

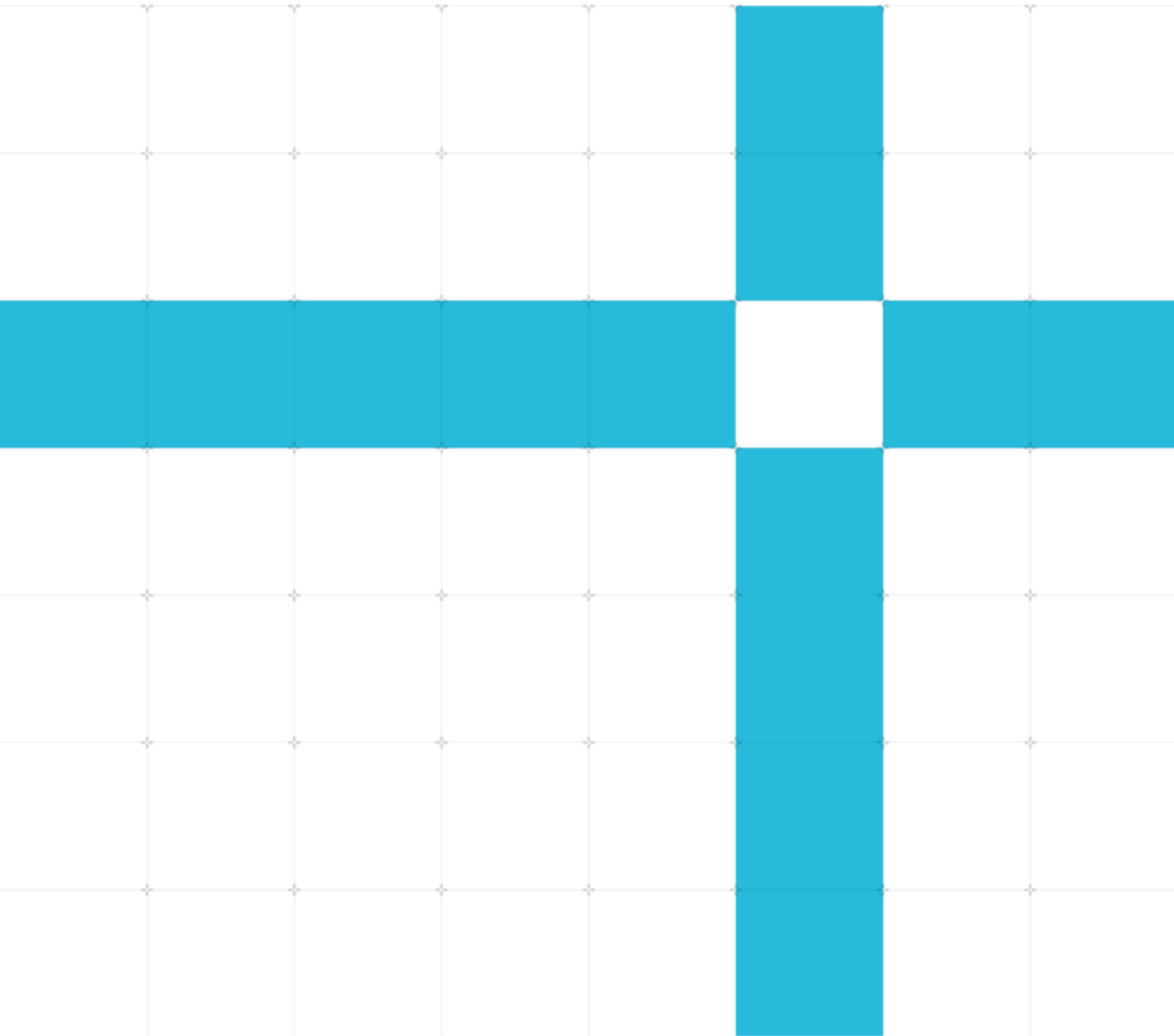


# Architecture Security Advisory ASA

Version: 1.0

## Prefetcher Side Channels

Non-Confidential  
Copyright © 2023 Arm Limited (or its affiliates).  
All rights reserved.



# Architecture Security Advisory

## Prefetcher Side Channels

Copyright © 2023 Arm Limited (or its affiliates). All rights reserved.

### Arm Non-Confidential Document Licence (“Licence”)

This Licence is a legal agreement between you and Arm Limited (“**Arm**”) for the use of Arm’s intellectual property (including, without limitation, any copyright) embodied in the document accompanying this Licence (“**Document**”). Arm licenses its intellectual property in the Document to you on condition that you agree to the terms of this Licence. By using or copying the Document you indicate that you agree to be bound by the terms of this Licence.

“**Subsidiary**” means any company the majority of whose voting shares is now or hereafter owner or controlled, directly or indirectly, by you. A company shall be a Subsidiary only for the period during which such control exists.

This Document is **NON-CONFIDENTIAL** and any use by you and your Subsidiaries (“**Licensee**”) is subject to the terms of this Licence between you and Arm.

Subject to the terms and conditions of this Licence, Arm hereby grants to Licensee under the intellectual property in the Document owned or controlled by Arm, a non-exclusive, non-transferable, non-sub-licensable, royalty-free, worldwide licence to:

- (i) use and copy the Document for the purpose of designing and having designed products that comply with the Document;
- (ii) manufacture and have manufactured products which have been created under the licence granted in (i) above; and
- (iii) sell, supply and distribute products which have been created under the licence granted in (i) above.

**Licensee hereby agrees that the licences granted above shall not extend to any portion or function of a product that is not itself compliant with part of the Document.**

Except as expressly licensed above, Licensee acquires no right, title or interest in any Arm technology or any intellectual property embodied therein.

THE DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. Arm may make changes to the Document at any time and without notice. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

NOTWITHSTANDING ANYTHING TO THE CONTRARY CONTAINED IN THIS LICENCE, TO THE FULLEST EXTENT PERMITTED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, IN CONTRACT, TORT OR OTHERWISE, IN CONNECTION WITH THE SUBJECT MATTER OF THIS LICENCE (INCLUDING WITHOUT LIMITATION) (I) LICENSEE’S USE OF THE

DOCUMENT; AND (II) THE IMPLEMENTATION OF THE DOCUMENT IN ANY PRODUCT CREATED BY LICENSEE UNDER THIS LICENCE). THE EXISTENCE OF MORE THAN ONE CLAIM OR SUIT WILL NOT ENLARGE OR EXTEND THE LIMIT. LICENSEE RELEASES ARM FROM ALL OBLIGATIONS, LIABILITY, CLAIMS OR DEMANDS IN EXCESS OF THIS LIMITATION.

This Licence shall remain in force until terminated by Licensee or by Arm. Without prejudice to any of its other rights, if Licensee is in breach of any of the terms and conditions of this Licence then Arm may terminate this Licence immediately upon giving written notice to Licensee. Licensee may terminate this Licence at any time. Upon termination of this Licence by Licensee or by Arm, Licensee shall stop using the Document and destroy all copies of the Document in its possession. Upon termination of this Licence, all terms shall survive except for the licence grants.

Any breach of this Licence by a Subsidiary shall entitle Arm to terminate this Licence as if you were the party in breach. Any termination of this Licence shall be effective in respect of all Subsidiaries. Any rights granted to any Subsidiary hereunder shall automatically terminate upon such Subsidiary ceasing to be a Subsidiary.

The Document consists solely of commercial items. Licensee shall be responsible for ensuring that any use, duplication or disclosure of the Document complies fully with any relevant export laws and regulations to assure that the Document or any portion thereof is not exported, directly or indirectly, in violation of such export laws.

This Licence may be translated into other languages for convenience, and Licensee agrees that if there is any conflict between the English version of this Licence and any translation, the terms of the English version of this Licence shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. No licence, express, implied or otherwise, is granted to Licensee under this Licence, to use the Arm trade marks in connection with the Document or any products based thereon. Visit Arm's website at <http://www.arm.com/company/policies/trademarks> for more information about Arm's trademarks.

The validity, construction and performance of this Licence shall be governed by English Law.

Copyright © 2023 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-21585 version 4.0

## Web Address

<http://www.arm.com>

## Contact

[psirt@arm.com](mailto:psirt@arm.com)

# Contents

**1 Introduction ..... 5**

1.1. Background: classes of prefetchers ..... 5

**2 Are Arm cores affected? ..... 7**

2.1. Advanced prefetchers ..... 8

2.2. R and M profiles ..... 8

**3 The attack ..... 9**

3.1. Cross-context side channel ..... 9

3.2. TrustZone covert channel ..... 9

**4 Mitigations..... 10**

**5 References ..... 11**

# 1 Introduction

Hardware data prefetchers can boost CPU performance by fetching data ahead of the current execution. This is possible by keeping track of memory access patterns performed by software and re-using them to anticipate future accesses. If the prefetcher succeeds in predicting software access pattern, next time the code runs, data will be available in the cache and the access time will be drastically reduced. The complexity of the detectable patterns varies with the type of prefetcher.

Hardware prefetchers are a shared resource and, as such, they can be a security risk if preventive measures, such as flushing during a context transition or tagging by context, are not taken. Security issues of a shared resource include:

- contention, which leads to side channels
- "state pollution", where a context consumes the state created by another context, which can lead to both side channels (**predictive leakage**) or injection attacks (**exploitative control**)

Cache side channels are an example of security issues created by contention. Branch predictors can suffer from both contention and state pollution; both can lead to branch-predictor side channels, but state pollution also introduces higher risk vulnerabilities, like Branch Target Injection (BTI), a.k.a. Spectre v2.

Arm is aware of a state-pollution based side channel attack [1] affecting the data prefetcher on a number of Arm cores. In this attack, an unprivileged adversary manages to observe secret-dependent memory accesses from a victim in another context and use that information to recover part of the key in the Mbed TLS 3.3.0 AES implementation. Note that this is a lookup-table based AES implementation.

Side channels do not directly leak secret data, but metadata, such as memory accesses performed or branches taken by a program. A program can indirectly leak secret data if it performs memory accesses or branches based on that data.

While the overall risk of this particular side-channel is deemed low due to its impact, complexity and all the mitigating factors, this document aims to address the concerns about the potential threat of state pollution attacks in more complex hardware prefetchers. An example of these is memory dependent prefetchers [2], where in a vulnerable implementation, actual data (and not only metadata) could be directly leaked.

The Arm architecture introduced FEAT\_CSV2 to mitigate Spectre v2 attacks and other variants, including those abusing other prediction mechanisms like hardware prefetchers.

## 1.1. Background: classes of prefetchers

Prefetchers adopt different strategies to identify memory accesses patterns. A simple example is a prefetcher accessing the next (or previous) cache line(s) after a memory access: this approach provides a performance boost in case of sequential access patterns, such as reading sequentially from an array.

Prefetchers can also compute the distance or "stride" between two subsequent memory accesses and detect the direction of it: the next time a load is performed, the prefetcher will start performing accesses by extending the pattern and filling the cache lines with the loaded data. However, this category of prefetchers require the stride between memory accesses to be constant, so it doesn't work well with irregular memory accesses.

Stream prefetchers work with irregular patterns, once the direction of the pattern is known, at each access, it prefetches one or more adjacent cache lines.

A different approach is adopted by “temporal prefetchers”, as they internally track sequences of memory accesses instead of trying to extrapolate patterns. As soon as memory accesses start following again the same sequence, the prefetcher can anticipate the future accesses and replay them.

More elaborated prefetching strategies can be based on analysing memory content to detect stored structures that could lead to non-linear memory accesses (for example, array of pointers or linked lists). In this case, the prefetcher first dereferences a pointer and then fetches the value stored in that location. If this is triggered on a location containing a secret rather than a valid pointer, the second data-dependent fetch can leave a footprint observable by a side-channel attacker.

## 2 Are Arm cores affected?

The techniques presented in the FetchBench research paper [1] involve re-using the hardware prefetcher's microarchitectural state of the victim's context in the adversary's context to learn its content and infer information about the memory access pattern of the victim. The attack has been demonstrated on a Cortex-A72 core.

Due to the nature of the affected prefetcher, the impact of this attack is limited to software performing secret-dependent memory accesses. Researchers were able to recover part of the bits of an AES key when targeting a timing side-channel vulnerable Mbed TLS implementation. However, it was already known that this particular implementation is vulnerable to traditional side-channel attacks, as constant time block ciphers are currently out of the scope of the Mbed TLS project [5].

Arm's general policy regarding side-channel attacks is to advise mitigating on the software side by discouraging the use of secret-dependent memory accesses or branches in security sensitive code.

The Arm ARM describes two threats which are two sides of the same coin:

- **Exploitative control of speculative execution.** This involves an adversary context influencing the speculation of a victim context in such a way that the execution leaks some architectural accessible state specified by the adversary. An example is BTI or Spectre v2.
- **Predictive leakage.** Victim's execution influences a predictive microarchitectural structure in such a way that it affects the speculative execution of other contexts, allowing an adversary to recover part of the victim's architectural state. For example, extracting context's branch targets via a side-channel.

The "**Restrictions on the effects of speculation**" B2.3.11 section of the Arm ARM [6] include FEAT\_CSV2 that mentions the following:

*"Code running in one hardware-defined context cannot either exploitatively control, or predictively leak to, the speculative execution of code in a different hardware-defined context, as a result of: [...] virtual address-based cache prefetch predictions [...]"*

We emphasize that FEAT\_CSV2 covers all types of prediction mechanisms, not only branch predictors.

The reported attack in the FetchBench research paper [1] is an example of predictive leakage via data prefetchers, where part of the addresses accessed, and PC are leaked. Cores not implementing FEAT\_CSV2, including the Cortex-A72, are affected.

Arm observed that the FEAT\_CSV2 restrictions about prefetch predictions may be stricter than what actual hardware implements. However, based on Arm's risk analysis:

- Most Arm data prefetchers are not susceptible to "exploitative control", and even if the adversary can influence the prefetching in the victim context, it is not possible to recover any of this context's architectural state.
- The risk of predictive leakage from these data prefetchers is low, as 1) the impact is limited to software performing secret-dependent operations, and 2) the difficulty of performing these techniques compared to other known side-channels is much higher.

Accordingly, Arm has relaxed the restrictions in FEAT\_CSV2 to only cover virtual address-based cache prefetch predictions based on data values from memory, which encompasses the more advanced class of prefetchers with a higher security risk.

## 2.1. Advanced prefetchers

In the last few years, more elaborate prefetching strategies have been proposed. One of them takes advantage of pointer chasing scenarios by monitoring the contents of data memory returned by a prefetch: if the prefetcher believes it is a pointer, it performs a memory access to the location it points to. This approach can be repeated for multiple levels of indirection.

Effectively, such a prefetching mechanism creates a hardware Spectre gadget with a secret-dependent memory access.

A paper on this class of prefetchers has been published [2] proving how, in some implementations, it can be abused to leak memory content that would be inaccessible otherwise.

Arm's implementations of **memory-dependent prefetchers prevent both exploitative control and predictive leakage attacks**, as predictions are only accessible in the context that was generated in.

## 2.2. R and M profiles

Current Cortex-M and Cortex-R processors do not support memory value-dependent prefetching mechanisms. Furthermore, the absence of virtual memory in the M-profile severely limits the scope of threats similar to the techniques described in the paper.

Arm considers the risk of FetchBench attacks on R-profile and M-profile cores as low with no further actions to be taken on current R-profile and M-profile CPUs.



## 3 The attack

Some prefetchers keep track of memory access patterns and perform them again as soon as the pattern starts repeating. Internally, these prefetchers may associate the detected patterns to the Program Counter (or part of it) of the instruction that started the access.

The FetchBench research paper [1] demonstrates two different threats, a side-channel attack from an unprivileged process, and a covert-channel between Secure and Non-Secure world.

### 3.1. Cross-context side channel

Researchers have been able to successfully exfiltrate part of an AES encryption key from a victim process. However, this type of attack presents several limitations:

- The AES implementation must be vulnerable to side-channel timing analysis.
- A fine-grained synchronization must be possible between the victim and the adversary processes.
- The adversary needs to be able to know when the victim is about to execute code that would lead to the pollution of the prefetcher state and interrupt the execution.
- The code handling the key must be in shared memory between the victim and the attacker processes.

Synchronization between the victim and the attacker contexts is required to avoid pollution of the prefetcher internal state with unrelated memory accesses patterns. The way this can be obtained heavily depends on the operating system [4].

This type of synchronization is far from being precise, as task scheduling could let the attacker detect the function execution in the victim process too late; in addition, the IPI may interrupt the victim at the wrong time. This means that spurious samples are likely to be generated and that the adversary must also be able to filter them out.

Exfiltrating data from another process with this technique requires a vast number of attempts, given the very low success rate due to the required synchronization mechanism.

### 3.2. TrustZone covert channel

It is possible to use the prefetcher's internal state to create a covert channel if cache lines are used to encode information.

This requires the two colluded parties to agree on aligning the load instruction address to the same lower bits to trigger the same prefetch training data.

However, even considering the low fail rate in decoding the data crossing the covert channel, building it requires the adversary to be able to register a Trusted Execution Environment (TEE) service before, which is a hard requirement in a secure firmware implementation. Secure services are normally registered only at boot time and being able to inject a new one from the Non-Secure world requires a poor TEE implementation.

## 4 Mitigations

To optionally ensure that all prefetchers are protected, on systems implementing FEAT\_SPECRES it is possible for software in more privilege exception levels to manually flush the prefetcher state whenever there is a context transition by using the CPP RCTX instruction [3].

In case this feature is not supported, the only option left is to disable the impacted prefetcher.

# 5 References

1. "FetchBench: Systematic Identification and Characterization of Proprietary Prefetchers", Till Schlüter, Amit Choudhari, Lorenz Hetterich, Leon Trampert, Hamed Nemat, Ahmad Ibrahim, Michael Schwarz, Christian Rossow and Nils Ole Tippenhauer, CCS 2023
2. "Augury: Using Data Memory-Dependent Prefetchers to Leak Data at Rest", Jose R. S. Vicarte, Michael Flanders, Riccardo Paccagnella, Grant Garrett-Grossman, Adam Morrison, Christopher W. Fletcher and David Kohlbrenner, S&P 2022
3. CPP RCTX, Cache Prefetch Prediction Restriction by Context,  
<https://developer.arm.com/documentation/ddi0601/2020-12/AArch64-Instructions/CPP-RCTX--Cache-Prefetch-Prediction-Restriction-by-Context>
4. "ExpRace: Exploiting Kernel Races through Raising Interrupts", Yoochan Lee, Changwoo Min and Byoungyoung Lee, 2021
5. "Mbed TLS project". <https://github.com/Mbed-TLS/mbedtls/blob/development/SECURITY.md#block-ciphers>
6. Arm Architecture Reference Manual for A-profile architecture,  
<https://developer.arm.com/documentation/ddi0487/latest>