WHITE PAPER

Intel® Software Guard Extensions (Intel® SGX)

Debugging Intel® Software Guard Extensions (Intel® SGX) Enclaves in Microsoft* Windows*

Scope

This paper describes the process for debugging Intel® Software Guard Extensions (Intel® SGX) enclaves for Microsoft* Windows*. The paper covers prerequisites and typical steps to debug an enclave using Microsoft Visual Studio*, the Intel SGX Debugger, and the Intel SGX debug API. Also included are examples of common errors that can occur in enclave code. This paper assumes a basic understanding of Intel SGX application development. Information on Intel SGX can be found on the Intel SGX portal at: https://software.intel.com/sgx.

Introduction

Much of the debugging process for Intel SGX enabled applications is similar to any other application developed for Windows. For example, the Windows Debugger can introspect application code in untrusted memory just as it can with typical (non-Intel SGX) applications. But the protected nature of Intel SGX enclaves prevents the Windows Debugger from introspecting enclave code.

However, the Intel SGX Debugger, an extension to the Windows Debugger included in the Intel SGX SDK, does provide debugging access to protected enclave code. The Intel SGX Debugger allows you to set breakpoints, step through enclave code, inspect/modify enclave variables, and output debug messages in enclaves from within Microsoft Visual Studio.

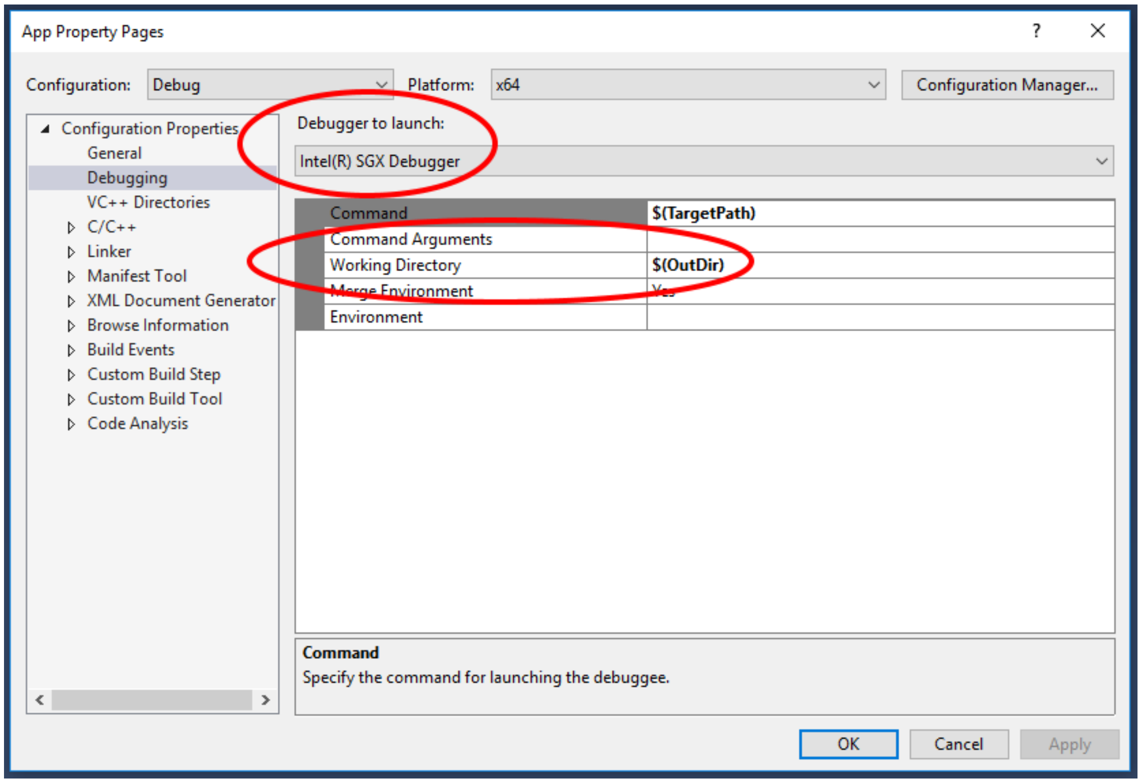
Debugging preconditions

Project setup

Specific settings must be configured to debug an Intel SGX enabled application. The settings are usually set by the Intel SGX SDK Visual Studio Wizard, but it's good to verify they have been done.

- The **Debugger to launch:** for the project must be `Intel(R) SGX Debugger`.
- The **Working Directory** for the application project must be set from the default of `$(ProjectDir)` to `$(OutDir)`. If the application is not set to `$(OutDir)`, the message `Error: Can't open enclave file.` is generated when the application tries to load the enclave.

These settings can be accessed using **Properties→Debugging**. Figure 1 shows these settings for the application.



Configuration: Debug

Platform: x64

Debugger to launch:

Intel(R) SGX Debugger

Command	$(TargetPath)
Command Arguments	
Working Directory	$(OutDir)
Merge Environment	Yes
Environment	

Command

Specify the command for launching the debuggee.

Figure 1. Project settings for debugging enclaves

Build configurations

There are three build configurations for Intel SGX applications:

Debug: Compiler optimizations are disabled and symbol information is saved. This mode is suitable for source-level debugging. Enclaves built in debug mode are launched in enclave-debug mode. *Only enclaves built in Debug mode can be debugged using the full capabilities of the Intel SGX Debugger.* (See below on Pre-Release debug limitations.)

Pre-Release: Compiler optimizations are enabled and symbols are not saved. But the enclave is set to debug mode so that production-level code can be debugged. Once your Debug mode enclave is working, you can rebuild the enclave in Pre-Release mode to apply compiler optimizations to look for performance issues and work out any final issues before moving to Release. Enclaves in Pre-Release mode can also be used with the Intel SGX Debugger, but since debug symbols are *not* saved and compiler optimizations *are* enabled, debug support is limited when compared with Debug mode enclaves.

Release: Compiler optimizations are enabled and no symbol information is saved. This mode is suitable for production build and final product release. Enclaves built in this mode are launched in enclave-production (non-debug) mode and cannot be introspected using the Intel SGX Debugger (or any debugger).

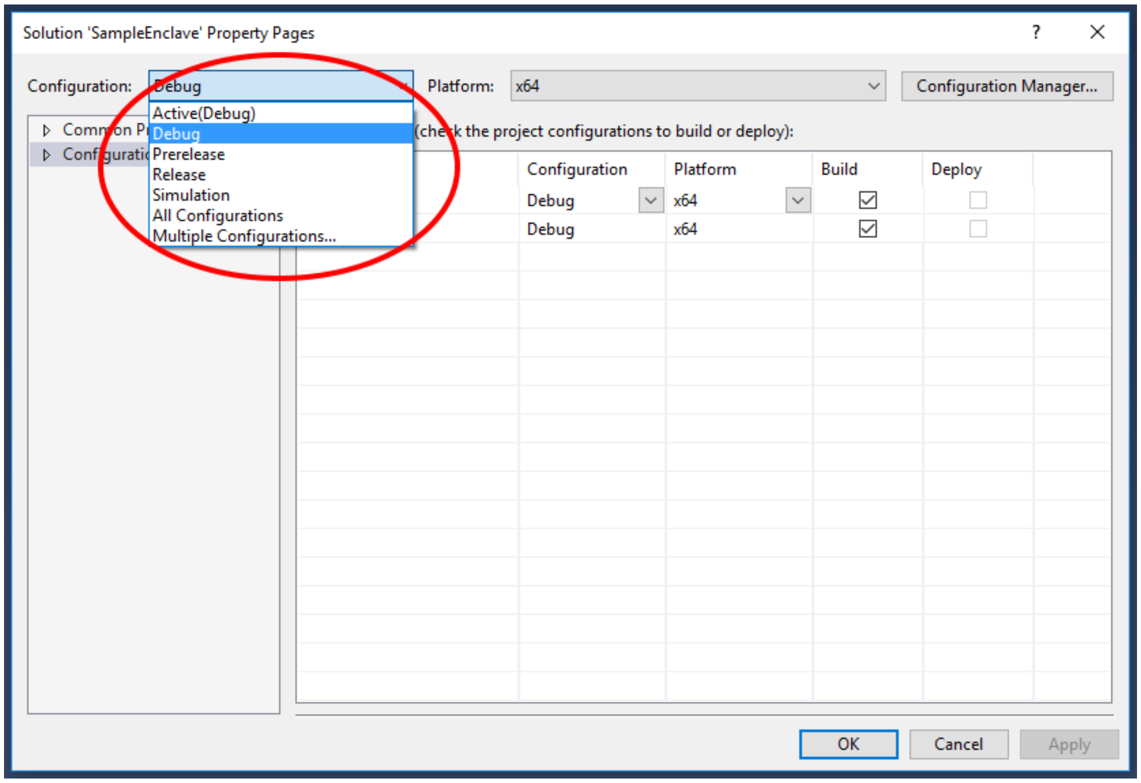
For details on the three configurations, see the following white paper: https://software.intel.com/sites/default/files/managed/e5/d8/intel-sgx-build-configuration.pdf.

Enclave debug setting

To debug an enclave, developers must:

1. Select a build configuration that allows debugging of enclaves (**Debug** or **Pre-Release** mode)
2. Launch the enclave in Debug mode

To accomplish this, configure the Visual Studio project to build the application in **Debug** (or **Pre-Release**) mode. In Visual Studio, select **Project Solutions→Properties→Configuration Properties→Configuration**.



(check the project configurations to build or deploy):

Configuration	Platform	Build	Deploy
Debug	x64	✓	☐
Debug	x64	✓	☐

Figure 2. Set the Configuration to Debug mode

This sets a helper macro in the underlying SDK call where the enclave is created, as follows:

```
ret = sgx_create_enclave(ENCLAVE_FILENAME, SGX_DEBUG_FLAG, &token,
                         &updated, &global_eid, NULL);
```

For Debug and Pre-Release builds, the helper macro `SGX_DEBUG_FLAG` is `1` to launch the enclave in debug mode. For Release builds, the helper macro `SGX_DEBUG_FLAG` is `0` to launch the enclave in non-debug mode.

You can also set the value of the debug parameter directly, as follows:

```
sgx_create_enclave(const char *file_name, const int debug,
            sgx_launch_token_t *launch_token, int *launch_token_updated,
            sgx_enclave_id_t *enclave_id, sgx_misc_attribute_t *misc_attr)
```

The `debug` parameter should be set to `1` to launch the enclave in debug mode and to `0` to launch the enclave in non-debug mode.

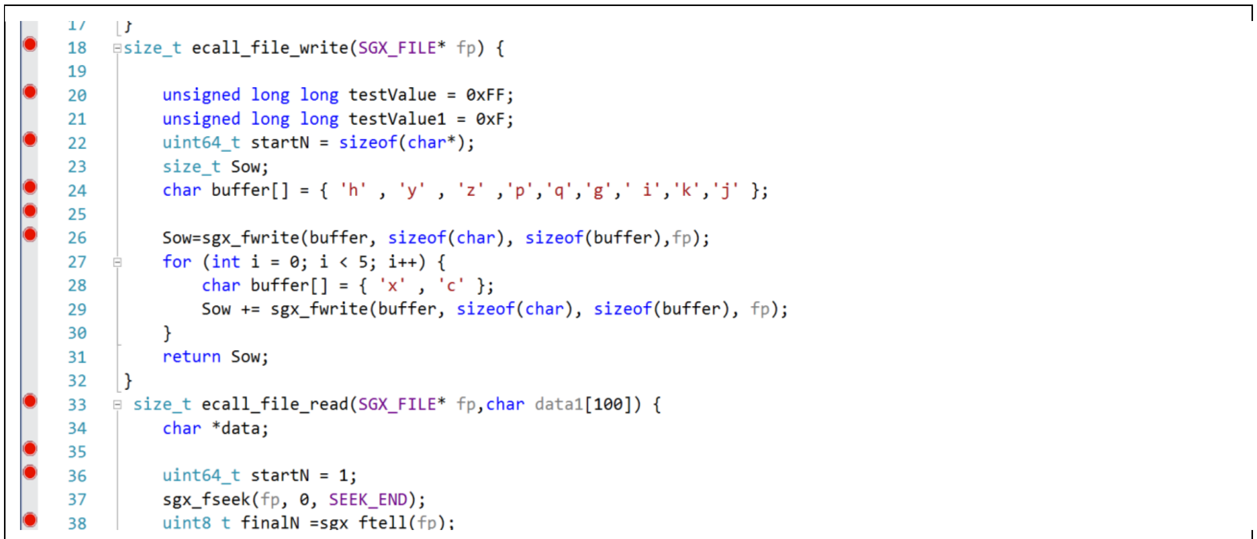
Debugging

Assuming the environment was configured correctly and the application and debug mode enclave were built successfully, you may have one or more issues that you suspect lie in the enclave code. So it's time to debug the enclave.

Standard breakpoints

First let's cover the use of standard Window Debugger breakpoints.

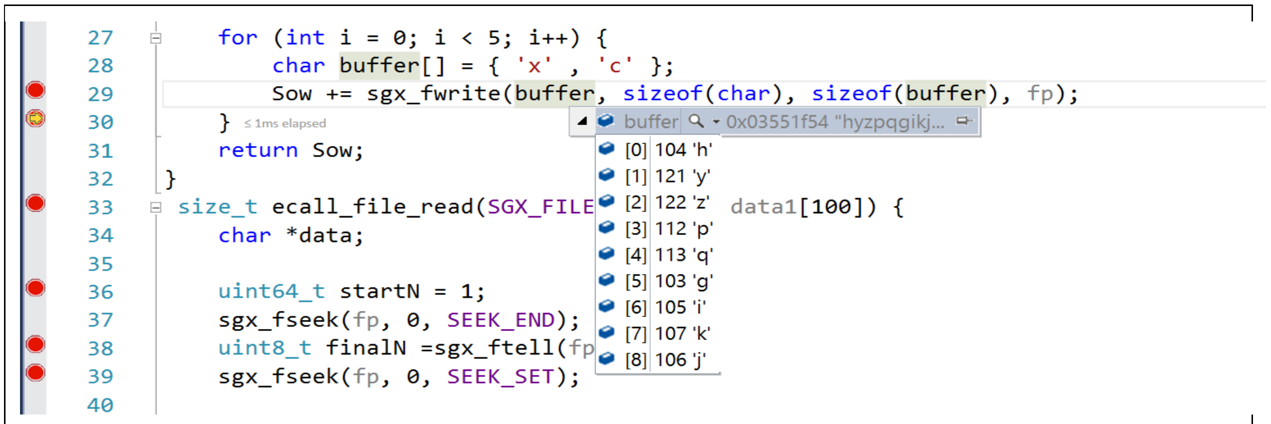
3. Verify that the `Intel(R) SGX Debugger` as the **Debugger to launch** for the project, and make sure that **Working Directory** is set to `$(OutDir)` for the application (shown in Figure 1). Note that the Windows Debugger can handle breakpoints in the *untrusted* part of the application, but cannot access breakpoints inside the *enclave*. Only the Intel® SGX Debugger can access breakpoints in the enclave.
4. Set break points in the enclave source code at the points where you want to inspect values. This allows you to check values of selected variables at specific points of time. Sample breakpoints are shown as red dots in the left column of Figure 3.



```
17  }
18  size_t ecall_file_write(SGX_FILE* fp) {
19
20      unsigned long long testValue = 0xFF;
21      unsigned long long testValue1 = 0xF;
22      uint64_t startN = sizeof(char*);
23      size_t Sow;
24      char buffer[] = { 'h' , 'y' , 'z' ,'p','q','g',' i','k','j' };
25
26      Sow=sgx_fwrite(buffer, sizeof(char), sizeof(buffer),fp);
27      for (int i = 0; i < 5; i++) {
28          char buffer[] = { 'x' , 'c' };
29          Sow += sgx_fwrite(buffer, sizeof(char), sizeof(buffer), fp);
30      }
31      return Sow;
32  }
33  size_t ecall_file_read(SGX_FILE* fp,char data1[100]) {
34      char *data;
35
36      uint64_t startN = 1;
37      sgx_fseek(fp, 0, SEEK_END);
38      uint8_t finalN =sgx_ftell(fp);
```

Figure 3. Sample code with breakpoints at lines 18, 20, etc.

5. Run the code using the Intel SGX Debugger. Execution pauses at each breakpoint, allowing you to inspect the values at that point by clicking on desired variable. Press **F5** to move through the code from one break point to the next. Use **F11** and **Shift+F11** to step into/out of nested code. Figure 4 shows inspection of the values for the variable `buffer`.



```
    for (int i = 0; i < 5; i++) {
        char buffer[] = { 'x' , 'c' };
        Sow += sgx_fwrite(buffer, sizeof(char), sizeof(buffer), fp);
    } ≤ 1ms elapsed
    return Sow;
}
size_t ecall_file_read(SGX_FILE                data1[100]) {
    char *data;

    uint64_t startN = 1;
    sgx_fseek(fp, 0, SEEK_END);
    uint8_t finalN =sgx_ftell(fp
    sgx_fseek(fp, 0, SEEK_SET);
```

```
buffer  0x03551f54 "hyzpqgikj...
[0] 104 'h'
[1] 121 'y'
[2] 122 'z'
[3] 112 'p'
[4] 113 'q'
[5] 103 'g'
[6] 105 'i'
[7] 107 'k'
[8] 106 'j'
```

Figure 4: Inspecting the value of variables during debugging

SGX Debug API

Now let's discuss Intel SGX Debugger support for three Windows APIs, which provide some additional capability to introspect enclaves. You must include the following header and library in your project configuration to use these APIs:

Header: `sgx_debug.h`

Library: `sgx_trts.lib` or `sgx_trts_sim.lib` (simulation)

IsDebuggerPresent

This call tests for the presence of the Intel SGX Debugger to prevent an application crash if `DebugBreak()` were included by itself, as follows:

```
if (IsDebuggerPresent()) {
    DebugBreak();
}
```

DebugBreak

`DebugBreak()` lets you programmatically insert a breakpoint in enclave code as shown in Figure 6. This can be useful in hard to reproduce situations where specific timing/resource conditions exist; for example, where something occurs only after *n* iterations of a loop.

> **Note:** `DebugBreak()` causes the application to crash (due to an illegal instruction) if the Intel® SGX Debugger is not attached. You should only include the call inside an `IsDebuggerPresent()` call.

Example1

```
    sgx_status_t ret = sgx_cpuid(cpuinfo, leaf);
    if (ret != SGX_SUCCESS)
    {
        if (IsDebuggerPresent())
        {
            // sgx_cpuid should not fail.
            DebugBreak();
        }
        else
        {
            abort();
        }
    }
```

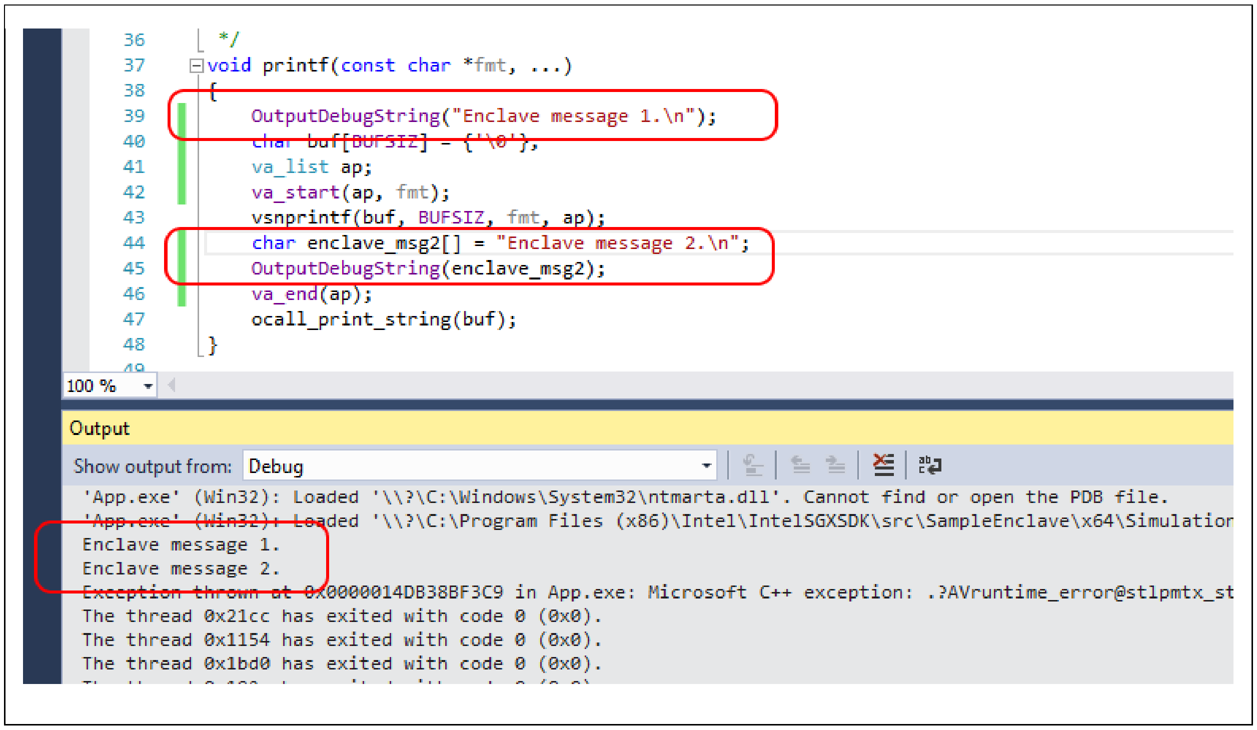
Example2

```
         for (int i = 0; i < 1000; i++)
         {
             if (i == 999 && IsDebuggerPresent())
             {
                 DebugBreak();
             }
         }
```

Figure 5. Example programmatic use of DebugBreak()

OutputDebugString

As shown in the code sample in Figure 6, the `OutputDebugString()` function lets you send a string to the debugger output. If the Intel® SGX Debugger is not attached, nothing happens; no message is sent and the application continues past the call.



```
 */
void printf(const char *fmt, ...)
{
    OutputDebugString("Enclave message 1.\n");
    char buf[BUFSIZ] = {'\0'};
    va_list ap;
    va_start(ap, fmt);
    vsnprintf(buf, BUFSIZ, fmt, ap);
    char enclave_msg2[] = "Enclave message 2.\n";
    OutputDebugString(enclave_msg2);
    va_end(ap);
    ocall_print_string(buf);
}
```

```
'App.exe' (Win32): Loaded '\\?\C:\Windows\System32\ntmarta.dll'. Cannot find or open the PDB file.
'App.exe' (Win32): Loaded '\\?\C:\Program Files (x86)\Intel\IntelSGXSDK\src\SampleEnclave\x64\Simulatior
Enclave message 1.
Enclave message 2.
Exception thrown at 0x0000014DB38BF3C9 in App.exe: Microsoft C++ exception: .?AVruntime_error@stlpmtx_st
The thread 0x21cc has exited with code 0 (0x0).
The thread 0x1154 has exited with code 0 (0x0).
The thread 0x1bd0 has exited with code 0 (0x0).
```

Figure 6. OutputDebugString usage example

Common issues during debugging

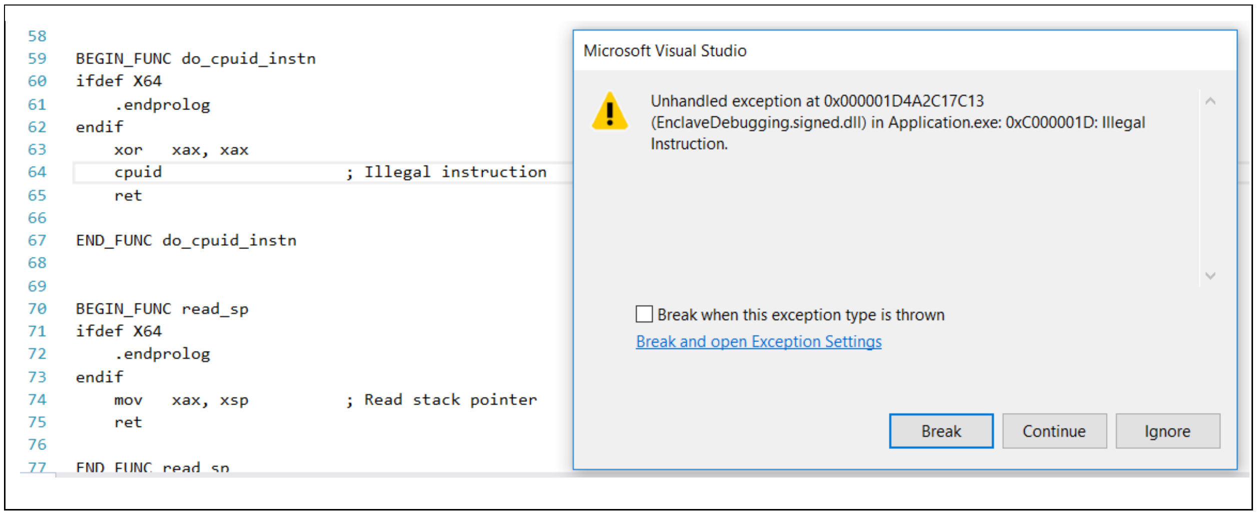
This section summarizes common errors you may see in enclaves during debugging.

Notes:

- The code used to generated error examples is not real-world code. It was written to intentionally generate the errors described.
- Certain exceptions are reported back to the enclave, which, therefore, provides an opportunity to handle them by including exception handling code.

Illegal Instruction in enclave

Intel SGX prohibits execution of CPU instructions inside an enclave that would gather host system attributes, perform I/O, or require a higher privilege level than ring 3. When code that depends on underlying illegal HW instructions attempts to execute, the enclave aborts with an Invalid Opcode Exception (#UD) at the machine level. See Figure 7 for the Visual Studio exception message that is based on the machine level exception. For a list of instructions that are illegal in enclaves, see Section 39.6.1 in Intel 64- and 32-bit Architectures Software Developer's Manual, Volume 3D, Part 4.



```
BEGIN_FUNC do_cpuid_instn
ifdef X64
    .endprolog
endif
    xor   xax, xax
    cpuid                    ; Illegal instruction
    ret

END_FUNC do_cpuid_instn


BEGIN_FUNC read_sp
ifdef X64
    .endprolog
endif
    mov   xax, xsp           ; Read stack pointer
    ret

END_FUNC read_sp
```

Microsoft Visual Studio

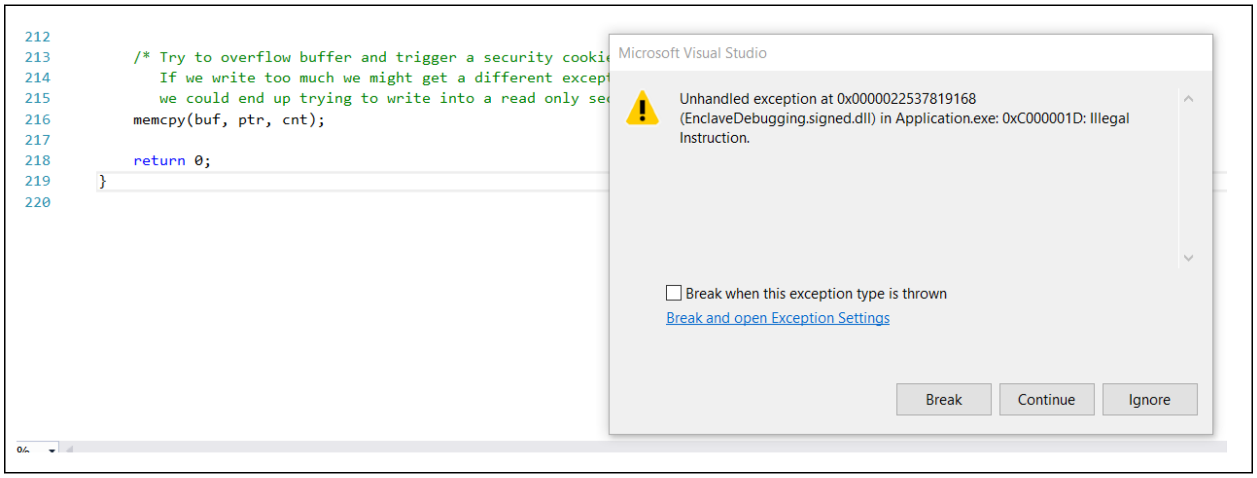
Unhandled exception at 0x000001D4A2C17C13 (EnclaveDebugging.signed.dll) in Application.exe: 0xC000001D: Illegal Instruction.

Break when this exception type is thrown
Break and open Exception Settings
Break
Continue
Ignore

Figure 7. Example CPUID (illegal) instruction in enclave

Buffer overflow in enclave

While Intel SGX SDK does not prevent developers from writing enclave code that can overflow a buffer, it does support runtime security features that help detect buffer overflows. The enclave aborts (based on a UD2 instruction at the machine level) with the Visual Studio error message shown in Figure 8 when a potential buffer overflow is detected.



```
        /* Try to overflow buffer and trigger a security cooki
           If we write too much we might get a different except
           we could end up trying to write into a read only se
        memcpy(buf, ptr, cnt);

        return 0;
    }
```

Microsoft Visual Studio

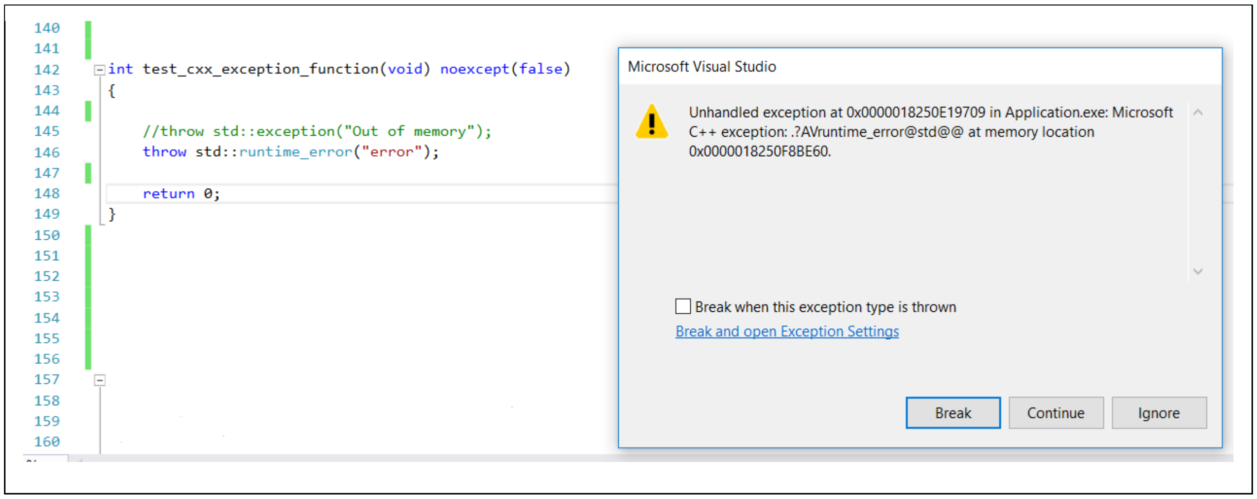
Unhandled exception at 0x0000022537819168 (EnclaveDebugging.signed.dll) in Application.exe: 0xC000001D: Illegal Instruction.

Break when this exception type is thrown
Break and open Exception Settings
Break
Continue
Ignore

Figure 8. Example buffer overflow in enclave

C++ runtime exception thrown in enclave

If a C++ runtime exception occurs in enclave code and it is not caught, the enclave aborts and a C++ exception is reported, as shown in Figure 9. For help understanding the C++ runtime exception in Windows, see: https://docs.microsoft.com/en-us/cpp/cppcx/exceptions-c-cx.



```
int test_cxx_exception_function(void) noexcept(false)
{

    //throw std::exception("Out of memory");
    throw std::runtime_error("error");

    return 0;
}
```

Microsoft Visual Studio

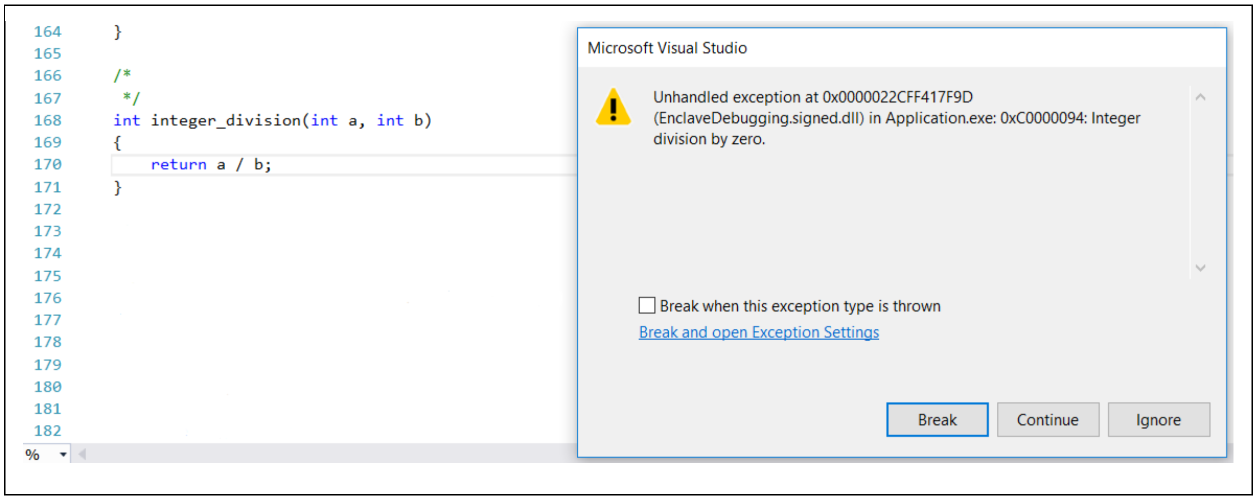
Unhandled exception at 0x0000018250E19709 in Application.exe: Microsoft C++ exception: .?AVruntime_error@std@@ at memory location 0x0000018250F8BE60.

Break when this exception type is thrown
Break and open Exception Settings
Break
Continue
Ignore

Figure 9. Example C++ runtime exception in enclave

Uncaught exception thrown in enclave

When an uncaught exception is thrown in an enclave, the enclave aborts with the message shown in Figure 10. The test case for generating this message is divide-by-zero.



```
}

/*
 */
int integer_division(int a, int b)
{
    return a / b;
}
```

Microsoft Visual Studio

Unhandled exception at 0x0000022CFF417F9D (EnclaveDebugging.signed.dll) in Application.exe: 0xC0000094: Integer division by zero.

Break when this exception type is thrown
Break and open Exception Settings
Break
Continue
Ignore

Figure 10. Example uncaught exception in enclave

Intel SGX enclave recovery (multiple threads)

In multi-threaded applications, it is important that only one thread perform the recovery of an enclave lost due to power events. If more than one thread could validly be responsible for attempted to handle the SGX_ERR_ENCLAVE_LOST message on an application abort, logic must be created that decides which thread will perform the actual recovery. If this is not done, the application can get trapped in an endless chain of enclave recovery operations. For details, see https://software.intel.com/articles/intel-sgx-tutorial-part-9-power-events-and-data-sealing.

Summary

The combination of the Intel SGX Debugger for Microsoft Visual Studio* 2015, and the Intel SGX Debug and Pre-Release mode capabilities, allow you to debug enclaves for Microsoft Windows applications.

Prepare to debug an enclave by configuring Visual Studio properly and, if appropriate, adding Debug API calls.

Build/execute the enclave in Debug mode (or Simulation mode) and step through the code with the Intel SGX Debugger, using breakpoints and messages to identify and correct errors. The Debug compilation profile (or Simulation profile) allows the full capabilities of the Intel SGX Debugger to be used with your enclaves.

References

1. Intel Software Guard Extensions SDK Users Guide for Windows OS — 2017 Intel Corporation. https://software.intel.com/sgx-sdk/documentation.
2. Intel Software Guard Extensions SDK Developer Reference for Windows OS — 2017 Intel Corporation. https://software.intel.com/sgx-sdk/documentation.
3. Intel SGX Forums. https://software.intel.com/forums/intel-software-guard-extensions-intel-sgx.