INTEL-SA-00219 SGX SW Developer Guidance

Revision 1.0

2019-11-12

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. Check with your system manufacturer or retailer or learn more at www.intel.com.

Intel provides these materials "as is", with no express or implied warranties.

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.

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

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

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

1 Vulnerability Description 2

2 Mitigation 3

2.1 Custom Alignment Interfaces 3

2.1.1 Interfaces For Secrets Statically-Defined In C++ Code 3

2.1.2 Interfaces For Secrets Statically-Defined In Pure C Code 3

2.1.3 Interfaces For Secrets Dynamically-Defined In Either C Or C++ Code 5

2.2 Examples Of Use Of Custom Alignment Interfaces 5

3 Secrets In Structures That Can't Be Aligned 11

4 Aligning Secrets Defined In Assembly Code 15

Revision History

Revision Number	Description	Date
1.0	Initial revision	2019-11-12

1 Vulnerability Description

As described in INTEL-SA-00219, insufficient access control in the protected memory subsystem for Intel(R) Software Guard Extensions (Intel(R) SGX) on affected CPUs with Intel(R) Processor Graphics and Intel(R) SGX may allow system SW adversary to extract SGX enclave memory content residing in specific, affected memory locations.

2 Mitigation

To mitigate the issues described in INTEL-SA-00219, the platform owner should apply the latest BIOS from the system provider and the updated Intel(R) SGX Platform Software and disable the integrated processor graphics when possible. To defend against potential attacks on affected systems that can't disable the integrated processor graphics, the SGX SW application can implement SW mitigation inside the enclave, by organizing the code/data within enclave memory in order to avoid affected regions of a cache line.

The Intel(R) SGX SDK now provides interfaces that facilitate custom alignment of secrets and structures containing secrets. The SGX SDK version 2.7.101 for Linux and version 2.5.101 for Windows are the first versions with this support. There are different interfaces for secrets that are statically-defined (stack, global, static) versus dynamically-defined (heap) and for secrets in C++ code versus C code.

2.1 Custom Alignment Interfaces

This section lists and describes the custom alignment interfaces now in the Intel(R) SGX SDK.

2.1.1 Interfaces For Secrets Statically-Defined In C++ Code

The following class template, defined in the Intel(R) SGX SDK, makes it easy to align secrets that are statically-defined, for example, on the stack.

```
template<class T, std:size_t A, std::size_t… OLs>
class custom_alignment_aligned
```

T is the class/struct/type needing alignment, for example, a struct representing or containing a cryptographic key.

A is the desired, traditional alignment of T. This should not be confused with the alignment needed to mitigate the vulnerability. The two are related, but different. The examples below illustrate this difference.

OLs is a variable-length list of (offset, length) pairs. Each pair describes a secret within T. If T simply represents a single secret, then there will just be one pair, (0, sizeof(T)).

See Listing 1 for an example.

2.1.2 Interfaces For Secrets Statically-Defined In Pure C Code

The class template above can't be used in pure C code. Instead the following type and functions are used.

```
Typedef struct
{
```

```
    size_t offset;
    size_t len;
} align_req_t;
```

```
    void *sgx_get_aligned_ptr(void *raw,
                              size_t raw_size,
                              size_t allocate_size,
                              size_t alignment,
                              align_req_t *data,
                              size_t count);
```

Return address within provided (raw) buffer to be used as the starting address of the structure described by remaining parameters, in order for the structure to be aligned.

Parameters

raw – pointer to buffer allocated by the caller.

raw_size - the size of raw.

allocate_size - the size of the structure to be aligned.

alignment - the desired, traditional alignment of the structure to be aligned. Must be a power of 2.

data - (offset, length) pairs to define the fields in the structure needing confidentiality protection. If data is NULL and count is 0, the whole structure is treated as needing confidentiality protection. See examples below.

count – number of `align_req_t` structures in data

Return Value

Address within provided buffer where structure to be aligned should start or NULL, including case where structure can't be aligned – see below.

Note:

The raw buffer is defined/allocated by the caller. In general, its size (specified by raw_size parameter) needs to be bigger than the structure being aligned. How much bigger depends on value of desired, traditional alignment. Raw_size >= sizeof(structure) + 64 + A, where A = max((desired, traditional alignment), 8).

2.1.3 Interfaces For Secrets Dynamically-Defined In Either C Or C++ Code

```
void *sgx_aligned_malloc(size_t size,
                         size_t alignment,
                         align_req_t *data,
                         size_t count);
```

Allocates memory for a structure on a specified (traditional) alignment boundary and returns address where structure should start in order to be aligned.

Parameters

size - the size of the structure to be aligned.

Alignment - the desired, traditional alignment value of the structure. Must be a power of 2.

data - (offset, length) pairs to define the fields in the structure needing confidentiality protection. If data is NULL and count is 0, the whole structure is treated as needing confidentiality protection. See examples below.

count - number of `align_req_t` structures in data

Return Value

A pointerto the memory block that was allocated or NULL if the operation failed, including case where structure can't be aligned – see below.

```
void sgx_aligned_free(void *ptr);
```

Frees a block of memory that was allocated with `sgx_aligned_malloc`.

Parameters

ptr - pointer returned by `sgx_aligned_malloc`.

2.2 Examples Of Use Of Custom Alignment Interfaces

With the interfaces provided in the new Intel(R) SGX SDK, the best way to protect statically-defined secrets is shown below.

	`typedef unsigned char sgx_key_128bit_t[16];`

Unmitigated (original)	
	`sgx_key_128bit_t key;`
Mitigated	

	`sgx::custom_alignment_aligned<`
	`  sgx_key_128bit_t,  // type needing protection`
	`  sizeof(sgx_key_128bit_t), // desired alignment`
	`  0,  // offset of secret in type`
	`  sizeof(sgx_key_128bit_t)  // size of secret`
	`> okey;`
	`sgx_key_128bit_t& key = okey.v;`

Listing 1. Aligning Statically-Defined Secret In C++ Code

The second template parameter requires further description. It specifies the desired, *traditional* alignment of the structure containing the secret (or of the secret itself if the structure and the secret are one and the same, as above). For example, the structure above can be made to avoid any leaking for "desired alignment" parameter values of 1, 2, 4, 8, 16 and 32; it can't be aligned for values of 64 and higher. (Alignment, as used here, must be a power of 2.)

If a structure contains more than one secret, then the mitigated code simply includes a list of (offset, size) pairs as shown below.

```
struct SMultipleSecrets {
    typedef sgx_key_128bit_t mac_key;      // secret
    typedef sgx_key_128bit_t encrypt_key;  // secret
};
sgx::custom_alignment_aligned<
    SMultipleSecrets,
    16,
    __builtin_offsetof(SMultipleSecrets, mac_key),
    sizeof(mac_key),
    __builtin_offsetof(SMultipleSecrets, encrypt_key),
    sizeof(encrypt_key)
> Okeys;
SMultipleSecrets& keys = okeys.v;
```

Listing 2. Aligning Statically-Defined Structure That Has More Than One Secret In C++ Code

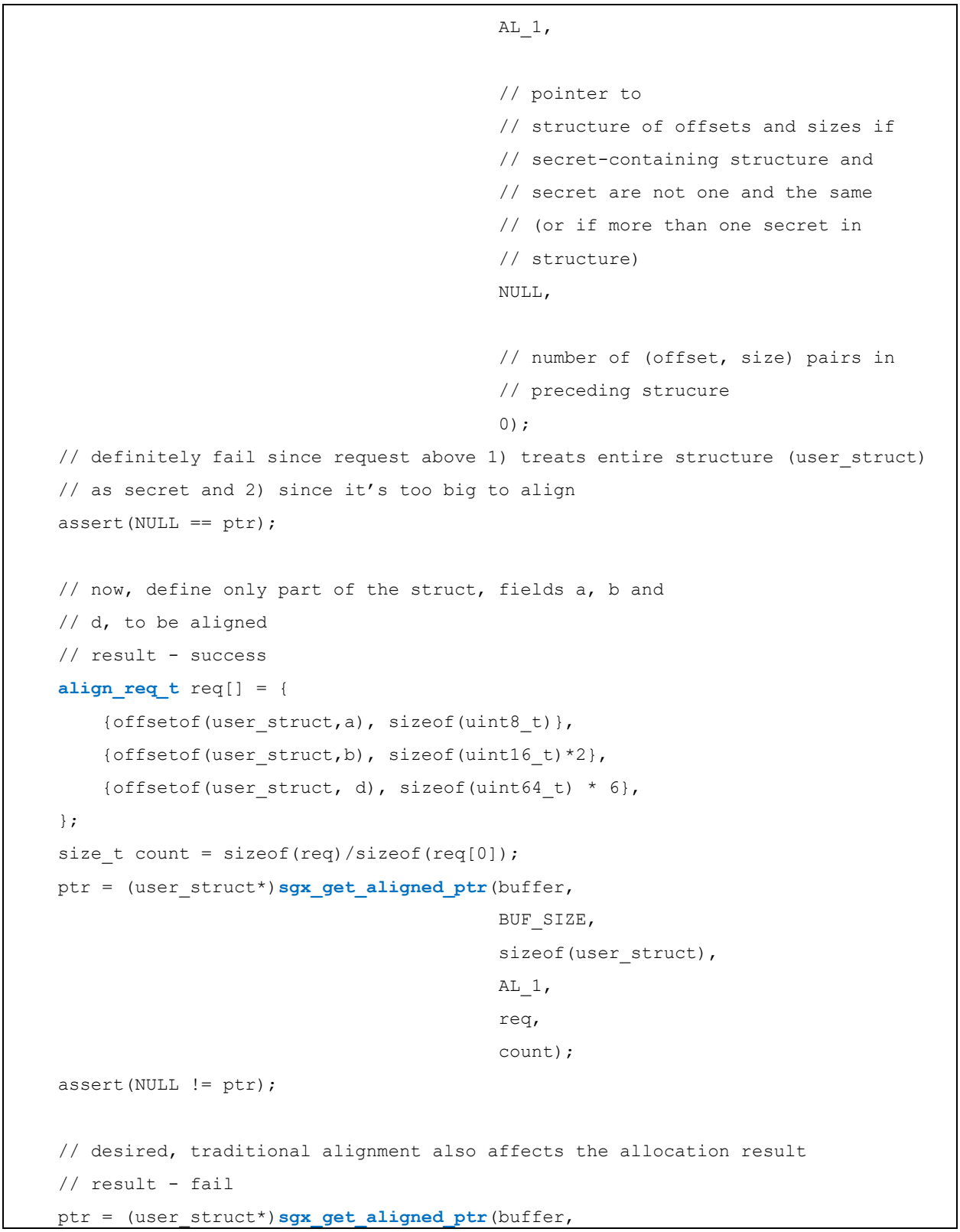
It's worth noting that many factors determine whether a structure is align-able: size(s) of secret(s), number of secrets, offsets of secrets and desired alignment. See section 3 for what to do if a given structure is not align-able.

Aligning statically-defined secrets in pure C code can be done as shown below.

```
const unsigned AL_1 = 1;
const unsigned AL_16 = 16;
typedef struct {
    uint8_t a;      // byte 0
    uint16_t b[2];  // byte 2 - 5
    uint32_t c;     // byte 8 - 11
    uint64_t d[6];  // byte 16 - 63
} user_struct;


user_struct *ptr = NULL;

// 80 = 64 + 16 below accounts for all traditional alignment
// values used below
const unsigned BUF_SIZE = sizeof(user_struct) + 80;
uint8_t buffer[BUF_SIZE];
```

Listing 3. Code Common To All Examples Below

```
// the whole struct needs to be aligned
// result - fail – cannot custom align a buffer with over 56 consecutive bytes
ptr = (user_struct*)sgx_get_aligned_ptr(buffer,
                                        BUF_SIZE,

                                        // size of structure containing secret
                                        // or of secret contents themselves
                                        sizeof(user_struct),

                                        // desired, traditional alignment
```



```
                                        AL_1,

                                        // pointer to
                                        // structure of offsets and sizes if
                                        // secret-containing structure and
                                        // secret are not one and the same
                                        // (or if more than one secret in
                                        // structure)
                                        NULL,

                                        // number of (offset, size) pairs in
                                        // preceding strucure
                                        0);
// definitely fail since request above 1) treats entire structure (user_struct)
// as secret and 2) since it's too big to align
assert(NULL == ptr);

// now, define only part of the struct, fields a, b and
// d, to be aligned
// result - success
align_req_t req[] = {
    {offsetof(user_struct,a), sizeof(uint8_t)},
    {offsetof(user_struct,b), sizeof(uint16_t)*2},
    {offsetof(user_struct, d), sizeof(uint64_t) * 6},
};
size_t count = sizeof(req)/sizeof(req[0]);
ptr = (user_struct*)sgx_get_aligned_ptr(buffer,
                                        BUF_SIZE,
                                        sizeof(user_struct),
                                        AL_1,
                                        req,
                                        count);
assert(NULL != ptr);

// desired, traditional alignment also affects the allocation result
// result - fail
ptr = (user_struct*)sgx_get_aligned_ptr(buffer,
```

```
                                        BUF_SIZE,
                                        sizeof(user_struct),

                                        // different desired, traditional
                                        // alignment value
                                        AL_16,
                                        req,
                                        count);
assert(NULL == ptr);
```

Listing 4. Aligning Statically-Defined Secrets In Pure C Code – Successful And Unsuccessful Cases

C++ callers are able to use either the class template approach in Listing 1 and Listing 2 or the approach that uses **sgx_get_aligned_ptr** in Listing 4. The class template approach is arguably better because an inability to align a structure will result in a build-time, not a run-time, error. Of course with the class template approach, all the template arguments must be constexpr per C++ specification.

Dynamically-defined secrets can be aligned using interfaces in the new SGX SDK. This is shown below. These interfaces are for both C and C++ code.

```
// the whole struct needs to be aligned
// result - fail - cannot custom align a buffer with over 56 consecutive bytes
ptr = (user_struct*)sgx_aligned_malloc(
                                        // size of structure containing secret
                                        // or of secret contents themselves
                                        sizeof(user_struct),

                                        // desired, traditional alignment
                                        AL_1,

                                        // pointer to
                                        // structure of offsets and sizes if
                                        // secret-containing structure and
                                        // secret are not one and the same
                                        // (or if more than one secret in
                                        // structure)
```

```
                                        NULL,

                                        // number of (offset, size) pairs in
                                        // preceding structure
                                        0);
    assert(NULL == ptr);

    // define only part of the struct, fields a, b and d, to be aligned
    // result - success
    align_req_t req[] = {
        {offsetof(user_struct, a), sizeof(uint8_t)},
        {offsetof(user_struct, b), sizeof(uint16_t)*2},
        {offsetof(user_struct, d), sizeof(uint64_t)*6},
    };
    size_t count = sizeof(req) / sizeof(req[0]);
    ptr = (user_struct*)sgx_aligned_malloc(sizeof(user_struct),
                                        AL_1,
                                        req,
                                        count);
    assert(NULL != ptr);
    sgx_aligned_free(ptr);

    // desired, traditional alignment also affects the allocation result
    // result - fail
    ptr = (user_struct*)sgx_aligned_malloc(sizeof(user_struct),

                                        // different desired, traditional
                                        // alignment value
                                        AL_16,
                                        req,
                                        count);
    assert(NULL == ptr);
```

Listing 5. Aligning Dynamically-Defined Secrets (C and C++) – Successful And Unsuccessful Cases

3 Secrets In Structures That Can't Be Aligned

As mentioned above, attempting to align a structure may fail for a combination of reasons. If alignment fails, but the failure is not due to a secret being too big, there are options, listed below.

1. Change the layout of the structure. The most general way to do this is to add padding before and/or after the secrets. Consider the following example.

Can't be aligned (original)	

	`Typedef struct {`
	`  sgx_key_128bit_t key1;`
	`  sgx_key_128bit_t key2;`
	`  sgx_key_128bit_t key3;`
	`  sgx_key_128bit_t key4;`
	`} struct_no_align_t;`

Can be aligned	

	`typedef struct {`
	`  sgx_key_128bit_t key1;`
	`  **unsigned char pad1[16];**`
	`  sgx_key_128bit_t key2;`
	`  **unsigned char pad2[16];**`
	`  sgx_key_128bit_t key3;`
	`  **unsigned char pad3[16];**`
	`  sgx_key_128bit_t key4;`
	`} struct_align_t;`

2. Change the desired alignment. For example

Can't be aligned	

	typedef struct { sgx_key_128bit_t key1; unsigned char pad1[16]; sgx_key_128bit_t key2; unsigned char pad2[16]; sgx_key_128bit_t key3; unsigned char pad3[16]; sgx_key_128bit_t key4; } struct_align_t; **sgx::custom_alignment_aligned**< struct_align_t, **// can't align the struct with** **// traditional** **// alignment of 32 bytes or** **// greater** **32,** __builtin_offsetof(struct_align_t, key1), sizeof(key1), __builtin_offsetof(struct_align_t, key2), sizeof(key2), __builtin_offsetof(struct_align_t, key3), sizeof(key3), __builtin_offsetof(struct_align_t, key4), sizeof(key4) > Okeys_no_align;
Can be aligned	
	sgx::custom_alignment_aligned< struct_align_t, **// can align the struct with traditional**

	// alignment of 16 bytes or less **16,** __builtin_offsetof(struct_align_t, key1), sizeof(key1), __builtin_offsetof(struct_align_t, key2), sizeof(key2), __builtin_offsetof(struct_align_t, key3), sizeof(key3), __builtin_offsetof(struct_align_t, key4), sizeof(key4) > Okeys_align;

3. Reduce the number of secrets in the structure. For example

Can't be aligned	
	Typedef struct { sgx_key_128bit_t key1; sgx_key_128bit_t key2; sgx_key_128bit_t key3; sgx_key_128bit_t key4; } struct_no_align_t;
Can be aligned	
	typedef struct { **sgx_key_128bit_t key1;** **sgx_key_128bit_t key2;** **} struct_align1_2_t;** **typedef struct {** **sgx_key_128bit_t key3;**

	sgx_key_128bit_t key4; } struct_align3_4_t;

4 Aligning Secrets Defined In Assembly Code

All of the approaches for aligning secrets described above can be applied to secrets defined in assembly code. This includes being able to use the following functions and types directly from assembly.

Functions	
	`sgx_get_aligned_ptr`
	`sgx_aligned_malloc`
	`sgx_aligned_free`
Types	
	`align_req_t`