorgi Maisuradze and Christian Rossow. (2018)
3. "RETBLEED: Arbitrary Speculative Code Execution with Return Instructions". Johannes Wikner and Raveh Razavi. (2022)
4. "Are Arm CPUs affected by the RETBLEED side-channel vulnerability?". https://developer.arm.com/documentation/ka005138/1-0/
5. "Straight-line speculation FAQ". https://developer.arm.com/documentation/102587/0102/Straight-line-speculation-frequently-asked-questions
6. "Technical Guidance for mitigating Branch Type Confusion". https://www.amd.com/system/files/documents/technical-guidance-for-mitigating-branch-type-confusion.pdf
7. "Branch History Injection: On the Effectiveness of Hardware Mitigations Against Cross-Privilege Spectre-v2 Attacks". E. Barberis et al. (2022)
8. "Spectre BHB". https://developer.arm.com/Arm%20Security%20Center/Spectre-BHB
9. "Speculative Processor Vulnerability". https://developer.arm.com/Arm%20Security%20Center/Speculative%20Processor%20Vulnerability
10. Arm Security Center: https://developer.arm.com/Arm%20Security%20Center