Getting Started with OpenEnclave

Introduction

This document provides a step-by-step tutorial to begin using the OpenEnclave SDK. It explains how to obtain, build, and install the SDK. It also describes how to develop and build a few simple enclave applications.

Licenses

Microsoft plans to release the OpenEnclave SDK under the MIT license, included here in the source distribution.

https://github.com/Microsoft/openenclave/blob/master/LICENSE

OpenEnclave builds on various third-party packages. It modifies and redistributes libunwind and in addition downloads other third-party packages on-the-fly during the build process. Licensing details for all third-party packages shown in the table below.

Package	License
dlmalloc	https://github.com/Microsoft/openenclave/blob/master/3rdparty/dlmalloc/LICENSE
libcxx	https://github.com/Microsoft/openenclave/blob/master/3rdparty/libcxx/LICENSE
libcxxrt	https://github.com/Microsoft/openenclave/blob/master/3rdparty/libcxxrt/LICENSE
libunwind	https://github.com/Microsoft/openenclave/blob/master/3rdparty/libunwind/LICENSE
mbedtls	https://github.com/Microsoft/openenclave/blob/master/3rdparty/mbedtls/mbedtls/LICENSE
musl libc	https://github.com/Microsoft/openenclave/blob/master/3rdparty/musl/COPYRIGHT

Obtaining the source distribution

OpenEnclave is available from GitHub. Use the following command to download the source distribution.

```
# git clone https://github.com/Microsoft/openenclave
```

This creates a source tree under the directory called openenclave.

Quick Start

Chapters 5 through 7 discuss prerequisites, building, and installing in some detail. This chapter explains how to perform these steps quickly when one wishes to install OpenEnclave into the default location (/opt/openenclave). If this suffices, then perform the steps below, skip those chapters and proceed to chapter 8.

Prerequisites

Execute the following commands from the root of the source tree to install the prerequisites (required packages, the SGX driver, and the SGX AESM service).

```
$ sudo ./scripts/install-prereqs
$ sudo make -C prereqs
$ sudo make -C prereqs install
```

The second and third commands are only necessary if you wish to install the Intel(R) SGX driver and the Intel(R) AESM service. OpenEnclave can be used in simulation mode without these components.

Building

Build is generally out-of-tree (in-tree is possible, though not recommended). To build, pick a directory to build under ("*build/*" below). Then use cmake to configure the build and generate the out-of-tree make files and build.

```
$ mkdir build/
$ cd build/
build$ cmake ..
build$ make
```

Installing

As of now, there is no real need to install the SDK system-wide, so you might use a tree in your home directory:

```
build$ cmake -DCMAKE_INSTALL_PREFIX:PATH=$home/openenclave ..
build$ make install
```

Prerequisites

The following are prerequisites for building and running OpenEnclave.

- Intel® X86-64bit architecture with SGX1 or SGX2
- Ubuntu Desktop-16.04-LTS 64bits
- Various packages: build-essential, ocaml, automake, autoconf, libtool, wget, python, libssl-dev, libcurl4-openssl-dev, protobuf-compiler, libprotobuf-dev, build-essential, python, libssl-dev, libcurl4-openssl-dev, libprotobuf-dev, uuid-dev, libxml2-dev, cmake, pkg-config
- Intel® SGX Driver (/dev/isgx)
- Intel® SGX AESM Service (from the Intel® SGX SDK)

Once Linux and the various packages are installed, it is necessary to install the SGX driver and the SGX AESM service. These can be obtained from the following GitHub repositories.

- https://github.com/01org/linux-sgx-driver
- https://github.com/01org/linux-sgx

Both contain detailed instructions about building and installing these pieces. As a convenience, OpenEnclave provides a script for downloading, building and installing both the driver and the AESM service. From the root of the OpenEnclave source tree, type the following command:

```
$ sudo make -C prereqs
$ sudo make -C prereqs install
```

After this completes, verify that the AESM service is running as follows.

```
$ service aesmd status
```

Look for the string “active (running)”, usually highlighted in green.

Building

Build is generally out-of-tree (in-tree is possible, though not recommended). To build, pick a directory to build under ("*build/*" below).

```
$ mkdir build/
$ cd build/
```

Configure with

```
build$ cmake ..
```

In addition to the standard CMake variables, the following CMake variables control the behavior of the Linux make generator for OpenEnclave:

Variable	Description
CMAKE_BUILD_TYPE	Build configuration (*Debug*, *Release*, *RelWithDebInfo*). Default is *Debug*.
ENABLE_LIBC_TESTS	Enable Libc tests. Default is enabled, disable with setting to "Off", "No", "0", ...
ENABLE_LIBCXX_TESTS	Enable Libc++ tests. Default is disabled, enable with setting to "On", "1", ...
ENABLE_REFMAN	Enable building of reference manual. Requires Doxygen to be installed. Default is enabled, disable with setting to "Off", "No", "0", ...

E.g., to generate an optimized release-build with debug info, use

```
build$ cmake .. -DCMAKE_BUILD_TYPE=relwithdebinfo
```

Multiple variables can be defined at the call with multiple "-D*Var*=*Value*" arguments.

Once configured, build with

```
build$ make
```

This builds the entire OpenEnclave SDK, creating the following files.

Filename	Description
output/bin/oegen	Utility for generating ECALL and OCALL stubs from IDL
output/bin/oesign	Utility for signing enclaves
output/lib/enclave/liboeenclave.a	Core library for building enclave applications
output/lib/enclave/liboelibc.a	C runtime library for enclave
output/lib/enclave/liboelibcxx.a	C++ runtime library for enclave
output/lib/host/liboehost.a	Library for building host applications
output/share/doc/openenclave/	HTML API reference for OpenEnclave

If things break, set the **VERBOSE** make variable to print all invoked commands.

```
build$ make VERBOSE=1
```

Building from within a subtree of the build-tree builds all dependencies for that directory as well. "**make clean**" is handy before a spot-rebuild in verbose mode.

A successful build only outputs the HTML API reference into the build-tree. To update the refman *.md-files in the source-tree, use

```
build$ make refman-source
```

Testing

After everything has been built, execute the tests via **ctest** (see "**man ctest**" for details).

```
build$ ctest
```

To run the tests in simulation mode, use

```
build$ OE_SIMULATION=1 ctest
```

If things fail, "**ctest -V**" provides test details. Executing ctest from a sub-dir executes the tests underneath.

libcxx tests are omitted by default due to their huge cost on building (30mins+). Enable by setting the cmake variable **ENABLE_LIBCXX_TESTS** before building.

```
build$ cmake -DENABLE_LIBCXX_TESTS=ON ..
build$ make
```

If you are in a hurry and just need a quick confirmation, disable the libc tests with the **ENABLE_LIBC_TESTS** cmake variable like so:

```
build$ cmake -DENABLE_LIBC_TESTS=OFF ..
[...]
```

To run valgrind-tests, add "**-D ExperimentalMemCheck**" to the ctest call. Enclave tests all seem to fail today, though this succeeds:

```
build$ ctest -D ExperimentalMemCheck -R oeelf
```

Installing

Specify the install-prefix to the cmake call. As of now, there is no real need to install the SDK system-wide, so you might use a tree in your home directory:

```
build$ cmake -DCMAKE_INSTALL_PREFIX:PATH=~/openenclave ..
build$ make install
```

If you want the SDK tools to be available to all users and headers/libs available from a system default location, you may opt to install system-wide. This naturally requires root privileges. Note that there is no uninstall script (we target an rpm/deb-based SDK install in the future), hence we recommend overwriting the default (/usr/local/) with a singular tree.

```
build$ cmake -DCMAKE_INSTALL_PREFIX:PATH=/opt/openenclave ..
build$ sudo make install
```

On Linux, there is also the **DESTDIR** mechanism, prepending the install prefix with the given path:

```
build$ make install DESTDIR=foo
```

The following table shows where key components are installed.

Path	Description
<install_prefix>/bin	Programs
<install_prefix>/include/openenclave	Includes
<install_prefix>/lib/openenclave/enclave	Enclave libraries
<install_prefix>/lib/openenclave/host	Host libraries

<install_prefix>/share/doc/openenclave	Documentation
<install_prefix>/share/openenclave	Samples and make-include

You may use the make-include in **<install_prefix>/share/openenclave/config.mak** in your own project for sourcing variables containing version info and SDK install paths.

Samples

Find the samples under **share/openenclave/samples/** of the installation.

Change to the new samples directory and build and run the samples.

```
$ cd $home/share/openenclave/samples
$ export OPENENCLAVE_CONFIG=$home/share/openenclave/config.mak
$ make
$ make run
[...]
```

If these samples run without an error, then OpenEnclave is installed and working correctly.

Developing a simple enclave (echo)

This chapter shows how to develop a simple enclave called echo. The next chapter explains how to use this enclave in a host application. This example is included in the installed samples directory (see /opt/openenclave/share/openenclave/samples/hello).

The ECALL

The echo enclave implements a single ECALL named EnclaveEcho(), which is called by the host (in the next chapter). This function has the following signature.

```
OE_ECALL void EnclaveEcho(void* args);
```

The args parameter can be whatever the host and the enclave agree on. In this case args is a pointer to a zero-terminated string. The OE_ECALL macro exports the function and injects it into a special section (.ecall) in the ELF image. When the host loads the enclave, it builds a table of all ECALLs exported by the enclave.

The Echo enclave Listing

Here's the full listing for the echo enclave (enc.c):

```
#include <openenclave/enclave.h>

OE_ECALL void EnclaveEcho(void* args)
{
    OE_CallHost("HostEcho", args);
}
```

Notice EnclaveEcho() performs an OCALL, calling the host's HostEcho() function with the same arguments.

Compiling enc.c

This sample includes a makefile for building this enclave, but to be more instructive, this chapter shows how to build components from scratch. First, we define the INCLUDES make variable as follows.

```
INCLUDES=-I/opt/openenclave/include/openenclave/enclave
```

This is the directory that contains the **openenclave.h** header, as well as C headers files.

Next, we define the CFLAGS make variable as follows.

```
CFLAGS=-O2 -nostdinc -fPIC
```

Finally, we compile the source file.

```
gcc -c $(CFLAGS) $(INCLUDES) enc.c
```

This produces enc.o.

Linking the enclave

Next, we link the enclave to produce echoenc.so. First, we define the LDFLAGS make variable.

```
LDFLAGS=\
    -nostdlib \
    -nodefaultlibs \
    -nostartfiles \
    -Wl,--no-undefined \
    -Wl,-Bstatic \
    -Wl,-Bsymbolic \
    -Wl,--export-dynamic \
    -Wl,-pie \
    -Wl,-eOE_Main
```

The -eOE_Main option requires some explanation (see the ld man page about other options). This option specifies the name of the entry point for the enclave. The linker stores the virtual address of the OE_Main() function in the ELF header (Elf64_Ehdr.e_entry) of the resulting binary. When the enclave is instantiated by the host, this entry point is copied to each TCS (Thread Control Structure) in the image. When the host invokes the SGX EENTER instruction on a TCS, the hardware fetches the entry point from the TCS and jumps to that address and the OE_Main() function begins to execute.

Next, we define the LIBRARIES make variable.

```
LIBRARIES=-L/opt/openenclave/lib/openenclave/enclave -loeenclave
```

The LIBRARIES make variable specifies the enclave library, which contains the enclave intrinsics, including the OE_Main() entry point. Note that this sample uses neither a C or C++ runtime library. Other samples will show how these are used.

Finally, we link the enclave.

```
gcc $(LDFLAGS) $(LIBRARIES) enc.o -o echoenc.so
```

Signing the enclave

The final step in creating an enclave is to sign it with the oesign tool. This tool takes the following parameters.

```
# oesign

Usage: oesign ENCLAVE CONFFILE KEYFILE
```

The CONFFILE argument is the name of a configuration file that defines enclave settings, such as stack size, heap size, and the maximum number of threads (TCSs). Here is a sample:

```
# echo.conf
Debug=1
NumHeapPages=1024
NumStackPages=1024
NumTCS=16
```

The KEYFILE argument is a private RSA key used to sign the enclave.

A self-signed private key can be generated using OpenSSL as follows.

```
# openssl genrsa -out private.pem -3 3072
```

Then the public key can be generated from this key as follows.

```
# openssl rsa -in private.pem -pubout -out public.pem
```

Finally, we sign the enclave as follows.

```
# oesign echoenc.so echo.conf private.pem
```

Created echoenc.signed.so

Developing a simple host (echohost)

Next, we develop a host to run the echoenc.signed.so enclave that we developed in the previous chapter. The listing follows.

```
#include <openenclave/host.h>
#include <stdio.h>

OE_OCALL void HostEcho(void* args)
{
    if (args)
    {
        const char* str = (const char*)args;
        printf("%s\n", str);
    }
}

int main(int argc, const char* argv[])
{
    OE_Result result;
    OE_Enclave* enclave = NULL;

    if (argc != 2)
    {
        fprintf(stderr, "Usage: %s ENCLAVE_PATH\n", argv[0]);
        return 1;
    }

    result = OE_CreateEnclave(argv[1], OE_FLAG_DEBUG, &enclave);
    if (result != OE_OK)
    {
        fprintf(stderr, "%s: OE_CreateEnclave(): %u\n", argv[0], result);
        return 1;
    }

    result = OE_CallEnclave(enclave, "EnclaveEcho", "Hello");
    if (result != OE_OK)
    {
        fprintf(stderr, "%s: OE_CallEnclave(): %u\n", argv[0], result);
        return 1;
    }

    OE_TerminateEnclave(enclave);

    return 0;
}
```

This host performs the following tasks:

- Defines an OCALL: HostEcho()
- Instantiates an enclave: OE_CreateEnclave()

- Calls into the enclave: OE_CallEnclave()
- Terminates the enclave: OE_TerminateEnclave()

After building the host application, we are ready to run the host.

```
# host/echohost ./enc/echoenc.signed.so

Hello
```