Debugging Intel® Software Guard Extensions (Intel® SGX) Enclaves for Linux* OS

Scope

This paper describes the process for debugging an Intel® Software Guard Extensions (Intel® SGX) enclave for the Linux* OS using the GDB debugger. The paper covers prerequisites and typical steps to debug an enclave using the GDB debugger with the Intel SGX GDB "plugin" from the Intel SGX SDK for Linux. Also included are descriptions of common errors that can occur in enclave code. This paper assumes a basic understanding of Intel SGX. 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 Linux. For example, the GDB debugger can introspect application code in trusted memory in the same way as with typical (non-SGX) applications. But the protected nature of Intel SGX enclaves prevents the GDB debugger from introspecting enclave code.

However, the Intel SGX GDB "plugin" (**sgx-gdb**) included in the Intel SGX SDK, does provide debugging access to protected enclave code. **sgx-gdb** allows you to set breakpoints, inspect/modify enclave variables, and step through enclave code.

Debugging preconditions

Project setup

Make sure the following software has been downloaded, built, and installed (open-source links included):

- Intel® SGX Linux driver (https://github.com/01org/linux-sgx-driver)
- Intel® SGX Platform SW (https://github.com/01org/linux-sgx)
- Intel® SGX SDK (same GitHub project as Platform SW above)

Another options is to download pre-built SDK and PSW installer binaries at: https://01.org/intel-software-guard-extensions/downloads.

If you can build the code samples in the SDK, then your SDK configuration is ready to run **sgx-gdb**. If not, debug your SDK installation so you can run the SDK code samples.

Build configurations

The Intel SGX architecture supports two modes of operation for enclaves: *Debug* mode and *Production* (non-debug) mode. Production mode enclaves have the full protection provided by the architecture. Debug mode enclaves do not have the full protection to allow introspection by debuggers.

There are three build configurations for Intel SGX enabled 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. *Enclaves built in Debug mode can be debugged using the full capabilities of the* ***sgx-gdb*.* (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 **sgx-gdb**, 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, performance testing, and final product release. Enclaves built in this mode are launched in enclave-production (non-debug) mode and *cannot* be introspected using **sgx-gdb** (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

By default, the `Makefile` sets parameters to build enclaves in Debug mode, which allows debugging. Supported `Makefile` parameters are listed in Table 1.

Table 1. Intel SGX Makefile Command Parameters

Build Configuration	Make Command Parameters
*Debug	SGX_MODE=HW SGX_DEBUG=1
Pre-Release	SGX_MODE=HW SGX_DEBUG=0 SGX_PRERELEASE=1
Release	SGX_MODE=HW SGX_DEBUG=0
Simulation	SGX_MODE=SIM SGX_DEBUG=1
*Debug configuration is the default (SGX_MODE=HW, SGX_DEBUG=1).	

The build configuration in the `Makefile` 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 above should be set to `1` to launch the enclave in debug mode and to `0` to launch the enclave in non-debug mode.

GDB debugger for enclaves

GDB must be invoked with **sgx-gdb** to allow introspection of enclave code.

Debugging an enclave is similar to debugging a shared library. But some standard GDB features are not available for use inside enclave code. Table 1 lists some *unsupported* GDB commands for enclaves.

Table 1. GDB commands not available inside enclaves

GDB Command	Notes
info sharedlibrary	Does not show the status of the loaded enclave
next/step	Does not allows to execute the next/step outside the enclave from inside the enclave. To go outside the enclave use the `finish` command.
call/print	Does not support calling outside the enclave from within an enclave function, or calling inside the enclave from a function in the untrusted domain.
charset	Only supports GDB's default charset.
gcore	Does not support debugging an enclave with the application dump file

Note: In addition to supporting GDB functions, the **sgx-gdb** "plugin" also supports measuring peak stack/heap usage of enclave code by invoking the Enclave Memory Measurement Tool (EMMT). Information on the EMMT tool is provided later in this paper.

Steps to debug an enclave

The sequence in this section is based on the `LocalAttestation` sample code distributed with the SDK. Follow these steps to debug an Intel SGX enclave:

1. Make sure you can build and run the SDK `SampleCode` applications.
2. Compile and run the `SampleCode/LocalAttestion` application in Debug mode.

   Once the build is complete, invoke the GDB debugger by running the **sgx-gdb** script, as shown in Figure 1 (user input in **blue**). This invokes GDB, loads the **sgx-gdb** "plugin", and loads application symbols that were saved during the build.

```
<user>projects/linux-sgx/SampleCode/LocalAttestation$ sgx-gdb ./app
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Source directories searched: /opt/intel/sgxsdk/lib64/gdb-sgx-plugin:$cdir:$cwd
Setting environment variable "LD_PRELOAD" to null value.
Reading symbols from ./app...done.
(gdb)
```

Figure 1: sgx-gdb invocation

3. Now set two breakpoints inside the enclave code (`break`/`b` command), as shown in Figure 2.

```
(gdb) b test_message_exchange
Function "test_message_exchange" not defined.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 1 (test_message_exchange) pending.
(gdb) b test_enclave_to_enclave_call
Function "test_enclave_to_enclave_call" not defined.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 2 (test_enclave_to_enclave_call) pending.
(gdb)
```

Figure 2: Setting sgx-gdb breakpoints

4. Run the application (`run`/`r` command), as shown in Figure 3. Execution halts at the first breakpoint.

```
gdb) run
Starting program: /home/dsmith/projects/linux-sgx/SampleCode/LocalAttestation/app
detect urts is loaded, initializing
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
add-symbol-file '/home/dsmith/projects/linux-sgx/SampleCode/LocalAttestation/libenclave1.so'
0x7ffff60f8640 -readnow -s .interp 0x7ffff60f6238  -s .note.gnu.build-id 0x7ffff60f6254  -s
.gnu.hash 0x7ffff60f6278  -s .dynsym 0x7ffff60f62b0  -s .dynstr 0x7ffff60f6358  -s
.gnu.version 0x7ffff60f63a4  -s .gnu.version_d 0x7ffff60f63b8  -s .rela.dyn 0x7ffff60f63f0  -
s .rodata 0x7ffff61ac000  -s .eh_frame_hdr 0x7ffff61b1f80  -s .eh_frame 0x7ffff61b3708  -s
.gcc_except_table 0x7ffff61c4068  -s .init_array 0x7ffff63c4b68  -s .fini_array
0x7ffff63c4b78  -s .data.rel.ro 0x7ffff63c4bc0  -s .dynamic 0x7ffff63c4e40  -s .got
0x7ffff63c4fc0  -s .got.plt 0x7ffff63c5000  -s .data 0x7ffff63c5020  -s .bss 0x7ffff63c6540
add-symbol-file '/home/dsmith/projects/linux-sgx/SampleCode/LocalAttestation/libenclave2.so'
0x7ffff58f8640 -readnow -s .interp 0x7ffff58f6238  -s .note.gnu.build-id 0x7ffff58f6254  -s
.gnu.hash 0x7ffff58f6278  -s .dynsym 0x7ffff58f62b0  -s .dynstr 0x7ffff58f6358  -s
.gnu.version 0x7ffff58f63a4  -s .gnu.version_d 0x7ffff58f63b8  -s .rela.dyn 0x7ffff58f63f0  -
s .rodata 0x7ffff59ab000  -s .eh_frame_hdr 0x7ffff59b0f80  -s .eh_frame 0x7ffff59b2708  -s
.gcc_except_table 0x7ffff59c3068  -s .init_array 0x7ffff5bc3b68  -s .fini_array
0x7ffff5bc3b78  -s .data.rel.ro 0x7ffff5bc3bc0  -s .dynamic 0x7ffff5bc3e40  -s .got
0x7ffff5bc3fc0  -s .got.plt 0x7ffff5bc4000  -s .data 0x7ffff5bc4020  -s .bss 0x7ffff5bc5540
```

```
add-symbol-file '/home/dsmith/projects/linux-sgx/SampleCode/LocalAttestation/libenclave3.so'
0x7ffff50f8640 -readnow -s .interp 0x7ffff50f6238  -s .note.gnu.build-id 0x7ffff50f6254  -s
.gnu.hash 0x7ffff50f6278  -s .dynsym 0x7ffff50f62b0  -s .dynstr 0x7ffff50f6358  -s
.gnu.version 0x7ffff50f63a4  -s .gnu.version_d 0x7ffff50f63b8  -s .rela.dyn 0x7ffff50f63f0  -
s .rodata 0x7ffff51ac000  -s .eh_frame_hdr 0x7ffff51b1f80  -s .eh_frame 0x7ffff51b3708  -s
.gcc_except_table 0x7ffff51c4068  -s .init_array 0x7ffff53c4b68  -s .fini_array
0x7ffff53c4b78  -s .data.rel.ro 0x7ffff53c4bc0  -s .dynamic 0x7ffff53c4e40  -s .got
0x7ffff53c4fc0  -s .got.plt 0x7ffff53c5000  -s .data 0x7ffff53c5020  -s .bss 0x7ffff53c6540

Avaliable Enclaves
Enclave1 - EnclaveID 432500000002
Enclave2 - EnclaveID 432500000003
Enclave3 - EnclaveID 432500000004

Secure Channel Establishment between Source (E1) and Destination (E2) Enclaves successful !!!


Breakpoint 1, 0x00007ffff60fa3a4 in test_message_exchange (src_enclave_id=0,
dest_enclave_id=0) at Enclave1/Enclave1.cpp:149
149     {
(gdb)
```

Figure 3: sgx-gdb breakpoint hit

Backtrace from the current execution frame using the `backtrace`/`bt` command, as shown in Figure 4.

```
(gdb) bt
#0  0x00007ffff60fa3a4 in test_message_exchange (src_enclave_id=0, dest_enclave_id=0) at
Enclave1/Enclave1.cpp:149
#1  0x00007ffff60f89c4 in sgx_test_message_exchange (pms=0x7fffffffe2b0) at Enclave1/Enclave1_t.c:169
#2  0x00007ffff60fd679 in do_ecall ()
#3  0x00007ffff60fd4d8 in enter_enclave ()
#4  0x00007ffff60ff1eb in enclave_entry ()
#5  0x00007ffff7bb1f3c in __morestack () from /opt/intel/sgxsdk/sdk_libs/libsgx_urts_sim.so
#6  0x00007ffff7baf687 in do_ecall(int, void const*, void const*, CTrustThread*) () from
/opt/intel/sgxsdk/sdk_libs/libsgx_urts_sim.so
#7  0x00007ffff7baae99 in CEnclave::ecall(int, void const*, void*) () from
/opt/intel/sgxsdk/sdk_libs/libsgx_urts_sim.so
#8  0x00007ffff7bae0e2 in sgx_ecall () from /opt/intel/sgxsdk/sdk_libs/libsgx_urts_sim.so
#9  0x0000000000401222 in Enclave1_test_message_exchange (eid=73826192850946, retval=0x7fffffffe2f0,
src_enclave_id=73826192850946, dest_enclave_id=73826192850947) at Enclave1/Enclave1_u.c:237
#10 0x0000000000402791 in main (argc=1, argv=0x7fffffffe3e8) at App/App.cpp:168
(gdb)
```

Figure 4: sgx-gdb back trace

Here are some useful GDB commands (examples not shown):

- The `parameter`/`p <parameter name>` shows the current state of a parameter.
- The `continue`/`c` command moves execution to the next breakpoint.
- The `frame`/`f` command displays the current **sgx-gdb** frame.
- The `step`/`s` command executes until the line of source code changes (`step` executes into a function call).

5. The `next`/`n` command executes over a function calls to the next line of code, as shown in Figure 5, but does *not* trace into a function call.

```
(gdb) n

Breakpoint 1, test_message_exchange (src_enclave_id=73826192850946,
dest_enclave_id=73826192850947) at Enclave1/Enclave1.cpp:149
149     {
(gdb) n
150         ATTESTATION_STATUS ke_status = SUCCESS;
(gdb) n
161         target_fn_id = 0;
(gdb) n
162         msg_type = MESSAGE_EXCHANGE;
(gdb) n
163         max_out_buff_size = 50;
(gdb) n
164         secret_data = 0x12345678; //Secret Data here is shown only for purpose of
demonstration.
(gdb) n
167         ke_status = marshal_message_exchange_request(target_fn_id, msg_type, secret_data,
&marshalled_inp_buff, &marshalled_inp_buff_len);
(gdb) n
168         if(ke_status != SUCCESS)
(gdb) n
173         std::map<sgx_enclave_id_t, dh_session_t>::iterator it =
g_src_session_info_map.find(dest_enclave_id);
(gdb) n
174         if(it != g_src_session_info_map.end())
(gdb)
```

Figure 5: sgx-gdb next/n command

Note: Neither `step/s` nor `next/n` allows execution to exit the enclave.

6. To step outside the enclave, use the `finish` command, as shown in Figure 6.

```
(gdb) finish
Run till exit from #0  test_message_exchange (src_enclave_id=73826192850946,
dest_enclave_id=73826192850947) at Enclave1/Enclave1.cpp:174
0x00007ffff60f89c4 in sgx_test_message_exchange (pms=0x7fffffffe2b0) at
Enclave1/Enclave1_t.c:169
169             ms->ms_retval = test_message_exchange(ms->ms_src_enclave_id, ms-
>ms_dest_enclave_id);
Value returned is $1 = 0
(gdb) finish
Run till exit from #0  0x00007ffff60f89c4 in sgx_test_message_exchange (pms=0x7fffffffe2b0)
at Enclave1/Enclave1_t.c:169
0x00007ffff60fd679 in do_ecall ()
Value returned is $2 = SGX_SUCCESS
(gdb) finish
Run till exit from #0  0x00007ffff60fd679 in do_ecall ()
0x00007ffff60fd4d8 in enter_enclave ()
(gdb)
```

Figure 6: sgx-gdb finish command

Notes:

- You can measure peak stack/heap usage by enabling the **sgx_emmt** tool in **sgx-gdb**. Follow these steps:

  - Make sure that your SGX application has the symbol `g_peak_heap_used` in the global section of the enclave linker script. For an introduction to this script, see *Creating Intel Software Guard Extensions Projects* in the *Intel SGX SDK Developers Reference*.
  - In **sgx-gdb**, use the `enable sgx_emmt` command before running the app. Figure 7 shows the results of **sgx_emmt** when run in **sgx-gdb**. (`disable sgx_emmt` disables the **sgx_emmt** tool.)
- Details on the **sgx_emmt** tool are provide at: https://software.intel.com/sites/default/files/managed/09/37/Enclave-Measurement-Tool-Intel-SGX.pdf.

```
(Previous screen display edited)

Close Session between Source (E2) and Destination (E3) Enclaves successful !!!

Enclave: "/home/dsmith/projects/linux-sgx/SampleCode/LocalAttestation/libenclave
  [Peak stack used]: 8 KB
  [Peak heap used]:  16 KB
remove-symbol-file -a 140737321600576
Enclave: "/home/dsmith/projects/linux-sgx/SampleCode/LocalAttestation/libenclave
  [Peak stack used]: 8 KB
  [Peak heap used]:  16 KB
remove-symbol-file -a 140737313211968
Enclave: "/home/dsmith/projects/linux-sgx/SampleCode/LocalAttestation/libenclave
  [Peak stack used]: 8 KB
  [Peak heap used]:  16 KB
remove-symbol-file -a 140737304823360
Close Session between Source (E3) and Destination (E1) Enclaves successful !!!

Hit a key....
```

Figure 7: sgx-gdb executing showing sgx_emmt results

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 8 for the Linux exception message and **sgx-gdb** backtrace 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.

```
Program received signal SIGILL, Illegal instruction.
do_cpuid_instn () at Enclave/illegal_instruction.S:59
59          cpuid                   /* Illegal instruction */
=> 0x00007ffff5401ea8 <do_cpuid_instn+3>:       0f a2   cpuid
(gdb) bt
#0  do_cpuid_instn () at Enclave/illegal_instruction.S:59
#1  0x00007ffff5401dbe in test_illegal_instructions_ecall (op_type=2) at
Enclave/Enclave.cpp:189
#2  0x00007ffff54017a2 in sgx_test_illegal_instructions_ecall (pms=0x7fffffffde10)
at Enclave/Enclave_t.c:101
#3  0x00007ffff5403740 in enter_enclave ()
#4  0x00007ffff5404c96 in enclave_entry ()
#5  0x00007ffff7bbb43b in __morestack () from /usr/lib/libsgx_urts.so
#6  0x00007ffff7bbe227 in CEnclave::ecall(int, void const*, void*) () from
/usr/lib/libsgx_urts.so
#7  0x00007ffff7bbfb22 in sgx_ecall () from /usr/lib/libsgx_urts.so
#8  0x0000000000400f0a in test_illegal_instructions_ecall (eid=2,
retval=0x7fffffffde44, op_type=2) at App/Enclave_u.c:99
#9  0x00000000004016fd in main (argc=1, argv=0x7fffffffe158) at App/App.cpp:555
(gdb)
```

Figure 8. 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 Linux error message shown in Figure 9 when a buffer overflow is detected. Also shown is **sgx-gdb** backtrace data for the error.

```
Program received signal SIGILL, Illegal instruction.
0x00007ffff5004e67 in abort ()
(gdb) bt
#0  0x00007ffff5004e67 in abort ()
#1  0x00007ffff5004f39 in __stack_chk_fail ()
#2  0x00007ffff5001eb0 in test_buffer_overflow_ecall (sync_flag=0x6032e8
<sync_flag>, ptr=0x7ffff5245010 "", cnt=528) at Enclave/Enclave.cpp:234
#3  0x00007ffff50018c5 in sgx_test_buffer_overflow_ecall (pms=0x7fffffffddf0) at
Enclave/Enclave_t.c:139
#4  0x00007ffff50037d0 in enter_enclave ()
#5  0x00007ffff5004be6 in enclave_entry ()
#6  0x00007ffff7bbb43b in __morestack () from /usr/lib/libsgx_urts.so
#7  0x00007ffff7bbe227 in CEnclave::ecall(int, void const*, void*) () from
/usr/lib/libsgx_urts.so
#8  0x00007ffff7bbfb22 in sgx_ecall () from /usr/lib/libsgx_urts.so
#9  0x0000000000400e7e in test_buffer_overflow_ecall (eid=2, retval=0x7fffffffde44,
sync_flag=0x6032e8 <sync_flag>, ptr=0x7fffffffde50 "", cnt=528) at
App/Enclave_u.c:115
#10 0x000000000040166e in main (argc=1, argv=0x7fffffffe158) at App/App.cpp:668
(gdb)
```

Figure 9. 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, as shown in Figure 10.

```
Program received signal SIGILL, Illegal instruction.
0x00007ffff5004d97 in abort ()
(gdb) bt
#0  0x00007ffff5004d97 in abort ()
#1  0x00007ffff501e65b in std::terminate() ()
#2  0x00007ffff501e86a in report_failure(_Unwind_Reason_Code,
__cxxabiv1::__cxa_exception*) ()
#3  0x00007ffff501e8bd in throw_exception(__cxxabiv1::__cxa_exception*) ()
#4  0x00007ffff501e915 in __cxa_throw ()
#5  0x00007ffff5001d37 in test_cxx_exception_function () at Enclave/Enclave.cpp:159
#6  0x00007ffff5001d56 in test_cxx_exception_ecall () at Enclave/Enclave.cpp:171
#7  0x00007ffff5001829 in sgx_test_cxx_exception_ecall (pms=0x7fffffffde10) at
Enclave/Enclave_t.c:88
#8  0x00007ffff5003700 in enter_enclave ()
#9  0x00007ffff5004b16 in enclave_entry ()
#10 0x00007ffff7bbb43b in __morestack () from /usr/lib/libsgx_urts.so
#11 0x00007ffff7bbe227 in CEnclave::ecall(int, void const*, void*) () from
/usr/lib/libsgx_urts.so
#12 0x00007ffff7bbfb22 in sgx_ecall () from /usr/lib/libsgx_urts.so
#13 0x0000000000400d71 in test_cxx_exception_ecall (eid=2, retval=0x7fffffffde44)
at App/Enclave_u.c:89
#14 0x00000000004015dd in main (argc=1, argv=0x7fffffffe158) at App/App.cpp:630
(gdb)
```

Figure 10. 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 11. The test case for generating this message is divide-by-zero.

```
Program received signal SIGFPE, Arithmetic exception.
0x00007ffff5401d7f in integer_division (a=0, b=0) at Enclave/Enclave.cpp:173
173         return a / b;
   0x00007ffff5401d7b <integer_division(int, int)+10>:  8b 45 fc        mov
eax,DWORD PTR [rbp-0x4]
   0x00007ffff5401d7e <integer_division(int, int)+13>:  99      cdq
=> 0x00007ffff5401d7f <integer_division(int, int)+14>:  f7 7d f8        idiv
DWORD PTR [rbp-0x8]
(gdb) bt
#0  0x00007ffff5401d7f in integer_division (a=0, b=0) at Enclave/Enclave.cpp:173
#1  0x00007ffff5401db4 in test_illegal_instructions_ecall (op_type=1) at
Enclave/Enclave.cpp:186
#2  0x00007ffff54017a2 in sgx_test_illegal_instructions_ecall (pms=0x7fffffffde10)
at Enclave/Enclave_t.c:101
#3  0x00007ffff5403740 in enter_enclave ()
#4  0x00007ffff5404c96 in enclave_entry ()
#5  0x00007ffff7bbb43b in __morestack () from /usr/lib/libsgx_urts.so
#6  0x00007ffff7bbe227 in CEnclave::ecall(int, void const*, void*) () from
/usr/lib/libsgx_urts.so
#7  0x00007ffff7bbfb22 in sgx_ecall () from /usr/lib/libsgx_urts.so
#8  0x0000000000400f0a in test_illegal_instructions_ecall (eid=2,
retval=0x7fffffffde44, op_type=1) at App/Enclave_u.c:99
#9  0x00000000004016fd in main (argc=1, argv=0x7fffffffe158) at App/App.cpp:575
(gdb)
```

Figure 11. 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 GDB, the Intel **sgx-gdb** "plugin", and the Intel SGX Debug and Pre-Release mode capabilities allow you to debug enclaves for Intel SGX applications for Linux.

Make sure you install the Intel SGX Linux driver, Intel SGX Platform SW, and Intel SGX SDK and can run the sample applications in the SDK.

Build/execute your enclave in Debug configuration (or Simulation configuration) and step through the code with the **sgx-gdb**, using breakpoints and other GDB commands to identify and correct errors. The Debug configuration (or Simulation configuration) 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 Linux OS — 2017 Intel Corporation. https://software.intel.com/sgx-sdk/documentation.
2. Intel Software Guard Extensions SDK Developer Reference for Linux OS — 2017 Intel Corporation. https://software.intel.com/sgx-sdk/documentation.
3. Intel Software Guard Extensions PSW Release Notes for Linux OS — 2017 Intel Corporation. https://software.intel.com/sgx-sdk/download.
4. Intel SGX Support Forums: https://software.intel.com/forums/intel-software-guard-extensions-intel-sgx — Intel.