Intel® Software Guard Extensions (SGX) SW Development Guidance for Potential Edger8r Generated Code Side Channel Exploits

White Paper

Revision 1.0

March 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.

Contents

§

Revision History

Revision Number	Description	Date
1.0	Initial version.	March 2018

THIS PAGE IS LEFT INTENTIONALLY BLANK

1 Glossary/Acronyms

Term/Acronym	**Meaning**
ECALL	Call into an enclave.
OCALL	Call outside an enclave.
Proxy Function	Function which is called by the application when making an ECall. It is a “proxy” for the function within the enclave.
Bridge Function	Function which is called within the enclave and ultimately calls the developer function within the enclave. It is a “bridge” to the actual enclave function.
Untrusted Domain	Code running in the application outside the enclave. It is considered untrusted in that it should not have access to confidential information and is not integrity checked.
Trusted Domain	Code running within the 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 February 16th, 2018, a team of security researchers at Catholic University of Leuven (KU Leuven) disclosed to Intel Corporation an issue with Edger8r Tool within the Intel® Software Guard Extensions (Intel® SGX) Software Developer's Kit (SDK). This issue could cause the Edger8r tool to generate source code that could, when used as intended within an SGX enclave, expose the enclave to a side-channel attack. The attack would then have the potential to disclose confidential data within the enclave.

Applications that use the Intel® SGX SDK may use the Edger8r tool to generate C language interface code. The interface code is comprised of Proxy and Bridge functions which are used to make a call from an application into an enclave (referred to as an ECall). An Intel® SGX enclave developer provides the definition of their enclave interface to the Edger8r Tool via an EDL file, which conforms to grammar specific to the Edger8r Tool. Two elements of the grammar, the 'string' attribute and the 'sizefunc' attribute, have been identified to cause the Edger8r to generate potentially vulnerable code. Developers who use these attributes in their enclave interface definition may have exposed their enclave to a potential side-channel exploit. 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 minimize potential impacts.

In this document we will identify changes that have been made to the Intel® SGX SDK Edger8r Tool EDL Grammar and provide clarifying guidance on what the Intel® SGX developer needs to do to adapt their interface code to the updated EDL grammar.

In addition, developers should always use latest version of the SDK to ensure that tools and libraries have updated mitigations for potential Bounds Check Bypass side channel exploits. Refer to [SGXCVE20175753] for updated information on Bounds Check Bypass mitigations.

3 Intel® SGX SDK Changes

The SDK is being updated to address the Edger8r vulnerability. The following table lists the changes in Intel® SGX SDK for Windows* OS Version 1.9.106 and Intel® SGX SDK for Linux OS Version 2.1.102 and describes the corresponding impact to developers.

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

SDK change	**Impact**	**SDK component impacted**
Modify the implementation of the 'string' attribute to calculate string length in untrusted code.	Developers who use 'string' attribute should regenerate edge routines (i.e. rerun the Edger8r program) and rebuild the application and the enclave	Edger8r Tool
Remove the 'sizefunc' attribute	Developers who use the 'sizefunc' attribute should modify their application and enclave code to use an alternate size indicator such as the 'size' attribute. In addition, they must also modify their EDL code and rebuild their application and enclave	Edger8r Tool
Stop speculative execution that could lead to an enclave operating on a secret in enclave memory as though it was not a secret. Refer to [SGXCVE20175753] for details.	Developers who pass pointers of buffers into the enclave with specific attributes should regenerate edge routines (i.e. rerun the Edger8r program) and rebuild the application and the enclave.	Edger8r Tool
Adapt SDK Sample Code to remove the deprecated 'sizefunc' attribute	Outline a method to adapt an application/enclave pair to use alternate methods to the 'sizefunc' attribute.	SDK Sample Code

4 Intel® SGX Developer Guidance

In order to take advantage of the SDK changes, developers should rebuild their applications and their enclaves with the updated SDK. The developer may choose to increment their enclave's ISVSVN in line with guidance in the Intel® SGX Developer Guide [SGXDEVGUIDE]. Provisioning of secrets to an enclave or otherwise deciding to trust an enclave should require an ISVSVN that indicates that the enclave was built with the updated SDK version.

If the developer is using the *sizefunc* attribute, then merely rebuilding their code will not be sufficient. The developer must also modify their code to remove the *sizefunc* feature. This paper proposes one alternative implementation using the *size* attribute.

The following sections describe the changes to Edger8r generated code and a mitigation technique for the *sizefunc* deprecation.

4.1 Edger8r string Attribute

The use of the *string* attribute in an ECall function can lead to an exploit. The generated code which is used inside the enclave will call the standard C function *strlen* to obtain the length of the string. This length is then used to declare a buffer within the enclave in which the string is then copied.

If the pointer to the string contains an address that is within the enclave, then the call to *strlen* will operate on enclave trusted memory until it reaches a null terminator '\0'. The generated code will then perform a check that the input string buffer is not within the enclave and return a failure.

Malicious code on the outside of the enclave may manipulate the string pointer and then use timing or other techniques to obtain the byte at which a null terminator was encountered.

4.1.1 Example – Current Edger8r

Figure 1: EDL with string Attribute demonstrated EDL code with the *string* attribute. In this example, function *ecall_pointer_string* contains a string argument *str.*

```
/*
 * [string]:
 *      the attribute tells Edger8r 'str' is NULL terminated string,
 *      so strlen will be used to count the length of buffer pointed
 *      by 'str'.
 */

public void ecall_pointer_string([in, string] char *str);
```

Figure 1: EDL with string Attribute

```
typedef struct ms_ecall_pointer_string_t {
      char* ms_str;
} ms_ecall_pointer_string_t;
```

Figure 2: Edger8r Generated Marshalling Structure for string Attribute

When the EDL code in Figure 1 is processed by the Edger8r Tool, it will produce a marshalling structure of type `ms_ecall_pointer_string_t.` The structure provides element *ms_str* which is a pointer to a `char` array.

```
sgx_status_t ecall_pointer_string(sgx_enclave_id_t eid, char* str)
{
      sgx_status_t status;
      ms_ecall_pointer_string_t ms;
      ms.ms_str = str;
      status = sgx_ecall(eid, 12, &ocall_table_Enclave, &ms);
      return status;
}
```

Figure 3: Edger8r Generated Proxy Function Using string Attribute

Figure 3: Edger8r Generated Proxy Function Using string Attribute shows the Edger8r generated proxy function which is compiled into the untrusted application. In this function, the user supplied pointer to the string is copied to the marshalling structure.

```
#define CHECK_REF_POINTER(ptr, siz) do {    \
    if (!(ptr) || ! sgx_is_outside_enclave((ptr), (siz)))  \
        return SGX_ERROR_INVALID_PARAMETER;\
} while (0)

#define CHECK_UNIQUE_POINTER(ptr, siz) do { \
    if ((ptr) && ! sgx_is_outside_enclave((ptr), (siz)))   \
        return SGX_ERROR_INVALID_PARAMETER;\
} while (0)

static sgx_status_t SGX_CDECL sgx_ecall_pointer_string(void* pms)
{
    CHECK_REF_POINTER(pms, sizeof(ms_ecall_pointer_string_t));
    ms_ecall_pointer_string_t* ms =
            SGX_CAST(ms_ecall_pointer_string_t*, pms);
    sgx_status_t status = SGX_SUCCESS;
    char* _tmp_str = ms->ms_str;
    size_t _len_str = _tmp_str ? strlen(_tmp_str) + 1 : 0;
    char* _in_str = NULL;

    CHECK_UNIQUE_POINTER(_tmp_str, _len_str);
    //
    // fence after pointer checks
    //
    _mm_lfence();

    if (_tmp_str != NULL && _len_str != 0) {
        _in_str = (char*)malloc(_len_str);
        if (_in_str == NULL) {
            status = SGX_ERROR_OUT_OF_MEMORY;
            goto err;
        }

        memcpy(_in_str, _tmp_str, _len_str);
        _in_str[_len_str - 1] = '\0';
    }

    ecall_pointer_string(_in_str);
err:
    if (_in_str) free(_in_str);

    return status;
}
```

Figure 4: Edger8r Generated Bridge Function Using string Attribute

Figure 4: Edger8r Generated Bridge Function Using string Attribute provides the trusted Bridge Function which is produced by the current Edger8r Tool. In this function, *strlen* is called on the

ms_str pointer supplied to the function in the marshalling structure. The pointer is not confirmed to be outside the enclave until after *strlen* has executed on it.

4.1.2 Example – Updated Edger8r

The Edger8r Tool has been updated to generate the marshalling structure depicted in Figure 5: Updated Edger8r Generated Marshalling Structure for string Attribute.

```
typedef struct ms_ecall_pointer_string_t {
      char* ms_str;
      size_t ms_len_str;
} ms_ecall_pointer_string_t;
```

Figure 5: Updated Edger8r Generated Marshalling Structure for string Attribute

When the EDL code in Figure 1 is processed by the updated Edger8r Tool, it will produce a marshalling structure of type `ms_ecall_pointer_string_t.` The structure provides element *ms_str* which is a pointer to a `char` array and an additional element *ms_len_str* which contains the length of the *ms_str* array including the null terminator.

```
sgx_status_t ecall_pointer_string(sgx_enclave_id_t eid, char* str)
{
      sgx_status_t status;
      ms_ecall_pointer_string_t ms;
      ms.ms_str = str;
      ms.ms_len_str = strlen(str) + 1;
      status = sgx_ecall(eid, 12, &ocall_table_Enclave, &ms);
      return status;
}
```

Figure 6: Updated Edger8r Generated Proxy Function Using string Attribute

Figure 6: Updated Edger8r Generated Proxy Function Using string Attribute shows the update Edger8r generated proxy function which is compiled into the untrusted application. In this function, the user supplied pointer to the string is copied to the marshalling structure. In addition, the string length is calculated and also supplied in the marshalling structure. It should be noted that this string length calculation is done in the untrusted domain which does not have access to enclave memory.

```
#define CHECK_REF_POINTER(ptr, siz) do {    \
      if (!(ptr) || ! sgx_is_outside_enclave((ptr), (siz)))  \
            return SGX_ERROR_INVALID_PARAMETER;\
} while (0)

#define CHECK_UNIQUE_POINTER(ptr, siz) do { \
      if ((ptr) && ! sgx_is_outside_enclave((ptr), (siz)))   \
            return SGX_ERROR_INVALID_PARAMETER;\
} while (0)


static sgx_status_t SGX_CDECL sgx_ecall_pointer_string(void* pms)
{
      CHECK_REF_POINTER(pms, sizeof(ms_ecall_pointer_string_t));
      //
      // fence after pointer checks
      //
      sgx_lfence();
      ms_ecall_pointer_string_t* ms =
              SGX_CAST(ms_ecall_pointer_string_t*, pms);
      sgx_status_t status = SGX_SUCCESS;
      char* _tmp_str = ms->ms_str;
      size_t _len_str = ms->ms_len_str;
      char* _in_str = NULL;

      CHECK_UNIQUE_POINTER(_tmp_str, _len_str);
      //
      // fence after pointer checks
      //
      sgx_lfence();

      if (_tmp_str != NULL) {
            _in_str = (char*)malloc(_len_str);
            if (_in_str == NULL) {
                  status = SGX_ERROR_OUT_OF_MEMORY;
                  goto err;
            }

            memcpy(_in_str, _tmp_str, _len_str);
            _in_str[_len_str - 1] = '\0';
      }
      ecall_pointer_string(_in_str);
err:
      if (_in_str) free(_in_str);

      return status;
}
```

Figure 7: Updated Edger8r Generated Bridge Function Using string Attribute

Figure 7: Updated Edger8r Generated Bridge Function Using string Attribute provides the trusted Bridge Function which is produced by the current Edger8r Tool. In this function, *strlen* is called on the *ms_str* pointer supplied to the function in the marshalling structure. The pointer is not confirmed to be outside the enclave until after *strlen* has executed on it.

Note: in Figure 7: Updated Edger8r Generated Bridge Function Using string Attribute the *_mm_lfence()* instrinsic has changed to the *sgx_lfence()* macro to be more compiler compatible. In addition, the tool will add an extra fence after the initial check of the marshalling structure pointer per [SGXCVE20175753] .

4.2 Edger8r sizefunc Attribute

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. The function runs inside the enclave and since it's responsible for determining the size of the input, the input has to be accessed before the check to make sure that it's entirely outside the enclave. This process is susceptible to side channels. Therefore, the sizefunc attribute is being removed. In its place, developers should use the size attribute and their enclave code should always make sure that it won't go past the end of the input before it processes the input (see example below) or that it's not going past the end of the input as it processes the input.

4.2.1 Example – Current Edger8r using 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 Figure 9: EDL for sizefunc the developer passes in a variable length structure defined in Figure 8: Variable Length Structure Type.

```
typedef struct _tlv_t {
        uint32_t buf_len;
        uint8_t  buf[];
} tlv_t;
```

Figure 8: Variable Length Structure Type

```
/*
 * [sizefunc]:
 *      the attribute tells Edger8r that calc_size can be used to
 *      calculate the size of varlen_input
 */

public void ecall_sizefunc([in, sizefunc = calc_size] tlv_t* varlen_input);
```

Figure 9: EDL for sizefunc

The developer would write a calc_size function that knows that varlen_input has information to compute the length of varlen_input. calc_size would use varlen_input to calculate the size of the variable length structure. A simple example is shown in Figure 10: calc_size Function Example.

```
size_t calc_size(const tlv_t* varlen_input)
{
   return ( sizeof(tlv_t) + varlen_input->buf_len );
}

void ecall_sizefunc(tlv_t* varlen_input)
{
   //process varlen_input
   return;
}
```

Figure 10: calc_size Function Example

Figure 11: Edger8r Generated Bridge Function Using sizefunc provides the trusted Bridge Function which is produced by the current Edger8r Tool. In this function, *calc_size* is first called on the *ms_varlen_input* pointer supplied to the function in the marshalling structure. The pointer is not confirmed to be outside the enclave until after *calc_size* has executed on it. *calc_size* has a dangerous side channel in that it does not check that *ms_varlen_input* is entirely outside the enclave. In fact, calc_size is called twice, once on a structure intended to be outside the enclave, ms->ms_varlen_input, and once on a structure inside the enclave, _in_varlen_input. This is to confirm that the size of the structure copied conforms to the size of the structure first calculated.

This presents a small conundrum for the implementation of sizefunc. The bridge routine must know the size of the structure passed to sizefunc in order to ensure that the structure is outside the enclave; however the bridge routine must use sizefunc to calculate the size.

```
static sgx_status_t SGX_CDECL sgx_ecall_sizefunc(void* pms)
{
        CHECK_REF_POINTER(pms, sizeof(ms_ecall_sizefunc_t));
        ms_ecall_sizefunc_t* ms = SGX_CAST(ms_ecall_sizefunc_t*, pms);
        sgx_status_t status = SGX_SUCCESS;
        tlv_t* _tmp_varlen_input = ms->ms_varlen_input;
        size_t _len_varlen_input = ((_tmp_varlen_input) ?
                                   calc_size(_tmp_varlen_input) : 0);
        tlv_t* _in_varlen_input = NULL;

        CHECK_UNIQUE_POINTER(_tmp_varlen_input, _len_varlen_input);
        //
        // fence after pointer checks
        //
        _mm_lfence();
```

```
    if (_tmp_varlen_input != NULL && _len_varlen_input != 0) {
        _in_varlen_input = (tlv_t*)malloc(_len_varlen_input);
        if (_in_varlen_input == NULL) {
            status = SGX_ERROR_OUT_OF_MEMORY;
            goto err;
        }

        memcpy(_in_varlen_input, _tmp_varlen_input,
            _len_varlen_input);

        /* check whether the pointer is modified. */
        if (calc_size(_in_varlen_input) != _len_varlen_input) {
            status = SGX_ERROR_INVALID_PARAMETER;
            goto err;
        }

        //
        // fence after final sizefunc check
        //
        _mm_lfence();
    }

    ecall_sizefunc(_in_varlen_input);
err:
    if (_in_varlen_input) free(_in_varlen_input);

    return status;
}
```

Figure 11: Edger8r Generated Bridge Function Using sizefunc

The clear solution is similar to the solution used with the variable length string parameter, which is to calculate the length of the string using a sizefunc on the outside of the enclave in the untrusted proxy function and then add a length parameter to the marshalling structure. This requires the developer to write a sizefunc to be used in the application.

This solution also presents another problem. There must be a sizefunc on the inside of the enclave to perform the second check of the buffer copied to the inside of the enclave. The second check is required to ensure that the processing of the variable length buffer does not overrun the same variable length buffer.

Thus, this solution will require the developer to write:

- A modification to the marshalling structure to supply the length of the buffer.
- An untrusted sizefunc to be used to calculate the length of the buffer passed to the enclave
- A trusted sizefunc to verify the length of the buffer which has been copied to the enclave.

The reason that the sizefunc was provided in the Edger8r was to save the developer some work in the marshalling of the data from the untrusted to the trusted environment. This solution limits that value because the developer must now write multiple functions.

For this reason, it has been decided that sizefunc will be removed from the Edger8r tool. The developer may use the size attribute instead within the EDL.

4.2.2 Example – Implementation without sizefunc

The *size* attribute can be used to implement the previous example of section 4.2.1 Example – Current Edger8r. The updated EDL using the *size* attribute is provided in Figure 12: EDL for Implementation without sizefunc

```
/*
 * [size]:
 *      the attribute tells Edger8r that len is the size of
 *      varlen_input
 */

public void ecall_no_sizefunc([in, size = len] tlv_t* varlen_input, size_t len);
```

Figure 12: EDL for Implementation without sizefunc

The developer will need to write untrusted application code to calculate the size of the variable length structure. In Figure 13: ucalc_size Function Example, the untrusted ucalc_size function is used to provide the size of the structure which is now a parameter in the ECall proxy function *ecall_no_sizefunc*

```
/*
 * ucalc_size:
 *   calculates the size of a tlv_t structure
 */
size_t ucalc_size(const tlv_t* varlen_input)
{
      return (sizeof(tlv_t) + varlen_input->buf_len);
}

int SGX_CDECL main(int argc, char *argv[])
{
      tlv_t varlen_struct;

      …

      ret = ecall_no_sizefunc(global_eid,
                      &varlen_struct,
                      ucalc_size(&varlen_struct));
      …
}
```

Figure 13: ucalc_size Function Example with Use

Figure 14: Edger8r Generated Bridge Function using size Attribute provides the trusted Bridge Function. This function does not analyze the variable length structure to determine its size. It first ensures that the entire structure is outside of the enclave and then copies it into a buffer allocated within the enclave.

```
static sgx_status_t SGX_CDECL sgx_ecall_no_sizefunc(void* pms)
{
      CHECK_REF_POINTER(pms, sizeof(ms_ecall_no_sizefunc_t));
      //
      // fence after pointer checks
      //
      sgx_lfence();
      ms_ecall_no_sizefunc_t* ms =
                  SGX_CAST(ms_ecall_no_sizefunc_t*, pms);
      sgx_status_t status = SGX_SUCCESS;
      tlv_t* _tmp_varlen_input = ms->ms_varlen_input;
      size_t _tmp_len = ms->ms_len;
      size_t _len_varlen_input = _tmp_len;
      tlv_t* _in_varlen_input = NULL;

      CHECK_UNIQUE_POINTER(_tmp_varlen_input, _len_varlen_input);

      //
      // fence after pointer checks
      //
      _mm_lfence();

      if (_tmp_varlen_input != NULL && _len_varlen_input != 0) {
            _in_varlen_input = (tlv_t*)malloc(_len_varlen_input);
            if (_in_varlen_input == NULL) {
                  status = SGX_ERROR_OUT_OF_MEMORY;
                  goto err;
            }

            memcpy(_in_varlen_input, _tmp_varlen_input,
                             _len_varlen_input);
      }

      ecall_no_sizefunc(_in_varlen_input, _tmp_len);
err:
      if (_in_varlen_input) free(_in_varlen_input);

      return status;
}
```

Figure 14: Edger8r Generated Bridge Function using size Attribute

Figure 15: Example Code with Enclave Function Verifying Structure Size provides an example of a developer function to calculate the size of the structure. Note the use of the _mm_lfence intrinsic to protect against speculative branch execution which could make the enclave vulnerable to side-channel exploits.

In this case the developer function must ensure that the variable size input structure is properly sized to size len.

```
size_t tcalc_size(const tlv_t* varlen_input, size_t maxlen)
{
      size_t len = sizeof(tlv_t);
      if (len > maxlen)
      {
            len = 0;
      }
      else
      {
            _mm_lfence();  //fence after maxlen check (CVE-2017-5753)
            len = sizeof(tlv_t) + varlen_input->buf_len;
            if (len > maxlen)
            {
                  len = 0;
            }
      }
      _mm_lfence();  //fence after maxlen check (CVE-2017-5753)
      return len;
}


void ecall_no_sizefunc(tlv_t* varlen_input, size_t len)
{
      if (tcalc_size(varlen_input, len) == 0)
      {
            //record the error
            return;
      }
      //tcalc_size has already performed the fence

      //process varlen_input
      return;
}
```

Figure 15: Example Code with Enclave Function Verifying Structure Size

5 References

Label	Item/Link	Comment
[SGXCVE20175753]	Intel® Software Guard Extensions (SGX) SW Development Guidance for Potential Bounds Check Bypass (CVE-2017-5753) Side Channel Exploits	See Revision 2.0 with updated Developer Guidance to address Bounds Check Bypass Exploit (CVE-2017-5753)
[SGXDEVGUIDE]	Intel® SGX Developer Guide	Developer Guidance issued with SGX SDK