Intel® Software Guard Extensions (SGX) SW Development Guidance for Potential Bounds Check Bypass (CVE-2017-5753) Side Channel Exploits

White Paper

Revision 1.0
February 2018

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at www.intel.com.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

Intel disclaims all implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

Intel, and the Intel logo are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others

Copyright © 2018, Intel Corporation.

Contents

1 Glossary/Acronyms 1

2 Background 2

3 Intel® SGX SDK Changes 3

4 Intel® SGX Developer Guidance 4

4.1 Enclave Inputs........ 4

4.1.1 Example 5

4.1.2 user_check EDL Attribute........ 6

4.1.3 Table/Array Indexing........ 7

4.1.4 sizefunc 7

5 Compiler Support........ 9

6 References........ 10

§

Revision History

Revision Number	Description	Date
1.0	Initial version.	February 2018

THIS PAGE IS LEFT INTENTIONALLY BLANK

1 Glossary/Acronyms

Term/Acronym	**Meaning**
ECALL	Call into an enclave.
OCALL	Call outside an enclave.
tRTS	Intel® SGX trusted runtime system. A static library included in the Intel® SGX SDK and built into any enclave built with the SDK.
EDL	Enclave Definition Language. A language like COM IDL used for defining interfaces, in this case this interface to an enclave.
Edger8r	SGX SDK Tool used to compile EDL files.

2 Background

On January 3, 2018, a team of security researchers at Google Project Zero disclosed [GPZ] three variants of a side-channel analysis method that, when used for malicious purposes, have the potential to improperly gather sensitive data from many types of computing devices with many different vendors' processors and system software. These three vulnerabilities are documented in Intel Security Advisory [SECADV], and are also known as 'Spectre' (referring to Variants 1 and 2) and 'Meltdown' (referring to Variant 3).

Applications that use Intel® Software Guard Extensions (Intel® SGX) are vulnerable to the Google Project Zero side-channel analysis methods until mitigations[1] described in this document have been implemented. While any attack on an SGX enclave will be specific to the way the enclave handles its data, the actions outlined in this document should be followed to help minimize potential impacts.

Guidance issued by Intel [ANALYSIS] identified that Software, not microcode, would be responsible for mitigating Bounds Check Bypass exploits.

In this document we will identify changes that have been made to the Intel® SGX Software Developer's Kit (SDK) and provide clarifying guidance on what the Intel® SGX developer needs to look for in cases that cannot be addressed automatically by recompiling with the updated SDK.

---

[1] *The mitigation techniques discussed in this document address the three new methods of side channel analysis originally disclosed by Google Project Zero. Intel continues to research these issues and cannot guarantee that these mitigation techniques will apply to vulnerabilities or variations resulting from current or future research.*

3 Intel® SGX SDK Changes

The SDK is being updated to address instances of "Bounds Check Bypass" [CVE20175753]. Stopping speculative execution in the SDK code is achieved by inserting LFENCE instructions where speculative execution might possibly lead to a secret-disclosing side channel. The following table lists the SDK changes and describes the corresponding bypass.

SDK change	**Bypass**	**SDK component impacted**
Stop speculative execution that could lead to overflow of the ECALL function table used by enclaves built with the SDK.	Outside code (attacker) controls index to this table. Check of this index could be bypassed.	tRTS library
Stop speculative execution that could lead to enclave operating on a secret in enclave memory as though it was not a secret.	In enclaves built with the SDK, buffers corresponding to pointer inputs (without certain EDL attributes) are checked to ensure that they are outside the enclave. These checks could be bypassed.	Edger8r and tRTS library
Stop speculative execution during first enclave call which doesn't execute Edger8r-generated code. The first enclave call is part of the enclave loading process.	The first enclave call passes a pointer that's treated as though it points to outside memory. Check of this can be bypassed.	tRTS library
Stop speculative execution that could lead to overflow of structure used for sealed data.	sgx_sealed_data_t structure includes length field that is used to calculate pointer values. Checks of these values could be bypassed.	tSeal library
Stop speculative execution in trusted key exchange library. Use of this library adds ecalls to enclaves that use it. One of these ecalls, sgx_ra_get_msg3_trusted, has a user_check pointer input.	Check that user_check pointer points to buffer outside enclave could be bypassed. Since user_check input, other SDK changes don't help as explained in section 4.1.2.	tkey_exchange library
Stop speculative execution in trusted key exchange library. Use of this library adds ecalls to enclaves that use it. The ecalls take a context parameter that is used as a session array index.	Outside code (attacker) controls context input. Check of context to make sure it doesn't overflow session array could be bypassed.	tkey_exchange library

4 Intel® SGX Developer Guidance

In order to take advantage of the SDK changes, developers should rebuild their enclaves with the updated SDK. The developer may choose to increment their enclaves' ISVSVNs in line with guidance in the Intel® SGX Developer Guide [SGXDEVGUIDE]. So, at some point, provisioning of secrets to an enclave or otherwise deciding to trust an enclave should require an ISVSVN that implies that the enclave was built with the updated SDK.

Rebuilding with the updated SDK may not be sufficient, as not all the conditions for Bounds Check Bypass exploit scenarios can be mitigated automatically. All loads from memory where an attacker can control the address should be analyzed. Absent any other exploit, this means the enclave developer needs to analyze all enclave inputs that aren't handled by the SDK as described in section 2. The following sections describe different types of enclave inputs and what the developer should do to add mitigations for each type.

4.1 Enclave Inputs

The following types of enclave inputs can lead to exploit

- Inputs that are interpreted as addresses/pointers
- Inputs that are used to calculate addresses
- Inputs whose contents (recursive) are interpreted as addresses/pointers or are used to calculate addresses

The reason is that an attacker can make the pointer or calculated pointer point to a secret in enclave memory. Then, code in the enclave that's only intended to run when the pointer points to outside memory can execute speculatively.

Even if the EDL for an enclave doesn't include any inputs like this, the enclave will have them. The code generated from the enclave EDL by the Edger8r puts the parameters of each ecall in a structure and passes a pointer to this "marshaling structure" to the enclave. Also, code in the tRTS along with Edger8r-generated code maintains an "ecall table" and converts the developer's ecall into an indirect call through this table. Attacks on these inputs are mitigated in the updated SDK.

The updated SDK also inserts mitigations for enclave inputs that appear in the enclave's EDL that are interpreted as addresses/pointers as long as

- They're declared as pointer types in the EDL.
- They don't use the user_check or sizefunc EDL attributes.
  - Whether "sizefunc inputs" are exploitable depends on the specified sizefunc function, which, by definition, is written by the developer. See section 4.1.4.
- The pointer-ness of the input isn't hidden via typedef.
  - In this case, the "isptr" EDL attribute can be used in order to have the input still be treated as a pointer.

However, even if a pointer input meets these criteria, if the pointer points to a structure that contains pointers or that contains fields used to calculate addresses/pointers, then the developer is responsible for analyzing the enclave code that uses these nested pointers. The updated SDK won't help in these cases, with one exception: an enclave that uses the sealing library in the SDK may take a pointer to a sealed blob as input. Code in the sealing library interprets the blob as a structure with an offset field in it. The updated SDK inserts mitigations related to the address calculated from this offset.

Real world, general cases that require developer analysis include variable-length enclave inputs with headers that include something like a payload length field. In cases like these, the overall size of the input is typically provided to the enclave and enclave code checks that the nested length fields don't cause overflow, but these checks can be mis-predicted leading to speculative execution of code that isn't supposed to run.

4.1.1 Example

```
// EDL
public uint32_t enclave_function([in, size = alloc_size]tlv_t*
varlen_input, uint32_t alloc_size);
```

```
typedef struct {
     unsigned type;
     unsigned length;
     void* payload;
} tlv_t;
```

With the EDL code above, the SDK code will make sure that alloc_size bytes are outside the enclave and the code in the updated SDK will guard against dangerous side channels in the process. However, in the code that the developer writes, there will presumably be some processing that depends on the length field of the varlen_input input. This field will not have been checked by the SDK code at all. Before the presumed length field-dependent processing in the Intel® SGX developer's code, there will presumably be a check to make sure that length is less than alloc_size. The results of this check can be mis-predicted so the developer is responsible for inserting an LFENCE instruction if the developer determines that the "valid length" path has a dangerous side channel when speculatively executed with an invalid length value. Also, if performance requirements allow, the developer can simply insert an LFENCE without exhaustive analysis of the valid length path. See below.

```
//
// make sure payload is outside enclave
//
If (varlen_input.length > alloc_size) {
     // error code
     …
}
Else {
     _mm_lfence();
     //
     // valid length path
     //
     …
}
```

_mm_lfence is the name of the LFENCE intrinsic in the Intel compiler.

4.1.2 user_check EDL Attribute

The user_check EDL attribute instructs the SDK code to not check the associated pointer input, the SDK code essentially treats the input the same way it treats an integer. So if user_check is used, the developer is responsible for analyzing the enclave code that uses the pointer. For user_check inputs and corresponding buffers that are supposed to be in memory outside the enclave, the following pattern can be used in cases where there appear to be dangerous side channels in the branch taken path or in cases where the performance impact of the LFENCE is considered to be acceptable.

```
// EDL
public uint32_t enclave_function([user_check]const uint8_t*
user_check_input, uint32_t user_check_size);
```

```
uint32_t enclave_function(const uint8_t* user_check_input,
uint32_t user_check_size)
{
	…
	//
	// make sure input buffer is outside enclave
	//
	int SGXAPI sgx_is_outside_enclave(const void *addr, size_t
	size);

	if (!sgx_is_outside_enclave(user_check_input,
	user_check_size)) {
		// error code
		…
	}
	else {
		_mm_lfence();
		…
	}
	…
}
```

sgx_is_outside_enclave is a function available in enclaves built with the SDK and, as the name implies, it checks whether the buffer specified by the pointer and size inputs is entirely outside the enclave.

If a user_check input corresponds to a structure that's supposed to be inside the enclave, an attacker can change the pointer value such that it points to the wrong memory inside the enclave. A simple check analogous to the one above isn't sufficient in this case. The enclave developer needs to use some other means to qualify the pointer.

4.1.3 Table/Array Indexing

For another example, we can look at the following victim function.

```
void victim_function(size_t x) {
     if (x < array1_size) {
          temp &= array2[array1[x] * 512];
     }
}
```

If victim_function were a trusted enclave function specified in an EDL file, the SDK code couldn't help avoid the dangerous side channel since it wouldn't know that the input was going to be used as an index. The developer of the enclave with a function like this would be responsible for changing the code to something like the following

```
void victim_function(size_t x) {
     if (x < array1_size) {
          _mm_lfence();
          temp &= array2[array1[x] * 512];
     }
}
```

4.1.4 sizefunc

The sizefunc EDL attribute allows a developer to specify a function that knows how to determine the size of an input from the contents of the input itself. In the EDL example in section 4.1.1, the developer could instead have the following EDL

```
// EDL
public uint32_t enclave_function([in, sizefunc =
calc_size]tlv_t* varlen_input);
```

The developer would write a calc_size function that knows that varlen_input has a length field and calc_size would use it

```
size_t calc_size(const tlv_t* varlen_input)
{
    return varlen_input->length;
}
```

The updated SDK will prevent exploit in simple cases like this, but if the sizefunc function itself has a dangerous side channel, then the developer is responsible for mitigating it; ie, inserting LFENCE in appropriate location(s). The reason is that that attacker controls the value of the input to the sizefunc function. The developer also has to know that the Edger8r-generated code calls the sizefunc function twice, once before copying the input into the enclave and once after. Changes to how sizefunc works are being considered for future versions of the SDK.

5 Compiler Support

Microsoft* released a blog [MSFTBLOG] on January 15, 2018 describing a new compiler switch, /Qspectre, intended to help mitigate Spectre Bounds Check Bypass vulnerabilities. Intel SGX developers can build their enclaves with compilers that support this switch and enable the switch.

6 References

Label	Item/Link	Comment
[ANALYSIS]	Intel Analysis of Speculative Execution Side Channels	Intel analysis of the side channel issues reported by Google Project Zero
[CVE20175753]	CVE-2017-5753	CVE Disclosure for Bounds Check Bypass exploit
[GPZ]	GPZ Blog	GPZ Blog on side channel issues
[MSFTBLOG]	Spectre mitigations in MSVC	Microsoft Visual C++ has introduced compile switch that will add LFENCE instructions where it discovers Bounds Check Bybass patterns
[SECADV]	Intel Security Advisory	Speculative Execution Security Advisory
[SGXDEVGUIDE]	Intel® SGX Developer Guide	Developer Guidance issued with SGX SDK