COBHAM

Meltdown, Spectre, and Security Boundaries in LEON/GRLIB

Technical note

Doc. No GRLIB-TN-0014

Issue 1.1

2018-03-15

CHANGE RECORD

Issue	Date	Section / Page	Description
1.0	2018-03-06		First release
1.1	2018-03-15		Fix typos

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Scope of the Document

This document describes how LEON3, LEON3FT, LEON4, and LEON4FT are immune to the Meltdown security vulnerability and the Spectre class of vulnerabilities.

The document also discusses other channels where information can leak in LEON/GRLIB systems.

1.2 Distribution

GRLIB users are free to use the material in this document in their own documents and to redistribute this document. Please contact Cobham Gaisler for inquires on other use.

1.3 Contact

For questions on this document, please contact Cobham Gaisler support at support@gaisler.com. When requesting support include the part name if the question is a specific device or the full GRLIB IP library package name if the question relates to a GRLIB IP library license.

1.4 Finding the build ID of a LEON device or GRLIB source tree

The GRLIB build ID is present in the AMBA plug&play area. The build ID is also reported by the GRMON debug monitor when connecting to the device.

If you have licensed GRLIB for use in your own FPGA or ASIC design, the GRLIB version can be seen in the file name of the downloaded release package, in the directory name after unpacking the release, and in the file lib/grlib/stdlib/version.vhd in the release file tree (constant grlib_build).

1.5 Reference documents

[RD1] Reading privileged memory with a side-channel. https://googleprojectzero.blogspot.se/2018/01/reading-privileged-memory-with-side.html

[RD2] Kocher, P., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M. & Yarom, Y. Spectre Attacks: Exploiting Speculative Execution. https://spectreattack.com/spectre.pdf

[RD3] Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Mangard, S., Kocher, P., Genkin, D., Yarom, Y. & Hamburg, M. Meltdown. https://meltdownattack.com/meltdown.pdf

2 MELTDOWN AND SPECTRE

2.1 Overview

Meltdown and Spectre make use of mis-speculated execution combined with data cache timing analysis to leak information across security boundaries. At the time of writing there are tree known variants [RD1] of the issues that affect major CPU vendors:

1. Bounds check bypass (CVE-2017-5753)
2. Branch target injection (CVE-2017-5715)
3. Rogue data cache load (CVE-2017-5754)

Variants 1 and 2 belong to the Spectre [RD2] class of vulnerabilities and variant 3 is the Meltdown vulnerability [RD3]. The mechanisms of these attacks are described in the research papers [RD2] [RD3].

Meltdown makes use of speculative execution to access restricted memory. It is possible to leak data from restricted memory by having speculatively executed code depend on a data value at a restricted memory location so that the speculatively executed instructions alter L1 cache state at different locations depending on the value at the restricted location. Data cache side channels can then be used to determine which location in L1 cache that was altered and the data value at the restricted location is revealed.

Spectre describes a wider range of attacks that depend on branch prediction and speculative execution. Also with Spectre, the speculative execution resulting from a misprediction leaves side effects that is then used to reveal data from restricted memory locations.

2.2 Meltdown, Spectre, and LEON Processors

LEON3, LEON3FT, LEON4, and LEON4FT processors are immune to the Meltdown and Spectre class of attacks. While the processors make use of branch prediction, the predicted instructions do not alter cache state as is required for the Meltdown and Spectre attacks.

- The LEON data cache is never altered as the result of speculative execution. The speculative instructions do not progress beyond the register file access stage in the processor pipeline.
- Branch prediction is only performed for branch instructions with a fixed PC-relative target and not for any kind of indirect jumps.
- The only cache effect of branch prediction is that the L1 cache may be populated with the branch target instructions in case of a cache miss. This behaviour can be dynamically controlled and fetching instructions on cache miss is disabled by default (functionality added in GRLIB build 4153).

3 POSSIBLE SOURCES OF INFORMATION LEAKS IN LEON/GRLIB SYSTEMS

3.1 Overview

LEON/GRLIB systems are implemented in many vastly different hardware configurations. A large part of the software implementations on these systems do not make use of a memory management unit and execute all code in supervisor mode. In these systems there is no privilege separation and any piece of code can access all parts of memory. Security in these systems is instead attained by testing and trusting all the code that runs on the system. The MMU can be used as a safety net in these systems by setting up a static MMU configuration that ensures that the software does not perform out of bounds accesses. Since software running in supervisor mode can disable the MMU, this measure is an added safety net and not a security precaution.

Other software implementations make use of an operating system that separates user processes from kernel mode by making use of processor privilege modes; user mode and supervisor mode. By using a memory management unit and these modes it is possible to constrain user processes from accessing data belonging to the kernel and other processes. In LEON/GRLIB systems there are ways in which this separation can be circumvented. The subsections following this section describe the main categories:

1. Level-1 cache privilege check bypass
2. Peripherals capable of direct memory access
3. Timing analysis of shared resources

3.2 Level-1 Cache Privilege Checks

LEON3 and LEON4 processors have two privilege modes; user mode and supervisor mode. When operating with the memory management unit (MMU) enabled, pages in memory can be annotated to be accessible from supervisor mode only. This functionality is used to separate user mode processes from the kernel in operating systems. Kernel code is executed in supervisor mode and the user process is executed with user mode privilege.

A user mode process typically also maps in kernel instructions and data in its MMU context. The user mode process is prevented from accessing kernel information by annotating the MMU pages so that the kernel's information is accessible in supervisor mode only. When kernel mode is entered, through for example a system call, then the MMU context remains unchanged and the kernel executes by accessing the pages marked as supervisor only. If a user process tries to access kernel data then this will generate a MMU trap. For this to work with data in the L1 cache, the data in the cache must also be annotated with the allowed access attributes. This is a side-effect of the L1 cache being virtually indexed and tagged. It is possible to look up data in the cache without traversing the MMU page table and finding the privilege information for the MMU page that is being accessed.

It is possible to implement the LEON processors without the supervisor/user information in the L1 cache (in GRLIB before build 4211 this was the only supported configuration). In this case a user mode process read kernel mode data that exists in the L1 cache. Writes to kernel data will be inhibited since a write operation will lead to a MMU lookup to translate the virtual address to the physical address used on the on-chip bus. During the MMU lookup, the mismatch between user mode and supervisor mode will be detected and the write operation will cause a trap. As part of this the data in the L1 cache will be invalidated. To prevent possible leaks of kernel information to a user process the LEON must either be implemented with the “supervisor only” bit added to the L1 cache tags, or the L1 cache needs to be flushed when returning to user space from kernel mode.

Lack of the supervisor annotation in the L1 cache tags does not mean that a user mode process can immediately access an arbitrary location belonging to the kernel. It would however be possible to construct a program that performs a system call and then guesses on kernel addresses that may be in L1 cache. By trying to access these locations from the user mode application, kernel mode data could be obtained for accesses that hit in cache. This is a security implication of the missing privilege check. A functionally correct, non-malicious, application will not depend on the privilege checks to be present.

3.3 Peripherals Capable of Direct Memory Access

Peripherals capable of direct memory access (DMA) include communication controllers such as the GRSPW2 SpaceWire controller and the GRETH_GBIT MAC. Most high-speed communication controllers in GRLIB include their own DMA engines and are controlled through descriptors in main memory.

There peripherals generally have access to the full 32-bit address space. Since a system is generally made up by one operating system controlling all processors and all DMA units, the DMA units are under the control of the kernel. In other variants, different operating systems may be running on a subset of the processors available in the design and in some cases user mode may have access to control the DMA units. In such scenarios it is important to identify that a DMA unit can be configured to access any memory area and thereby bypass the protection offered by the processor MMU.

To avoid security boundaries to be crossed, code that controls DMA units must be trusted and the system can also be implemented to include a IOMMU (IP core GRIOMMU) that performs the same function for the DMA peripherals as the processor MMU provides to software. Note that the processor MMU can be used to prevent user space code from accessing the DMA units.

3.4 Timing Analysis of Shared Resources

A typical LEON/GRLIB system has a significant amount of shared resources. Sharing hardware resources lead to efficient implementations from a hardware perspective and at the same time opens up for a wide range of timing attacks from untrusted software. Examples of shared resources in a typical LEON/GRLIB system include:

- Processor L1 cache
- Level-2 cache
- Memory controller
- Processor AHB bus

If it is possible to extract knowledge of a software's inner working by, for example, monitoring the number of cache misses then another processor connected to the same bus could make a series of accesses to a shared resource and use execution timing to determine if the bus is accessed by another peripheral. A similar analysis can be make by accessing shared regions in the Level-2 cache and use the execution time to determine if an access was a hit or miss in the Level-2 cache. Another effect of sharing data in the Level-2 cache is that superscalar performance benefits can be obtained by one processor populating the Level-2 cache and the data is reused by the other processor. In other words, the functionality that allows timing attacks is the same functionality that improves performance in the general case. Timing attacks need to be mitigated by the applications that need to withstand analysis by constructing the application to be resilient against timing analysis and avoiding to leave traces in shared state.

4 CONCLUSION

- LEON/GRLIB systems are immune from Meltdown and Spectre attacks by design.
- An attack vector to reveal kernel data from a user process exists in systems that have been implemented without the supervisor tag bit.
- Software systems need to be designed with an understanding that peripherals with DMA capabilities are not subject to the separation enforced on software by a processor MMU.
- Shared resources can give rise to timing attacks. This type of attack is enabled by design choices made to provide high performance in the general purpose case and to provide a flexible hardware platform. If untrusted code is run on a system then critical software parts need to implement mitigation measures against timing attacks.