White Paper | # TECHNICAL GUIDANCE FOR MITIGATING BRANCH TYPE CONFUSION

REVISION 1.0 2022-07-12

This white paper is a technical explanation of what the discussed technology has been designed to accomplish. The actual technology or feature(s) in the resultant products may differ or may not meet these aspirations. Each description of the technology must be interpreted as a goal that AMD strived to achieve and not interpreted to mean that any such performance is guaranteed to be fully achieved. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated.

1. INTRODUCTION

Branch Type Confusion (BTC) is a speculative side channel attack in which the type of branch predicted by the CPU branch predictor does not match the actual instruction bytes. In some cases, this can lead to speculative execution at a target chosen by an attacker, potentially leading to information disclosure. Existing mitigations for indirect branch speculation, such as for Spectre v2, are not sufficient to mitigate all BTC cases and additional mitigations may be required.

This paper explains how AMD has analyzed the BTC vulnerability and the conditions that can cause BTC. It also details various techniques that AMD has developed to help mitigate the different forms of BTC. AMD encourages developers to review this guidance and determine the appropriate mitigations for their environments.

Not all AMD processors are affected by BTC. For a detailed list of affected processors and available mitigations, please see Appendix: Table of Affected Processors at the end of this document.

2. BTC EXPLANATION

The Branch Target Buffer (BTB) is a microarchitectural structure in AMD CPUs that assists with branch prediction. The BTB is indexed by a combination of bits from the instruction pointer (RIP) and holds information about the location, type, and in some cases target of branches. The BTB is accessed by hardware at the beginning of the CPU pipeline while instruction bytes are being read. The information from the BTB, along with the instruction bytes, are then passed to the decode stage of the pipeline.

As noted, the BTB information includes prediction information about branches that may or may not be located in the instruction bytes read. During the decode stage, this information is compared against the actual instruction bytes. For various reasons it is possible that the prediction information provided to the decode stage may be incorrect. For instance, the BTB may predict that a branch exists at a certain byte offset in the cacheline, but in fact that offset is in the middle of another instruction. This scenario primarily occurs due to aliasing in the BTB, where multiple cachelines map to the same BTB entry, but can also arise due to conditions like self-modifying code.

BTC arises from the specific case where the BTB correctly predicts the location of an instruction within the cacheline but incorrectly predicts the presence or type of branch. For example, the BTB may predict that an indirect jump exists at a certain location, but in reality, that location contains a conditional branch.

On affected AMD processors, speculation at the target predicted by the BTB may occur even if the branch type or presence was predicted incorrectly. When an attacker is able to control the target of speculation, along with sufficient registers, they may be able to create a side channel which leaks sensitive data similar to Spectre v2 (CVE-2017-5715).

3. BTC ANALYSIS

To help classify the different types of BTC, we divide the actual and predicted branch types into four groups as shown in Table 1.

BTC occurs when there is a mismatch between the predicted and actual branch groups, as shown in Table 2. Note that in the case of a direct branch, the BTB may correctly predict the type of branch but incorrectly predict the target (e.g., predict a Jcc is taken but to an address that is not the actual branch target). For simplicity, this case is also tracked as a type of BTC.

GROUP	DESCRIPTION
No Branch	Instructions never predicted as taken branches. Includes non-branch instructions (e.g., ADD) as well as far branches (e.g., CALL far) which are never predicted as taken
Direct	RIP-relative branches (Jcc, JMP near, CALL near)
Indirect	Indirect branches (JMP indirect, CALL indirect)
RET	RET instructions (RET, RET imm)

TABLE 1: BRANCH TYPE GROUPS

ACTUAL INSTRUCTION	PREDICTION	RESULT (ON FAMILY 17h[1])
No Branch	Direct	Early Redirect
	Indirect	Early Redirect
	RET	Early Redirect
Direct	No Branch	Early Redirect
	Direct (wrong target)	Early Redirect
	Indirect	Early Redirect
	RET	Early Redirect
Indirect	No Branch	Late Redirect
	Direct	Late Redirect
	RET	Late Redirect
RET	No Branch	Late Redirect
	Direct	Late Redirect
	Indirect	Late Redirect

TABLE 2: BTC CASES

As shown in the table, on affected processors the pipeline behavior depends exclusively on the actual instruction bytes. If the actual instruction is not a branch or is a direct branch, an "early redirect" occurs where the decode unit signals a pipeline flush and execution is redirected at the correct target. If the actual instruction was an indirect branch or return, a "late redirect" occurs where the branch must execute in the ALU before the pipeline is flushed.

The amount of speculation that may occur at the incorrectly predicted target is dependent on the type of redirect that occurs. In the late redirect case, the processor will not flush the pipeline until the branch executes. This means that anything which delays the execution of the branch, such as a cache miss to determine the branch target, can extend the speculation window allowing a larger number of instructions to execute speculatively.

In the early redirect case, the speculation is more limited as the pipeline is quickly flushed when the branch type mismatch is detected. On affected CPUs, some speculation at the predicted target may however, still be observed even after an early redirect.

Based on AMD analysis to date, the speculation that may occur at the predicted target in the early redirect cases is limited and two dependent loads (e.g., a load-load gadget) will not be able to execute before the pipeline is flushed. A single load however, may be observed. As a result, a possible disclosure gadget is limited to values already present in the CPU registers at the time of the prediction.

[1] For further details about different processor generations, see Table of Affected Processors p.11

In the rest of this paper, we refer to the different BTC scenarios based on the actual instruction bytes. BTC-NOBR refers to branch type confusion on an instruction that is in the "no branch" group. BTC-DIR, BTC-IND, and BTC-RET refer to the branch type confusion cases when the instruction is a direct, indirect, or return instruction respectively.

CVE-2022-29900 has been assigned for BTC-RET, and CVE-2022-23825 for the other three cases.

4. SCOPE OF BTC

The speculation that occurs due to BTC is constrained to the active address space and is subject to standard page table and permissions checks as described in [1]. Existing speculation mitigations that involve restricting the address space and/or isolating workloads in unique process contexts remain effective for BTC.

Due to these restrictions, AMD believes that BTC is primarily a concern for operating systems and hypervisors, as well as software which implements "sandboxing". For example, AMD believes that if exploited, BTC may be able to be used to leak kernel data to user-space, or hypervisor data to a guest virtual machine (VM). AMD does not believe that BTC is exploitable for inter-process or inter-VM scenarios in most up-to-date operating systems or hypervisors. As discussed later in this paper, flushing the BTB structure with the Indirect Branch Prediction Barrier (IBPB) operation mitigates all forms of BTC and this operation is typically performed when switching between user processes or VMs.

5. EXISTING MITIGATIONS

Although BTC may occur in all the cases described in Table 2, operating systems and hypervisors typically implement a variety of existing protections for other speculation related vulnerabilities. Many of those existing mitigations eliminate BTC cases too.

AMD's recommended mitigations for CVE-2017-5715 (Spectre v2) include the use of the Indirect Branch Restricted Speculation (IBRS) mode or the use of 'retpoline' (mitigations V2-1 and V2-4 in [2]). On processors affected by BTC, IBRS prevents speculation at the predicted target of any instruction that is decoded as an indirect branch, regardless of the predicted branch type. In contrast, retpoline works by eliminating all uses of indirect branches. Either of these effectively mitigate all cases of BTC-IND.

AMD has also previously recommended mitigations for transient speculation that may occur following an indirect branch, return, or following a direct branch (CVE-2021-26341) as described in mitigation G-5 in [2]. These mitigations eliminate so-called "straight-line speculation" (SLS) that occurs when these branches are not predicted by the BTB, meaning the predicted branch type is "No Branch". We refer to these mitigations as "SLS Protection".

Finally, branches that are predicted as 'ret' instructions get their predicted targets from the Return Address Predictor (RAP). AMD recommends software use a RAP stuffing sequence (mitigation V2-3 in [2]) and/or Supervisor Mode Execution Protection (SMEP) to ensure that the addresses in the RAP are safe for speculation. Collectively, we refer to these mitigations as "RAP Protection".

Applying these existing mitigations reduces the number of BTC cases as shown in Table 3.

ACTUAL INSTRUCTION	PREDICTION	RESULT (ON FAMILY 17h)
No Branch	Direct	Early Redirect
	Indirect	Early Redirect
	RET	Safe (RAP Protection)
Direct	No Branch	Safe (SLS Protection)
	Direct (wrong target)	Early Redirect
	Indirect	Early Redirect
	RET	Safe (RAP Protection)
Indirect	No Branch	Safe (IBRS/Retpoline)
	Direct	Safe (IBRS/Retpoline)
	RET	Safe (IBRS/Retpoline)
RET	No Branch	Safe (SLS Protection)
	Direct	Late Redirect
	Indirect	Late Redirect

TABLE 3: BTC CASES AFTER EXISTING MITIGATIONS

6. NEW MITIGATIONS

This section describes new mitigations that AMD has developed for various BTC cases. Not all mitigations may be available on every affected processor. Please refer to Appendix: Table of Affected Processors for details.

Some of the mitigations included in this document may have effects on system performance when enabled. AMD recommends that customers evaluate the risk profile of their workloads to determine which mitigations are most appropriate for their environments.

6.1 Mitigations for BTC-RET

6.1.1 JMP2RET

Jmp2Ret is a software-based mitigation that mitigates BTC-RET by ensuring that an attacker-controlled BTB entry is never used for predicting privileged 'ret' instructions. It consists of two primary elements. First, all 'ret' instructions are consolidated into a single piece of code. Instead of functions ending with a 'ret' instruction, they instead end with "jmp __ x86_return_thunk". Second, upon entry into privileged code, software safely trains the BTB entry for __x86_return_thunk so attacker-controlled prediction information is not used.

Consolidating function returns into a single thunk location may be assisted through the use of the GCC/Clang "-mfunction-return=thunk-extern" option. It is important to note that all 'ret' instructions that may be reachable at runtime should be consolidated to the thunk. This includes the 'ret' instruction in the retpoline sequence (if used), as well as any 'ret' instructions emitted dynamically by privileged code, such as eBPF.

The second element of jmp2ret is the safe training of the thunk code upon kernel/hypervisor entry. AMD recommends the training sequence as shown in Figure 1 that works for 32/64-bit mode on affected processors. Upon kernel/ hypervisor entry, software should execute 'call __ x86_return_ thunk_train' to safely train the thunk code.

```
.align 64
.fill 63, 1, 0xcc
__x86_return_thunk_train:
   .byte 0x3d
_x86_return_thunk:
   ret
   lfence
   lfence
   jmp _x86_return_thunk
   int3
```

FIGURE 1: JMP2RET TRAINING CODE

Training the thunk involves executing code in such a way that first deletes any prediction information associated with the 'ret' at __x86_return_thunk and then retraining it as the correct branch type. This execution flow of training is shown in blue in Figure 2 while the execution of the normal thunk is shown in red. During training the 0xc3 byte ('ret') is not used as a branch but is part of a compare instruction whose result is ignored. Because of how the 0xc3 byte is positioned in the middle of this instruction, any BTB information associated with that byte is discarded without being used. After an 'lfence' the code then jumps back to __x86_return_thunk where the 0xc3 byte is located. The execution of this 0xc3 results in the BTB being correctly trained that a 'ret' instruction is present at this location. Additionally, the alignment of the thunk as prescribed ensures that subsequent execution of the thunk always uses the same BTB entry so clearing the information associated with that single BTB entry is sufficient for BTC-RET protection.

Note that the training code shown here is an example and developers may, at their discretion, implement a similar and functionally equivalent sequence that maintains the same properties. In particular, software should ensure that during training the 0xc3 byte is interpreted in the middle of a larger instruction.

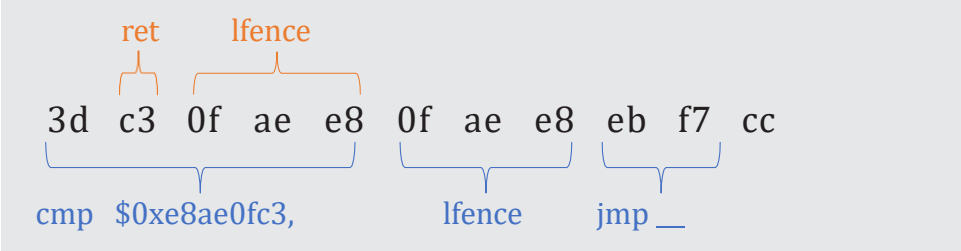

FIGURE 2: TRAINING FLOW

SMT Safety
When the Single Thread Indirect Branch Predictor (STIBP) bit in MSR 0x48 (SPEC_CTRL) is 0, BTB entries are shared between both SMT threads. If the code on the sibling thread cannot be trusted, software should set STIBP to 1 or disable SMT to ensure SMT safety when using Jmp2Ret. If software chooses to toggle STIBP (e.g., set STIBP on kernel entry, and clear it on kernel exit), software should set STIBP to 1 before executing the return thunk training sequence.

6.1.2 IBPB On Privileged Mode Entry

As noted earlier, on AMD processors affected by BTC, the IBPB operation (WRMSR PRED_CMD[0]) flushes all BTB branch prediction information. Software may choose to perform an IBPB command on entry into privileged code in order to avoid any previous branch prediction information from subsequently being used. This effectively mitigates all forms of BTC for scenarios like user-to-supervisor or VM-to-hypervisor attacks.

Developers should note that the IBPB flush operation is rather time consuming (~10k cycles on AMD CPUs codenamed "Zen" and ~2.5k cycles on AMD CPUs codenamed "Zen 2") and is liable to remove both correct as well as potentially malicious branch information. As such, developers may first wish to consider the Jmp2Ret mitigation if possible.

SMT Safety
Similar to the Jmp2Ret mitigation, if the code on the sibling thread cannot be trusted, software should set STIBP to 1 or disable SMT to ensure SMT safety when using this mitigation. If software chooses to toggle STIBP on entry/exit to privileged mode, it should set STIBP on entry prior to issuing the IBPB command.

6.2 Mitigations for BTC-NOBR & BTC-DIR

6.2.1 MSR BIT

AMD "Zen 2" CPUs support a configuration bit in MSR C001_10E3 (DE_CFG2) which changes the behavior of the decode block when the processor attempts to predict a branch on a non-branch instruction (BTC-NOBR case). When bit 1 (SuppressBPOnNonBr) is set the branch prediction information on non-branch instructions is ignored and no speculation at the predicted target will be observed. Setting this bit mitigates the risk of potential information disclosure as a result of speculation in the BTC-NOBR case.

Some systems with recent microcode updates installed may already have this MSR bit set to 1. For details, see the Appendix. In general, AMD recommends keeping systems up-to-date and installing the latest available microcode. In cases where this is not possible, software may directly set DE_CFG2[1] to mitigate BTC-NOBR.

AMD believes the performance impact of this behavior to be negligible and recommends that software set this bit on all supported CPUs to help mitigate the BTC-NOBR case. On CPUs that do not support this bit, software may mitigate BTC-NOBR using the other techniques described in this paper.

6.2.2 IBPB ON PRIVILEGED MODE ENTRY

The technique described earlier of performing an IBPB on entry into privileged mode entry also effectively mitigates BTC-NOBR and BTC-DIR. Please refer to that section for further details.

7. DEFENSE-IN-DEPTH MEASURES

If software is not able to, or chooses not to apply the mitigations described in the previous section, the techniques described in this section may still help mitigate and/or reduce the risk of some cases of BTC, making these type of attacks more difficult to exploit. Many of these measures have benefits beyond the scope of speculation vulnerabilities as well.

7.1 Register Clearing Before RET

In July 2020, the GCC compiler added support for -fzero-call-used-regs which is a compiler option that generates code to clear certain General Purpose Registers (GPRs) before the return from a function. Versions 5.15 and later of the Linux kernel may be configured to be built with the 'used-gpr' option. This option clears the value of all call-used used registers at the end each function. Code compiled with this option is more difficult to exploit with BTC-RET. BTC-RET requires that an attacker control sufficient GPR values at the time of the 'ret' instruction so that the attacker can control the memory values accessed during speculation. For extra protection, the 'all-gpr' option clears additional registers before 'ret' instructions, even if they are not used in that subroutine.

This mitigation is considered a defense-in-depth measure since it does not mitigate all possible BTC-RET exploitation points, however it may make a BTC-RET attack more difficult.

7.2 FGKASLR

Function Granular Kernel Address Space Layout Randomization (FGKASLR) is a security counter-measure that has been under discussion with the Linux kernel community since early 2020. As of this paper, the most recent version of FGKASLR is v10 from February 2022, but the feature has not yet been accepted into upstream Linux.

FGKASLR implements additional layout randomization in the kernel which may make it harder for attackers to find disclosure gadgets. Exploiting any form of BTC requires that an attacker find the address of a gadget in privileged address space using the desired registers and which performs the memory accesses necessary to read and/or leak the contents of memory. Making it harder to find the address of such a gadget, as FGKASLR does, makes many attacks including all forms of BTC more difficult to exploit.

7.3 Half-v1 Protection

As the speculation window associated with BTC-NOBR and BTC-DIR is limited, AMD believes these vulnerabilities are of greatest concern when combined with other speculation related behavior. For instance, BTC may be able to be combined with a single out-of-bounds memory load to potentially create a universal read gadget. An example of this is shown in Figure 3. In this example, an attacker-controlled value is subject to a bounds check. If the 'jae' in this example is mispredicted, the processor may perform an arbitrary memory read into RAX during speculation. An attacker could then potentially use BTC-DIR to generate speculation to another gadget (not shown) which accesses an attacker-visible array based on the value in RAX. The code shown is referred to as a 'half v1' gadget because it requires speculation due to a conditional branch (like in Spectre v1) but only a single memory load exists in the code path.

```
; Assume RCX is
; attacker-controlled
cmp $limit, %rcx
jae bad_value
movq (%rdx, %rcx), %rax
jmp cont
```

FIGURE 3: UNMITIGATED HALF-V1 GADGET

AMD has previously [2] recommended various techniques for mitigating Spectre v1 including the use of 'lfence' to prevent unwanted conditional branch speculation, the use of 'cmov' to limit bad-path speculation, or array masking to prevent out-of-bounds memory accesses. When code sequences like the one shown are mitigated with any of the recommended Spectre v1 mitigations, they can no longer be combined with BTC to create a universal read gadget. Many existing operating systems and hypervisors have applied these techniques already, and AMD recommends that developers inspect their code for any unmitigated cases of 'half v1' gadgets in order to limit the risk when combined with BTC-NOBR or BTC-DIR.

8. DETECTING BTC

AMD has defined a new CPUID bit Fn8000_0008 EBX[29] (BTC_NO) which when set, indicates the processor is not affected by branch type confusion. Specifically, a processor that is not affected by branch type confusion will not speculatively execute operations at a predicted branch target unless one of the following conditions is met:

- The instruction is an indirect branch (JMP reg/mem, CALL reg/mem), excluding far branches
- The instruction is a direct unconditional branch (JMP rel, CALL rel) and the predicted target matches the target encoded in the branch instruction
- The instruction is a direct conditional branch (Jcc), and the predicted target matches either the target encoded in the branch instruction or the straight-line (not-taken) path of the branch
- The instruction is a return (RET), and the predicted target was supplied by the return address predictor

AMD recommends that software, especially virtualized software, use this CPUID bit to detect whether BTC mitigations are required.

Note that AMD Family 19h CPUs are not vulnerable to BTC but do not set this CPUID bit. Bare-metal software that detects a Family 19h CPU should assume the CPU is not vulnerable to BTC and BTC mitigations are not needed. Hypervisor software should synthesize the value of the BTC_NO CPUID bit on such platforms as desired so guest software can rely on using CPUID to detect if BTC mitigations are required.

9. CONCLUSION

This whitepaper has presented several potential mitigations for various BTC cases on affected AMD processors, as well as defense-in-depth measures that may help reduce the risk of BTC in cases where full mitigations may be impractical to deploy. Table 4 summarizes the mitigation and defense-in-depth measures applicable to each variant of BTC. BTC-IND is not included since AMD believes that existing mitigations for CVE-2017-5715 effectively mitigate BTC-IND. Please consult the Table of Affected Processors in the Appendix for details on the availability of various mitigations on specific processors.

AMD recommends that software developers review the provided mitigation options and evaluate the best mitigations that align with the requirements of their deployments and risk profile.

	BTC-NOBR	BTC-DIR	BTC-RET
MITIGATIONS			
Jmp2Ret	—	—	X
Indirect Branch Predictor Barrier (IBPB)	X	X	X
DE_CFG2[1] [SuppressBPOnNonBr]	X	—	—
DEFENSE-IN-DEPTH MEASURES			
GPR Clearing before RET	—	—	X
FGKASLR	X	X	X
Half-v1 Protection	X	X	—

TABLE 4: SUMMARY OF BTC MITIGATIONS

10. REFERENCES:

[1] AMD, "Speculation Behavior in AMD Micro-Architectures," 14 5 2019. [Online]. Available: https://www.amd.com/system/files/documents/security-whitepaper.pdf.

[2] AMD, "Software Techniques for Managing Speculation on AMD Processors," 8 3 2022. [Online]. Available: https://www.amd.com/system/files/documents/software-techniques-for-managing-speculation.pdf.

APPENDIX: TABLE OF AFFECTED PROCESSORS

	Fam 15h[2] Models 00-7Fh	Fam 17h Models 00h-2Fh Models 50h-5Fh	Fam 17h Models 30h-4Fh Models 60h-7Fh	Fam 19h All Models
CPU MICRO-ARCHITECTURE	"Bulldozer"	"Zen"/"Zen+"	"Zen 2"	"Zen 3"
Affected by				
BTC-NOBR	X	X	X	_[3]
BTC-DIR	X[4]	X	X	_[3]
BTC-IND	X	X	X	_[3]
BTC-RET	X	X	X	_[3]
Available Mitigations				
Jmp2Ret	X[5]	X[6]	X	-
Indirect Branch Predictor Barrier (IBPB)	X	X	X	-
DE_CFG2[1]			X[7]	-
Applicable Defense-in-Depth Measures				
GPR Clearing before RET	X	X	X	-
FGKASLR	X	X	X	-
Half-v1 Protection	X	X	X	-

At this time, AMD is not providing statements about the susceptibility of, or mitigations for BTC on processors older than the ones listed in this table.

REVISION 1.0 22/07/12

AMD.com/productsecurity

AMD

[2] As of the date of initial publication of this whitepaper, please note that Family 15h architectures are not supported by AMD PSIRT (product security incident response team)

[3] The AMD "Zen 3" microarchitecture is not vulnerable to any form of BTC and no BTC mitigations are required for these processors

[4] AMD "Bulldozer" processors may use late redirects in BTC-DIR cases, resulting in a larger speculation window

[5] AMD "Bulldozer" processors do not support STIBP

[6] AMD "Zen" and "Zen+" processors do not support STIBP

[7] This bit is set automatically if the following microcode (or newer) is installed:

Family/Model/Stepping	Code Name	Microcode Version
Fam 17h, Model 31h, Stepping 0h	"Rome" / "Castle Peak"	08301055h
Fam 17h, Model 60h, Stepping 1h	"Renoir"	08600109h
Fam 17h, Model 68h, Stepping 1h	"Lucienne"	08608104h
Fam 17h, Model 71h, Stepping 0h	"Matisse"	08701030h