Firmware interfaces for mitigating cache speculation vulnerabilities

System Software on Arm Systems

Version 1.3

Firmware interfaces for mitigating cache speculation vulnerabilities

System Software on Arm Specification

Release Information

Document History

Version/Issue	Date	Confidentiality	Change
1.0	26 January 2018	Non-Confidential	First release
1.1	31 January 2018	Non-Confidential	Included missing details about Cortex-A8
1.2	09 March 2018	Non-Confidential	Extension of SMCCC_ARCH_FEATURES to provide per-PE mitigation discovery
1.3	21 May 2018	Non-Confidential	Added SMCCC_ARCH_WORKAROUND_2 to mitigate CVE-2018-3639. Clarified SMCCC_ARCH_WORKAROUND_1 discovery semantics

Non-Confidential Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Web Address

http://www.arm.com

Contents

1 About this document

CVE-2017-5715, also known as Spectre variant 2, is a vulnerability in some Arm CPU designs that might allow an attacker to control the speculative execution flow within a victim execution context and disclose data that is architecturally inaccessible to the attacker. On affected Arm CPUs the recommended mitigations include invalidation of some or all of the Branch Predictor caches when transitioning to an execution context that requires protection from previous executing contexts. See section 1.2 for a definition of the term execution context.

CVE-2018-3639, also known as variant 4, is a vulnerability in some Arm CPU designs that might allow a speculative read of a memory location to read a data value from before the most recent write to that memory location. The speculatively read data might be attacker-controlled and forwarded to later speculative accesses, which may disclose data that is architecturally inaccessible. On affected Arm CPUs the recommended mitigations include disabling the bypassing of writes by reads (including speculative reads), either permanently during CPU initialization, or dynamically as required by software requiring protection.

For more information on these and related vulnerabilities, please see the material provided at the *Arm Security Update* website [1].

The mechanism by which such software mitigations are implemented is CPU implementation specific, and is not always accessible by software running at EL1 or EL2. This specification defines additional services that should be provided by firmware on systems with affected CPUs, enabling operating system and hypervisor software to apply appropriate workarounds for these vulnerabilities, and to discover the presence of these firmware services.

The interfaces are specified as extensions to the *SMC Calling Convention* (SMCCC) [3] and *Power State Coordination Interface* (PSCI) [4] in order ensure a standard interface for affected CPUs.

The interfaces and recommended usage patterns described in this document will be incorporated into future versions of the SMCCC and PSCI specifications.

Arm Trusted Firmware [5] provides a reference implementation of this functionality, which is enabled in default configurations of this firmware

1.1 References

See the Arm Infocenter, **http://infocenter.arm.com**, and **Arm Developer** for access to Arm documentation.

Reference	Document	Author	Title
1	N/A	Arm	Arm Security Update **https://www.arm.com/security**
2	ARM DDI 0487	Arm	Arm Architecture Reference Manual, Armv8 for Armv8-A architecture profile
3	ARM DEN 0028B	Arm	SMC Calling Conventions (SMCCC)
4	ARM DEN 0022D	Arm	Power State Coordination Interface (PSCI)
5	N/A		Arm Trusted Firmware **https://github.com/Arm-Software/arm-trusted-firmware**
6	N/A	Arm	Arm Developer processor documentation **https://developer.arm.com/products/processors/cortex-a**
7	ARM DEN 0054A	Arm	Software Delegated Exception Interface (SDEI)

1.2 Terms and Abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
AArch64 state	The 64-bit Execution state. In AArch64 state, addresses are held in 64-bit registers, and instructions in the base instruction set can use 64-bit registers for their processing. AArch64 state supports the A64 instruction set.
ACPI	The Advanced Configuration and Power Interface specification. This defines a standard for device configuration and power management by an OS.
CPU	A hardware implementation of the Arm Architecture
EL0	The lowest Exception level. This Exception level is unprivileged. The Exception level used to execute user applications, in Non-secure state.
EL1	Privileged Exception level. The Exception level typically used to execute operating systems.
EL2	Hypervisor Exception level. The Exception level used to execute hypervisor code. EL2 is always in Non-secure state.
EL3	Secure monitor Exception level. This Exception level has the highest privilege and is always in Secure state. If implemented, a PE always reset and commence execution at this Exception level.
Execution context	The PE state associated with a thread of execution, including register state, exception level and security state. Usually an execution context is is managed by another execution context at a higher exception level or an exception level in the secure state, for example firmware manages one or more system software execution contexts. However, the managing and managed execution contexts may reside at the same exception level and security state, for example a runtime environment manages one or more interpreted applications.
FDT	Flattened Device Tree. This is a hardware description methodology. Firmware tables are constructed that describe the hardware. These tables are passed to the OS at boot time. An OS can interrogate the data they contain when it needs to discover the hardware properties of a device.
Firmware	Software that provides platform specific services, typically operating at an exception level higher than the Operating System or Hypervisor which makes use of the firmware services.
Function Identifier	A 32-bit integer, which identifies the function being invoked by this SMC/HVC call. Passed in X0 into every SMC/HVC call.
HVC	Hypervisor Call. An instruction that causes a synchronous exception that is taken to EL2.
Hypervisor	The hypervisor executes at EL2. It supports the execution of multiple EL1 operating systems.
Non-secure state	The Security state that restricts access to only the Non-secure system resources such as memory, peripherals and System registers.
Normal world	The execution environment when the core is in the Non-secure state.
OS	Application operating system such as Linux or Windows. This also includes virtualized OS running under a hypervisor.
PE	The abstract machine defined in the ARM architecture, as documented in an ARM Architecture Reference Manual. A processing element implementation that is compliant with the ARM architecture must conform with the behaviors described in the corresponding ARM Architecture Reference Manual.

Term	Meaning
Secure EL1	The Secure EL1 Exception level, the Exception level used to execute the S-EL1 software in Secure state. The software can be a Secure OS or S-EL1 firmware.
Secure state	The ARM Security state that enables access to the Secure and Non-secure systems resources, such as memory, peripherals and System registers.
SMC	Secure Monitor Call. An instruction that causes a synchronous exception that is taken to EL3.
SoC	System on Chip.
Unknown Function Identifier	A reserved return code defined by SMCCC that indicates that the function is not implemented. It is declared as `NOT_SUPPORTED` in the interface specification and takes the value -1.

1.3 Firmware definition

Implementations of the services described in this specification will be implemented at multiple levels to be available to both host operating systems and hypervisors that run against hardware platforms, and to guest operating systems that run against virtual platforms.

Platform firmware typically runs at EL2 and EL3 to provide services to host Operating Systems and hypervisors. The services described in this specification would be implemented in EL3 for this software.

Virtualized firmware typically runs at EL1 and EL2 to provide services to guest Operating Systems. The services described in this specification would be implemented in EL2 for this software.

Both platform firmware and virtual firmware implementations of these services are referred to as *firmware* in the remainder of this document.

2 Changes to the SMC Calling Convention

2.1 Overview

2.1.1 Optimize SMC/HVC calling convention from AArch64

In order to optimize the execution of SMC and HVC calls from A64 assembly, the register calling convention from AArch64 is updated to reduce the number of caller-save registers from 18 to just 4, which matches the calling convention for AArch32.

This permits firmware operations to be invoked from assembly code without saving and then restoring a large number of GP registers.

The calling convention change is backwards compatible from the perspective of the caller. However, unmodified callers will perform more saving and restoring of registers than is strictly necessary.

Adding this calling convention change to the firmware implementation should not add a significant overhead, as most existing implementations need to correctly handle calls from AArch32, which requires sanitizing GP register state on returning to the caller.

This change in calling convention is mandatory for all SMCCC calls, not just the new functions defined in this specification. Note that it must also apply to all unimplemented SMCCC calls that return the *Unknown Function Identifier* (`NOT_SUPPORTED`) value.

The changes to the calling convention are highlighted in gray in **Table 1** below, which is modified from section 3.1 in the original SMCCC specification [3].

Register Name		Role during SMC or HVC call		
SMC32/HVC32	SMC64/HVC64	Calling values	Modified	Return state
SP_ELx		ELx stack pointer	No	Registers values are preserved
SP_EL0		EL0 stack pointer	No	
X30		The Link Register	No	
X29		The Frame Pointer	No	
X19…X28		Registers that are saved by the called function	No	
X18		The Platform Register	No	
X17		The second intra-procedure-call scratch register	No	Registers values are preserved
X16		The first intra-procedure-call scratch register	No	
X9…X15		Temporary registers	No	
X8		Indirect result location register	No	
W7	W7	Optional Client ID in bits[15:0] (ignored for HVC calls) Optional Secure OS ID in bits[31:16]	No	
W6	X6 (or W6)	Parameter register Optional Session ID register	No	
W4…W5	X4…X5	Parameter registers	No	

Register Name		Role during SMC or HVC call		
SMC32/HVC32	SMC64/HVC64	Calling values	Modified	Return state
W1…W3	X1…X3	Parameter registers	Yes	SMC and HVC Result registers
W0	W0	Function Identifier	Yes	

Table 1 Register Usage in AArch64 SMC32, HVC32, SMC64, and HVC64 calls

2.1.2 SMCCC Version

For system software to identify the implementation of the new calling convention, a SMCCC version number is introduced, and a function to retrieve this value.

The version that introduces the new calling convention and *Arm Architecture Services* is v1.1.

All previous versions of SMCCC specification (revisions A and B) are v1.0.

As the SMCCC Version function is new in this specification, existing firmware implementations that conform to SMCCC are expected to return `NOT_SUPPORTED`. When this value is returned then the caller should treat this as if it had returned v1.0.

Firmware implementations are permitted to implement this function and return v1.0 if they do not provide the new calling convention or feature discovery function.

This function is mandatory if SMCCC is version 1.1 or later.

See section 2.2.2 for a full specification of the SMCCC version function.

2.1.3 SMCCC Arm Architecture Service function discovery

The new SMCCC functions are added to the *Arm Architecture Service* range of fast calls.

For system software to dynamically detect the implementation of the *Arm Architecture Service* functions, a new feature discovery function is specified. This will also enable additional *Arm Architecture Service* functions to be detected when they are added to the firmware implementation in future.

This function is mandatory if SMCCC is version 1.1 or later.

See section 2.2.3 for a full specification of the discovery function.

2.1.4 Workarounds for cache speculation vulnerabilities

These workaround functions should be provided in firmware on systems containing at least one PE affected by CVE-2017-5715 or CVE-2018-3639 with an available firmware workaround (see [1] for details on which Arm CPUs are affected).

See sections 2.2.4 and 2.2.5 for a full specification of the workaround functions.

2.1.5 Deprecation of the General Service Queries for the Arm Architecture Service

The General Service Queries for SMCCC call ranges are described in section 6.2 of the document ARM DEN 0028B SMC Calling Conventions.

These functions are not always well suited to firmware that is integrated with multiple sub-services being combined into one service range. For example, PSCI and SDEI in the *Standard Service* range. In particular, the 'call count' and 'revision' functions do not provide useful information to the caller when multiple functions are provided. As a result, these are not widely used to identify firmware services.

The requirement to implement these functions for the *Arm Architecture Service* is deprecated from version 1.1 of SMCCC, and might be removed in a future version. The function IDs which can be omitted are: 0x8000 FF00, 0x8000 FF01 and 0x800 FF03.

2.2 Interface

2.2.1 Return Codes

Table 2 defines the possible values for error codes used with the interface functions. The error return type is signed integer. Zero and positive values denote success and negative values indicates error.

Name	Description	Value
`SUCCESS`	The call completed successfully.	0
`NOT_SUPPORTED`	Not supported by the implementation.	-1
`NOT_REQUIRED`	The call is deemed not required by the implementation.	-2

Table 2 Return code and values

2.2.2 SMCCC_VERSION

Dependency	**MANDATORY** from SMCCC v1.1 **OPTIONAL** for SMCCC v1.0		
Description	Retrieve the implemented version of the SMC Calling Convention		
Parameters	uint32 Function ID	0x8000 0000	
Return	int32	`NOT_SUPPORTED`	Treat as v1.0
		`major:minor`	• Bit[31] must be zero • Bits [30:16] Major version • Bits [15:0] Minor version

2.2.2.1 Usage

This call is used by system software to determine the version of SMCCC implemented, which indicates the calling convention for AArch64 callers and the presence of the SMCCC_ARCH_FEATURES function.

2.2.2.2 Discovery

This function was not defined in SMCCC version 1.0, and might not be safe on all platforms. Calling software can detect the implementation of this function by one of the following methods:

- Built-in knowledge of the firmware implementation.
- Discovery via PSCI_FEATURES (see section 3.2.1 which defines this function and Appendix A: Discovery of Arm Architecture Service functions for the full discovery procedure).

- Information from firmware tables such as *Flattened Device Tree* or ACPI tables.

See Appendix A: Discovery of Arm Architecture Service functions for a description of the full discovery sequence.

If SMCCC_VERSION is implemented, calling SMCCC_ARCH_FEATURES with `arch_func_id` equal to 0x8000 0000 will return `SUCCESS`.

2.2.2.3 Caller responsibilities

Prior to calling this function, Arm recommends that the caller determines if it is safe to do so on the platform as some firmware implementations do not implement this function safely. Appendix A: Discovery of Arm Architecture Service functions provides the recommended discovery protocol.

The caller must interpret a `NOT_SUPPORTED` response as indicating the presence of firmware implementing SMCCC v1.0.

2.2.2.4 Implementation responsibilities

For firmware that implements the old calling convention for AArch64 callers, this function must return `NOT_SUPPORTED` (-1) or version 1.0 (0x10000).

For firmware that implements the new calling convention for AArch64 callers this function must return version 1.1 (0x10001).

2.2.3 SMCCC_ARCH_FEATURES

Dependency	**MANDATORY** from SMCCC v1.1 **OPTIONAL** for SMCCC v1.0		
Description	Determine the availability and capability of *Arm Architecture Service* functions		
Parameters	uint32 Function ID	0x8000 0001	
	uint32 arch_func_id	Function ID of an *Arm Architecture Service* Function	
Return	int32	`< 0`	Function not implemented or `arch_func_id` not in *Arm Architecture Service* range. The reason is indicated by an error code specific to the function.
		`SUCCESS`	Function implemented
		`> 0`	Optional Function implemented. Function capabilities are indicated using feature flags specific to the function.

2.2.3.1 Usage

This call is used by system software to determine whether a specific *Arm Architecture Service* function is implemented in the firmware. This function might also provide information about the capabilities of the function.

2.2.3.2 Discovery

The implementation of this function can be detected by checking the SMCCC version. This function is mandatory if SMCCC_VERSION indicates that version 1.1 or later is implemented.

See Appendix A: Discovery of Arm Architecture Service functions for the full discovery sequence.

If SMCCC_ARCH_FEATURES is implemented, calling SMCCC_ARCH_FEATURES with `arch_func_id` equal to 0x8000 0001 will return `SUCCESS`.

2.2.3.3 Parameters

The `arch_func_id` parameter is the Function ID in the *Arm Architecture Service* range: 0x8000 0000-0x8000 FFFF and 0xC000 0000-0xC000 FFFF. Values outside of this range are invalid.

2.2.3.4 Return

If the result is non-negative it indicates that the function is implemented.

Some functions provide information about their capabilities in the result.

A description of how to interpret the result of calling SMCCC_ARCH_FEATURES is provided in the *Discovery* section of the documentation for each function.

2.2.3.5 Caller responsibilities

The caller must only call SMCCC_ARCH_FEATURES on implementations compliant to SMCCC version 1.1 or later.

2.2.3.6 Implementation responsibilities

This function must return `NOT_SUPPORTED` if the SMCCC version is lower than version 1.1.

This function must return `SUCCESS` when `arch_func_id` is the SMCCC_VERSION or SMCCC_ARCH_FEATURES function id.

2.2.4 SMCCC_ARCH_WORKAROUND_1

Dependency	**OPTIONAL** from SMCCC v1.1 **NOT SUPPORTED** in SMCCC v1.0	
Description	Execute the mitigation for CVE-2017-5715 on the calling PE	
Parameters	uint32 Function ID	0x8000 8000
Return	void	This function has no return value

2.2.4.1 Usage

This call is used by system software to execute a firmware workaround required to mitigate CVE-2017-5715.

2.2.4.2 Discovery

The implementation of this function can be detected by calling SMCCC_ARCH_FEATURES (see 2.2.3) with `arch_func_id` equal to 0x8000 8000. The result of that call should be interpreted as follows:

`NOT_SUPPORTED`	The discovery call will return `NOT_SUPPORTED` on every PE in the system. SMCCC_ARCH_WORKAROUND_1 must not be invoked on any PE in the system. Either: • None of the PEs in the system require firmware mitigation for CVE-2017-5715, • The system contains at least 1 PE affected by CVE-2017-5715 that has no firmware mitigation available, or • The firmware does not provide any information about whether firmware mitigation is required.
`0`	SMCCC_ARCH_WORKAROUND_1 can be invoked safely on all PEs in the system. The PE on which SMCCC_ARCH_FEATURES is called requires firmware mitigation for CVE-2017-5715.
`1`	SMCCC_ARCH_WORKAROUND_1 can be invoked safely on all PEs in the system.

	The PE on which SMCCC_ARCH_FEATURES is called does not require firmware mitigation for CVE-2017-5715.

2.2.4.3 Caller responsibilities

The caller must not call this function unless it has determined that it is implemented in the firmware, see Discovery above.

Arm recommends the caller only call this on PEs affected by CVE-2017-5715 when a firmware based mitigation is required and a local workaround is infeasible. Calling this on other PEs is wasted execution.

See the *Arm Security Update* [1] and

Appendix B: Mitigating CVE-2017-5715 on Arm CPUs for more information.

2.2.4.4 Implementation responsibilities

This function must not be provided in firmware implementations not compliant to SMCCC version 1.1 or later.

If implemented, the firmware must provide discovery of this function as defined in the Discovery section above.

Arm recommends that firmware does not provide an implementation of this function on systems that return a negative error code in the discovery call above.

If implemented, the firmware must fully implement this function for all PEs in the system requiring firmware mitigation for CVE-2017-5715

In heterogeneous systems with some PEs that require mitigation and others that do not, the firmware must provide a safe implementation of this function on all PEs. This permits callers to call the function on all PEs in a system where the firmware implements the workaround, without risking functional stability. In such systems, on PEs that do not require firmware mitigation, the firmware must provide an implementation that has no effect.

See the *Arm Security Update* [1] and

Appendix B: Mitigating CVE-2017-5715 on Arm CPUs for more information.

2.2.5 SMCCC_ARCH_WORKAROUND_2

Dependency	**OPTIONAL** from SMCCC v1.1 **NOT SUPPORTED** in SMCCC v1.0	
Description	Enable or disable the mitigation for CVE-2018-3639 on the calling PE	
Parameters	uint32 Function ID	0x8000 7FFFF
	uint32 enable	A non-zero value indicates that the mitigation for CVE-2018-3639 must be enabled. A value of zero indicates that it must be disabled.
Return	void	This function has no return value.

2.2.5.1 Usage

This call is used by system software to enable or disable a firmware workaround required to mitigate CVE-2018-3639. The call only affects the mitigation state (enabled or disabled) for the calling execution context. The workaround is enabled by default for all execution contexts managed by the firmware. Once the workaround is disabled, it remains disabled until explicitly re-enabled by a subsequent call to this function.

2.2.5.2 Discovery

The implementation of this function can be detected by calling SMCCC_ARCH_FEATURES (see 2.2.3) with arch_func_id equal to 0x8000 7FFF. The result of that call should be interpreted as follows:

`NOT_SUPPORTED`	The discovery call will return `NOT_SUPPORTED` on every PE in the system. SMCCC_ARCH_WORKAROUND_2 must not be invoked on any PE in the system. Either: • The system contains at least 1 PE affected by CVE-2018-3639 that has no firmware mitigation available, or • The firmware does not provide any information about whether firmware mitigation is required or enabled.
`NOT_REQUIRED`	The discovery call will return NOT_REQUIRED on every PE in the system. SMCCC_ARCH_WORKAROUND_2 must not be invoked on any PE in the system. For all PEs in the system, firmware mitigation for CVE-2018-3639 is either permanently enabled or not required.
`0`	SMCCC_ARCH_WORKAROUND_2 can be invoked safely on all PEs in the system. The PE on which SMCCC_ARCH_FEATURES is called requires dynamic firmware mitigation for CVE-2018-3639 using SMCCC_ARCH_WORKAROUND_2.
`1`	SMCCC_ARCH_WORKAROUND_2 can be invoked safely on all PEs in the system. The PE on which SMCCC_ARCH_FEATURES is called does not require dynamic firmware mitigation for CVE-2018-3639 using SMCCC_ARCH_WORKAROUND_2. Firmware mitigation on this PE is either permanently enabled or not required.

2.2.5.3 Caller responsibilities

The caller must not call this function unless it has determined it is implemented in the firmware, see Discovery above.

Arm recommends the caller only call this on PEs affected by CVE-2018-3639 when a dynamic firmware-based mitigation is required and a local workaround is infeasible. Calling this on other PEs is wasted execution.

Arm recommends that secure world software does not use this call so that it remains protected at all times.

See the Arm Security Update [1] and Appendix C: Mitigating CVE-2018-3639 on Arm CPUs for more information.

2.2.5.4 Implementation responsibilities

This function must not be provided in firmware implementations not compliant to SMCCC version 1.1 or later.

If implemented, the firmware must provide discovery of this function as defined in the Discovery section above.

Arm recommends that firmware does not provide an implementation of this function on systems that return a negative error code in the discovery call above.

If implemented, the firmware must fully implement this function for all PEs in the system requiring dynamic firmware mitigation for CVE-2018-3639.

In heterogeneous systems with some PEs that require dynamic firmware mitigation and others that do not, the firmware must provide a safe implementation of this function on all PEs. This permits callers to call the function on all PEs in a system where the firmware implements the workaround, without risking functional stability. In such systems, on PEs that do not require dynamic firmware mitigation, the firmware must provide an implementation that has no effect.

If implemented, the firmware must separately maintain the logical mitigation state (enabled or disabled) for each execution context it manages. The default mitigation state (enabled) must be applied:

- To the primary PE following cold boot.
- To a PE when it starts up following a `CPU_ON` call, as defined by the PSCI specification [4].
- To a PE when it wakes up from a powerdown state (for example, following a `CPU_SUSPEND` call), as defined by the PSCI specification [4].

If implemented, Arm recommends that the firmware enables mitigation during its own execution.

If the firmware implements this function and the *Software Delegated Exception Interface* (SDEI) specification [7], then the firmware must apply the default mitigation state (enabled) to the execution context of each SDEI client handler following each triggered event, irrespective of the mitigation state of the interrupted or client execution contexts. The firmware must restore the mitigation state of the interrupted or client execution context following a call to `SDEI_EVENT_COMPLETE` or `SDEI_EVENT_COMPLETE_AND_RESUME` respectively.

See the Arm Security Update [1] and Appendix C: Mitigating CVE-2018-3639 on Arm CPUs for more information.

3 Changes to PSCI

3.1 Overview

3.1.1 Discoverability of SMCCC implementation

On platforms that fully implement the SMC Calling Convention as described in revision B of that specification, the above additional functions are adequate for detecting the presence of the *Arm Architecture Service* functionality.

However, some platform and virtualization firmware only implements a subset of SMCCC. Specifically, such firmware might only implement the PSCI specification in order to meet Operating System requirements, but not provide a safe implementation of unimplemented SMCCC functions. A safe implementation is one that conforms to the SMCCC calling convention and returns `NOT_SUPPORTED` (-1) for functions that are not defined or not implemented.

System software that needs to use the mitigation functions described above but must also run correctly on such platforms, requires one or more additional mechanisms to discover whether SMCCC is implemented, and specifically whether it is safe to call SMCCC_VERSION.

One mechanism is to add discoverability of this to the firmware description (e.g. Device Tree or ACPI tables).

To accelerate adoption of these mitigations and protect more systems more rapidly from this vulnerability, Arm strongly recommends the firmware implementations also provide an additional discovery mechanism though the firmware PSCI implementation of PSCI_FEATURES.

3.2 Interface

This is an updated excerpt from the PSCI specification for PSCI_FEATURES. The significant changes are highlighted in gray. Note that the changes will apply from PSCI v1.0.

3.2.1 PSCI_FEATURES

Dependency	Introduced in PSCI v1.0		
Description	Query API that allows discovering whether a specific PSCI function is implemented and its features. See the PSCI specification [4] for more details.		
Parameters	uint32 Function ID	0x8400 000A	
	uint32 psci_func_id	Function ID for a PSCI Function or SMCCC_VERSION	
Return	int32	`NOT_SUPPORTED`	Function identified by `psci_func_id` not is not implemented or not valid
		`SUCCESS`	Function implemented
		`> 0`	Optional A set of feature flags associated with an implemented function indentified by `psci_func_id`. Feature flags are specific to each function. In all cases the format is: • Bit[31] is zero • Bits[0:30] represent the feature flags. See PSCI [4] for details.

3.2.1.1 Usage

As for PSCI v1.0 and later.

In addition, this interface can be used to detect the implementation of SMCCC_VERSION in the firmware.

3.2.1.2 Parameters

The `psci_func_id` parameter is valid if it is any of:

- a PSCI SMC32 function identifier, in the range 0x8400 0000-0x8400 001F
- a PSCI SMC64 function identifier, in the range 0xC400 0000-0xC400 001F
- the SMCCC_VERSION function identifier, 0x8000 0000

3.2.1.3 Return

For valid PSCI function identifiers see PSCI for details of the return value.

When `psci_func_id` is SMCCC_VERSION, a return value of `SUCCESS` (zero) indicates that SMCCC_VERSION is implemented.

3.2.1.4 Implementation responsibilities

Arm recommends that this function reports the presence of SMCCC_VERSION for any firmware that implements SMCCC v1.1 as described in this specification.

4 Appendix A: Discovery of Arm Architecture Service functions

System software needs to run safely on any existing platform, but should make use of the mitigation functionality whenever it is available. The following approach to discovery should maximize the ability to detect this functionality without causing unsafe behavior on existing platforms.

4.1 Step 1: Determine if SMCCC_VERSION is implemented

The following pseudo code summarizes the proposed discovery flow using PSCI 1.0:

```
if (FirmwareTablesLookup(PCSI) == SUCCESS)
{
    if (invoke_pcsi_version() >= 0x10000)
    {
        if (invoke_pcsi_features(SMCCC_VERSION) == SUCCESS)
            return SUCCESS;
    }
}
return NOT_SUPPORTED
```

The steps are:

1. Use firmware data, device tree PSCI node, or ACPI FADT PSCI flag, to determine whether a PSCI implementation is present.
2. Use PSCI_VERSION to ascertain whether PSCI_FEATURES is provided. PSCI_FEATURES is mandatory from version 1.0 of PSCI.
3. Use PSCI_FEATURES with the SMCCC_VERSION function identifier as a parameter to determine that SMCCC_VERSION is implemented.

In future ACPI and device tree might also be extended to indicate the compliance to the SMCCC directly.

4.2 Step 2: Determine if Arm Architecture Service function is implemented

The following pseudo code summarizes the proposed discovery flow of an *Arm Architecture Service* function, using SMCCC_ARCH_WORKAROUND_1 as an example:

```
if (invoke_smccc_version() >= 0x10001)
{
    if (invoke_smccc_arch_features(SMCCC_ARCH_WORKAROUND_1) >= 0)
        return SUCCESS;
}
return NOT_SUPPORTED
```

The steps are:

1. Use SMCCC_VERSION to ascertain that the calling convention complies to version 1.1 or above.
2. Use SMCCC_ARCH_FEATURES to query whether the *Arm Architecture Service* function is implemented on this system.

If the software is running on a heterogenous system (e.g. bit.LITTLE), it can optimize use of an *Arm Architecture Service* function by invoking SMCCC_ARCH_FEATURES on each PE and eliminating the calls to the function on PEs that indicate the function call is not required, for example on PEs that return one (1) in the case of `SMCCC_ARCH_WORKAROUND_1`.

5 Appendix B: Mitigating CVE-2017-5715 on Arm CPUs

Mitigations for CVE-2017-5715 on Arm Cortex CPUs involves invalidation of the branch predictor. The approach differs depending on the CPU model. See also the *Arm Security Update* [1] and the Technical Reference Manuals (TRMs) for each CPU on the *Arm Developer* website [6].

Table 3 provides a summary of the actions required.

	Cores	AArch64 Workarounds	AArch32 Workarounds
Armv8-A	Cortex-A57 Cortex-A72	MMU Disable/Enable [(a)]	MMU Disable/Enable [(a)]
	Cortex-A73 Cortex-A75	BPIALL instruction [(b)]	BPIALL instruction
	Cortex-A35 Cortex-A53 Cortex-A55	Not affected	Not affected
	Cortex-A32	- [(e)]	Not affected
Armv7-A	Cortex-A8	- [(e)]	BPIALL instruction [(c)]
	Cortex-A9 Cortex-A12 Cortex-A17	- [(e)]	BPIALL instruction
	Cortex-A15	- [(e)]	ICIALLU instruction [(d)]
	Cortex-A5 Cortex-A7	- [(e)]	Not affected

Table 3 CVE-2017-5715 mitigations on Arm Cortex CPUs

(a) The software must disable the MMU for the currently active translation regime. The toggling code must be mapped so that physical and virtual address are the same.

(b) This must be done from a privileged exception level in AArch32 execution state.

(c) For Cortex-A8, ACTLR[6] ‘IBE’ must be equal to 1 to enable the BPIALL instruction.

(d) For Cortex-A15, ACTLR[0] must be equal to 1 in order to invalidate the branch predictor when performing the ICIALLU instruction.

(e) These cores are AArch32 only

Constraints (a) and (b), in particular, imply that the firmware based mitigation described in this document is the most effective, or only, mitigation available for system software.

6 Appendix C: Mitigating CVE-2018-3639 on Arm CPUs

Mitigations for CVE-2018-3639 on Arm Cortex CPUs involve disabling the bypassing of writes by reads (including speculative reads), either permanently during CPU initialization, or dynamically as required. The approach differs depending on the CPU model. See also the *Arm Security Update* [1] and the Technical Reference Manuals (TRMs) for each CPU on the *Arm Developer* website [6].

	Cores	Workarounds
Armv8-A	Cortex-A57 Cortex-A72	Permanently set bit 55 (Disable load pass store) of CPUACTLR_EL1 (S3_1_C15_C2_0)
	Cortex-A73	Permanently set bit 3 of S3_0_C15_C0_0 (not in TRM)
	Cortex-A75	Permanently set bit 35 (reserved in TRM) of CPUACTLR_EL1 (S3_0_C15_C1_0)
	Cortex-A35 Cortex-A53 Cortex-A55 Cortex-A32	Not affected
Armv7-A	Cortex-A8 Cortex-A9 Cortex-A12 Cortex-A17 Cortex-A15 Cortex-A5 Cortex-A7	Not affected

Table 4 CVE-2018-3639 mitigations on Arm CPUs