WHITE PAPER

Intel® Software Guard Extensions (Intel® SGX)

Input Types and Boundary Checking in Enclave-Definition Language (EDL) Files

Scope

This paper explains the input types used in Intel® Software Guard Extensions (Intel® SGX) Enclave-Definition Language (EDL) files and describes the boundary conditions for each type. The paper also covers common build errors related to the definitions in an EDL file. The information in this paper applies to Intel SGX applications for both Microsoft* Windows* and for the Linux* OS. The paper assumes a basic knowledge of Intel SGX. Information on Intel SGX can be found on the Intel SGX portal at: https://software.intel.com/en-us/sgx.

Introduction

Intel SGX applications are divided into two logical components:

- **Trusted component** — The portion of the application code that accesses the secret or sensitive data. This component of the application code is also referred to as an "enclave." An application can have more one enclave.
- **Untrusted component** — The remainder of the application, including all of its modules, which does not access the secret information.

An enclave is a protected area in an application's address space, which provides confidentiality and integrity. Attempts to access an enclave memory area from software not resident in the enclave are prevented, even from privileged software such as virtual-machine monitors, BIOS, or OSs.

An enclave provides a protected area for applications to process sensitive data. Intel provides special hardware instructions to create and support enclaves. Intel SGX enclaves use the same OS and hardware as other applications; thus, applications that make use of Intel SGX enclaves can harness the capabilities and features provided by the OS.

From an application perspective, making an enclave call (ECALL) appears as a function call when using the untrusted-proxy function. In exceptional cases, the code within the enclave needs to call external functions that reside in untrusted (unprotected) memory. This type of function call is called an OCALL.

Enclave-Definition Language

Enclave-Definition Language (EDL) files define the ECALL and OCALL functions as well as how data is to be moved into and out of enclaves. The **sgx_edger8r** tool that ships as part of the Intel SGX SDK takes an EDL file as input and creates C wrapper functions (edge routines) for

both enclave ECALLs and OCALLs. Normally, the **sgx_edger8r** tool runs automatically as part of the enclave build.

EDL files contain both trusted and untrusted blocks. ECALL functions are declared in the trusted section, and OCALL functions are declared in the untrusted section. Figure 1 shows the structure of the EDL file template.

```
enclave {
        // Include files
        // Import other EDL files
        // Data structure declarations to be used as
        // parameters of the function prototypes in EDL

    trusted {
        /* Define ECALLs here */
        // Include file if any
        // It will be inserted in the trusted header file (enclave_t.h)
        // Trusted function prototypes
    };

    untrusted {
        /* Define OCALLs here */
        // Include file if any
        // It will be inserted in the untrusted header file (enclave_u.h)
        // Untrusted function prototypes
    };
}
```

Figure 1. Enclave Definition Language file template

EDL Input and Bound Check

Developers define the input/output parameters to be handled and checked at the enclave boundary in the EDL file. **sgx_edger8r** reads the EDL file and generates the edge routines. The boundary check is accomplished at runtime by the trusted bridge and trusted proxy, which are the edge routines inside the enclave and are shown Figure 2.

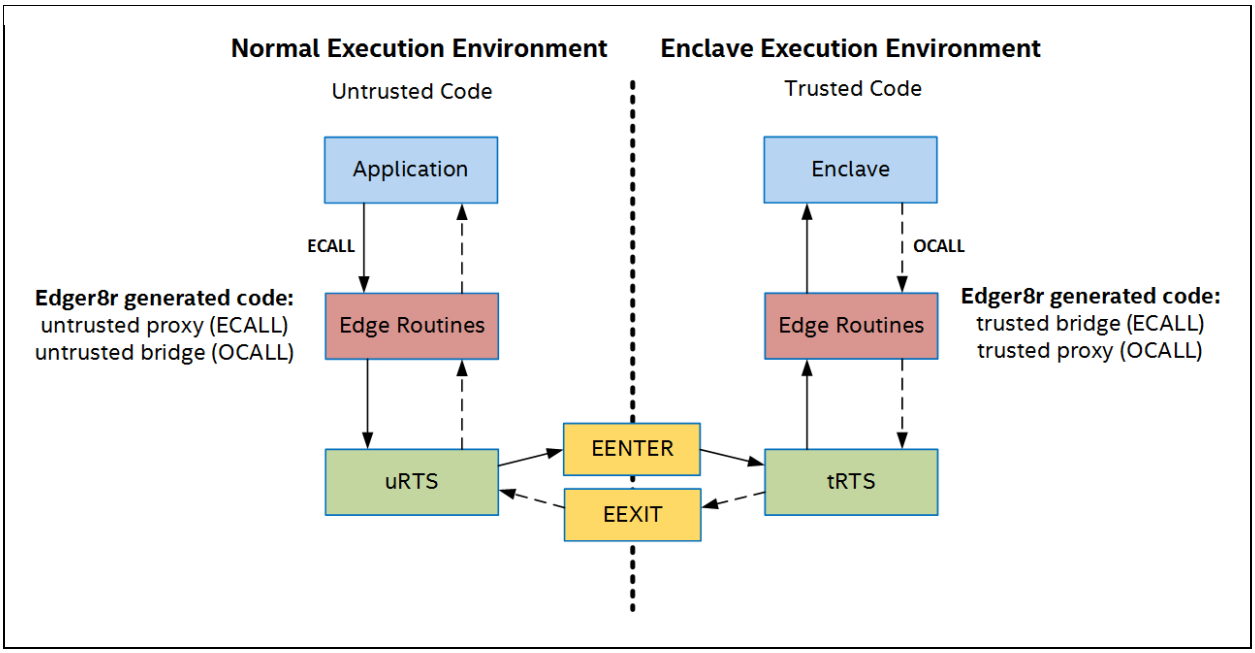

Figure 2. Enclave interface showing trusted edge routines

Input parameters passed between the application and enclave are verified by the trusted edge routines for security. The trusted bridge verifies the input parameters of ECALLs to ensure enclave memory is not unintentionally overwritten and to marshal necessary data, while preventing Time of Check to Time of Use (TOCTOU) attacks. The trusted proxy mainly copies the input parameters from an enclave to untrusted memory at the beginning of an OCALL. Depending on the marshalling attributes specified by the user in the EDL file, if certain conditions are not satisfied, the trusted bridge or trusted proxy report an error.

Pointer Handling in EDL

When an application makes an ECALL with the `in` attribute and with a pointer or array argument the trusted edge routine copies the memory content into a trusted memory area and pass the copy to the trusted environment. This is shown in the first part of Figure 3.

When an application makes an ECALL with the `out` attribute and with an array or pointer argument, the trusted edge routine allocates a buffer in trusted memory area, zeroes it and passes it to the trusted environment. This is shown in the second part of Figure 3.

When the trusted function returns, the trusted bridge copies the buffer contents from trusted memory to untrusted memory, as shown in the third part of Figure 3.

> **Note:** An enclave should not store sensitive data in a buffer marshalled with the `out` attribute. If the ECALL fails prematurely, the trusted bridge, which cannot know if the ECALL completes successfully or with an error, will copy the sensitive data to untrusted memory.

Arguments are handled by OCALL functions in a similar fashion, copying relevant memory content from the trusted to the untrusted memory areas for the `in` attribute, or in the reverse direction for the `out` attribute.

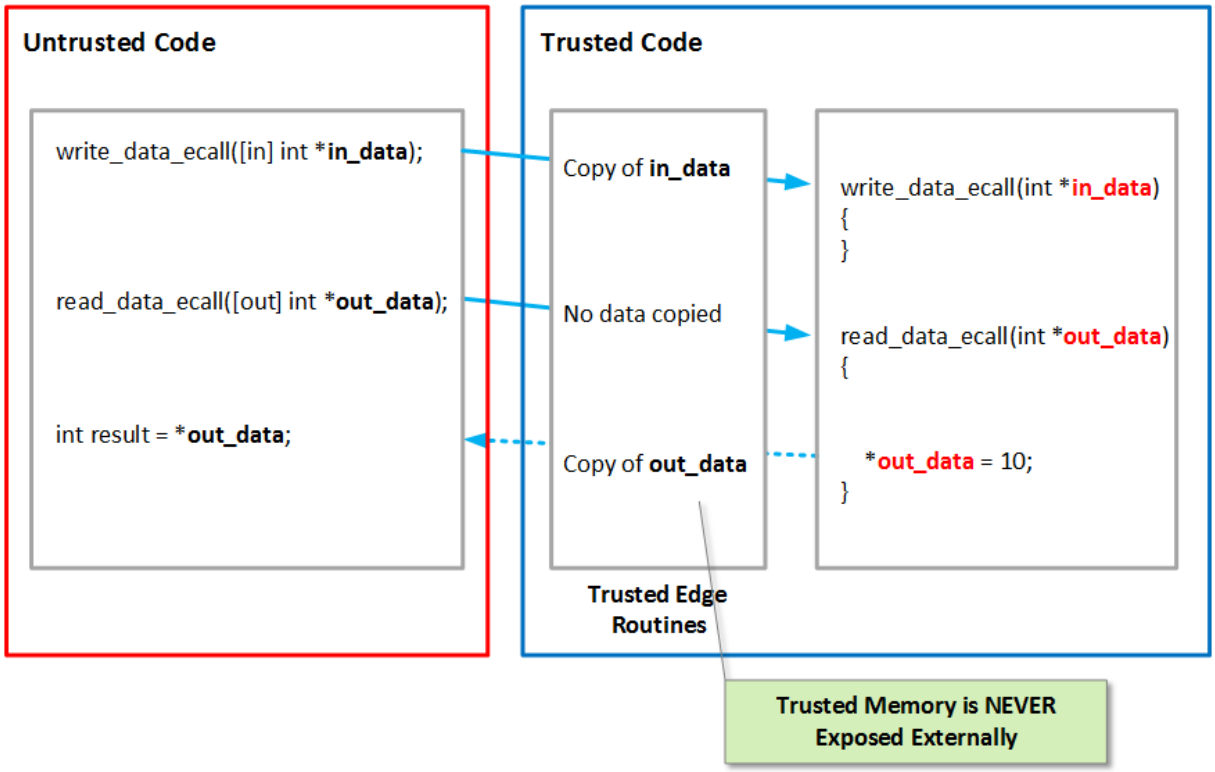

Figure 3. ECALL functions with pointers arguments and "in" and "out" attributes

A pointer argument can also be used with the `user_check` attribute. If the pointer argument is passed with the `user_check` attribute, then the trusted edge routine will not verify the pointer. ***The developer must write code to verify the pointer before using it.*** The buffer pointed to by the pointer is *not* copied; rather the address is passed. If a pointer argument is used with `user_check` attributes, the `in` and `out` attributes and attribute modifiers like `count` and `size` *cannot* be used (**sgx_edger8r** generates an error.)

An appropriate use of the `user_check` attribute in an ECALL would be to load and store an encrypted collection of data, such as an encrypted password vault. While design constraints put a practical limit on the size of the data, generally speaking these sorts of bulk reads and writes benefit from allowing the enclave to process larger data collections in smaller chunks.

The following example shows the usage of the `user_check` attribute.

```
enclave {
    trusted {
        public void test_ecall_user_check([user_check] int * ptr);
    };
    untrusted {
        void test_ocall_user_check([user_check] int * ptr);
    };
```

The developer may use a user-defined pointer type in the EDL file as long as it has been previously defined in a header file. In this case, the developer must also provide the attribute `isptr`. Otherwise, the **sgx_edger8r** tool will report an error.

Intel SGX provides attribute modifiers for `in` and `out` to help pass pointers between trusted and untrusted memory through an ECALL/OCALL. The following sections summarize these attribute modifiers.

count

The `count` modifier indicates the number of elements pointed by the pointer used for copy depending on the direction attribute (`[in]`/`[out]`). (The default number of elements is 1; the `count` modifier overrides that default.) The number of bytes copied by the trusted bridge or trusted proxy is the product of the `count` and the size of the data type to which the pointer parameter points. The `count` may be either an integer constant or one of the parameters to the function. The following example shows valid uses of `count`.

```
enclave{
    trusted {
        // Copies '100 * sizeof(int)' bytes
        public void test_count_fix([in, count=100] int* ptr);

        // Copies 'cnt * sizeof(int)' bytes
        public void test_count_var([in, count=cnt] int* ptr, unsigned cnt);
    };
};
```

size

The `size` modifier indicates the buffer size in bytes used for copy depending on the direction attribute (`[in]`/`[out]`) (when there is no `count` modifier specified). (The default size is the size of the type pointed to; the `size` modifier overrides that default.) The `size` may be either an integer constant or one of the parameters to the function. The `size` attribute alone is generally used for `void` pointers. The following examples show valid uses of `size`.

```
enclave{
    trusted {
        // Copies '100' bytes
        public void test_size_fix([in, size=100] void* ptr);

        // Copies 'sz' bytes
        public void test_size_var([in, size=sz] void* ptr, size_t sz);

        // Copies 'cnt * sz' bytes
        public void test_count_size([in, count=cnt, size=sz] int* ptr, unsigned cnt, size_t sz);

    };
};
```

The `size` and `count` attribute modifiers may also be combined. In this case, the trusted edge-routine will copy a number of bytes that is the product of the `count` and `size` parameters (`size*count`) specified in the function declaration in the EDL file.

sizefunc

The `sizefunc` attribute modifier depends on a user-defined trusted function that is called by the edge-routines to determine the number of bytes to be copied. An example of where `sizefunc` can be used is for marshaling in variable-length structures, which are buffers whose total size is specified by a combination of values stored at well-defined locations inside the buffer (although typically it is at a single location). To prevent “check first, use later” type of attacks, the function specified by `sizefunc` is called twice. In the first call, the function operates in untrusted memory. The second time, it operates in the data copied into trusted memory. If the sizes returned by the two calls do not match, the trusted bridge cancels the

ECALL and reports an error to the untrusted application. The developer must provide a function definition with the following signature:

```
size_t sizefunc_function_name(const parameter_type * p);
```

where `parameter_type` is the data type of the parameter annotated with the `sizefunc` attribute. In the example below, `parameter_type` would be `void`.

```
enclave{
    trusted {
        // Copies get_packet_size bytes
        public void test_sizefunc([in, sizefunc=get_packet_size] void* ptr);
    };
};
```

Note: Do not use `sizefunc` with `strlen` and `wstrlen`; use `string` and `wstring` instead. See the discussion of string handling in a later subsection for details.

Handling Arrays in EDL

The **sgx_edger8r** tool imposes a limitation on attempted marshaling of arrays. The array must have a defined size. Otherwise, the **sgx_edger8r** tool does not know how much data it needs to copy across the enclave boundary. This means that arrays without a given dimension cannot be used in the declaration of ECALL/OCALL functions in the EDL file.

The example below shows an array of 500 integers passed as an ECALL parameter. To marshal this array, the trusted proxy allocates `500*sizeof(int)` bytes in trusted memory. Then it copies the array content from untrusted to trusted memory. The ECALL function only operates on the copy of the array that was allocated in trusted memory.

The developer may use a user-defined array type in the EDL file as long as it has been previously defined in a header file. In this case, the developer must also provide the attribute `isary`. Otherwise, the **sgx_edger8r** tool will report an error.

```
enclave{
    trusted {
        public void test_array([in] int arr[500]);
    };
};
```

Handling Strings in EDL

The attributes `string` and `wstring` indicate that the parameter is a `NULL` terminated C string or a `NULL` terminated `wchar_t string`, respectively. To prevent “check first, use later” type of attacks, the trusted edge-routine first operates in untrusted memory to determine the length of the string. Once the string has been copied into the enclave, the trusted bridge

explicitly `NULL` terminates the string. The size of the buffer allocated in memory accounts for the length determined in the first step as well as the size of the string termination character.

The `string` and `wstring` attributes must not be combined with any other modifier such as `size`, `count`, or `sizefunc`. `string` and `wstring` cannot be used with `out` alone. In all these cases, **sgx_edger8r** will report an error. However, `string` and `wstring` with both `in` and `out` are accepted. The following example shows valid uses of `string` and `wstring`.

```
enclave{
    trusted {
        public void test_string([in, out, string] char* str);

        public void test_wstring([in, out, wstring] wchar_t* wstr);

    };
};
```

Notes

- Be aware that `in`/`out` arguments with large buffer sizes result in large enclaves, which can restrict the usage of trusted memory for other enclave code (your enclave or additional enclaves). To achieve efficient memory usage, make sure that only data that must be protected is passed to the enclave. Another way to help ensure efficient memory usage is to adjust the heap/stack size of your enclave based on actual usage; information on how to achieve this is provided in this white paper: https://software.intel.com/sites/default/files/managed/09/37/Enclave-Measurement-Tool-Intel-SGX.pdf.)
- Large enclave/buffer sizes can also affect the performance of enclave-code execution. For more information on enclave performance see this white paper: https://software.intel.com/sites/default/files/managed/09/37/Intel-SGX-Performance-Considerations.pdf.

Common build errors

This section lists the most common EDL errors seen during builds. Examples of correct syntax are provided for each potential issue.

- **string or char pointer:** While passing a string to the enclave, the string must be passed as input in the following format:

```
public void test_string([in, out, string] char* str);
public void test_wstring([in, out, wstring] char_t* wstr);
public void test_const_string([in, string] const char* str);
```

[in]: Must be present when sending input from the application to the enclave in string format during ECALL; during an OCALL the parameter is passed from the enclave to the application.
[out]: Must be present when we need to return a value from the enclave to the application, but it cannot be used alone.
[string]: The value passed is a string.
[wstring]: The value passed is a wstring.

Make sure you include the pointer direction to avoid the following errors:

1. string /wstring attributes must NOT be used without pointer direction([in/out])

```
public void test_string_cant([string] char* str);
```

Error:

```
size/string attributes must be used with pointer direction
```

2. string/wstring attributes cannot be used with [out] attribute alone

```
public void test_string_out([out, string] char* str);
```

Error:

```
string/wstring/sizefunc should be used with an `in' attribute
```

3. sizefunc can't be used for strings, use [string/wstring]

```
public void test_string_sizefunc_cant([in, string, sizefunc=strlen] char* str);
```

Error:

```
size attributes are mutual exclusive with (w)string attribute
```

- **Arrays:** While passing arrays to an Enclave, input must be passed in the following format:

```
public void test_array([in] int arr[400]);
public void test_array_multi([in] int arr[4][400]);
public void test_isary([in, isary] array_t arr);
```

Note the following limitations/restrictions to avoid the related errors:

1. Flexible array is not supported

```
public void test_flexible(int arr[][400]);
```

Error:

```
Flexible array is not supported
```

2. Zero-length array is not supported.

```
public void test_zero(int arr[0]);
```

Error:

```
Zero-length array is not supported
```

3. User-defined array types need "isary"

```
public void test_miss_isary([in] array_t arr);
```

Error:

```
`array_t' is considered plain type but decorated with pointer attributes
```

4. "size" or "count" cannot be used with "isary"

```
public void test_array_with_size([in, isary, size=sz] array_t arr, size_t sz);
```

Error:

```
Pointer size attributes cannot be used with foreign array
```

5. "size" or "count" cannot be used with array

```
public void test_array_with_size([in, size=sz] int arr[400], size_t sz);
```

Behavior:

`size` or `count` is not used, but no warning is provided.

- **Pointers:** While passing pointers, the direction attribute instructs trusted edge-routines to copy the buffer pointed by the pointer. However, the trusted-edge routines need to know how much data must be copied. Therefore, when passing pointers, the required memory should be assigned to the pointer prior to passing the data from application to enclave or vice-versa.
  Pointers syntax form is as follows:

```
public void test_size1([in, size=100] void* ptr);
public void test_size2([in, size=sz] void* ptr, size_t sz);
```

Note the following limitations/restrictions to avoid the related errors:

1. Pointers without a direction attribute or 'user_check' attribute are not allowed

```
public void test_ecall_not(int * ptr);
```

Error:

```
pointer/array should have direction attribute or `user_check'
```

2. User-defined pointer types must use 'isptr'

```
public void test_ecall_func([in]ptr_t ptr);
```

Error:

```
`ptr_t' is considered plain type but decorated with pointer attributes
```

3. Function pointers are not allowed

```
public void test_ecall_func([in]int (*func_ptr)());
```

Error:

```
parse error
```

Summary

Proper data handling/marshalling at the edge between the trusted/untrusted code parts plays a vital role in contributing to enclave's security. The security of an application's enclave with weak input/output boundary checks can be easily compromised.

With Intel SGX, boundary checking of input/output parameters is specified by the EDL file and translated into edge routines by the **sgx_edger8r** tool. Actual parameter checking is performed at runtime.

Developers should be aware of the crucial role in defining the input/output parameters and arguments to properly implement data handling/marshalling. They should especially understand that the responsibility of pointer validation is on them if they choose to use the `user_check` parameter.

References

1. Intel Software Guard Extensions SDK — 2017 Intel Corporation. https://software.intel.com/sgx-sdk/download.
2. Intel Software Guard Extensions SDK Users Guide for Windows OS Developer Guide — 2017 Intel Corporation. https://software.intel.com/sgx-sdk/documentation.
3. Intel Software Guard Extensions SDK for Windows OS Developer Reference — 2017 Intel Corporation. https://software.intel.com/sgx-sdk/documentation.
4. Intel Software Guard Extensions SDK Users Guide for Linux OS Developer Guide — 2017 Intel Corporation. https://software.intel.com/sgx-sdk/documentation.
5. Intel Software Guard Extensions SDK for Linux OS Developer Reference — 2017 Intel Corporation. https://software.intel.com/sgx-sdk/documentation.