Dynamic Loading to Build Intel® SGX Applications in Linux

## Scope

## This article describes dynamic loading of shared library objects to build Intel® Software Guard Extensions (Intel® SGX applications in Linux. The applications need to make use of dynamic loading to avoid runtime load errors.

# Intel SGX application basics

Intel SGX application development requires dividing the application into two logical components:

* Trusted component — The code that accesses the secret resides here. This component is also called an enclave. More than one enclave can exist in an application.
* Untrusted component — The rest of the application including all its modules.

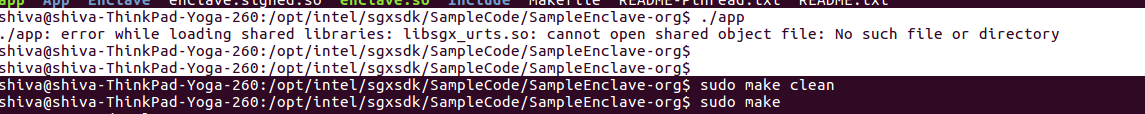
An enclave is a protected area in the application’s address space, which provides confidentiality and integrity even in the presence of privileged malware. Attempted accesses to the enclave memory area from a software not resident in the enclave are prevented even from privileged software’s such as virtual machine monitors, BIOS, or operating systems. It provides a protected place for code and data in application. Intel provides some special hardware instruction to create and support enclave. Intel SGX enclave memory is protected even from privileged software. For more information on Intel SGX technology, refer to link provided in 1.

## Dynamic Loading requirement

Most Linux applications use dynamic linking, but this presents a problem for Intel SGX: if the required Intel SGX Platform Software (PSW) libraries are not present or not installed, the application will encounter runtime load errors that prevent it from executing. With the dynamic loading technique, symbols are loaded and unloaded at run time, allowing applications to take actions based on whether or not required libraries are present rather than be completely blocked from executing at all.

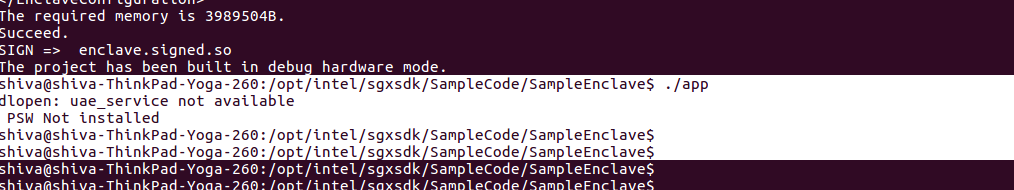
# Dynamic Linking vs Dynamic loading when PSW not present

Figure 1 shows error messages when a dynamically-linked Intel SGX Application is run. In this case, application is unable to link the required libraries.



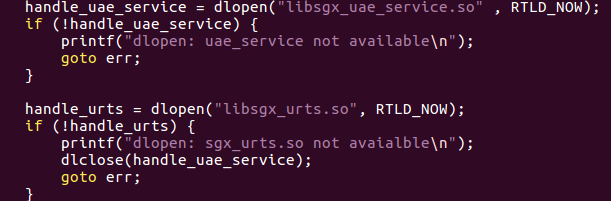
***Figure 1: Dynamic Linking error screen shot***

Figure 2 shows the messages when dynamically loaded SGX Applications is run. In this case, application exits gracefully with appropriate message.



***Figure 2: Dynamic Loading error screen shot***

In this scenario, we use the dl\* family of library calls to dynamically load the required symbols at run time. Here we open the shared object using dlopen and use dlsym for resolving the required symbol. If the PSW is not present, we just exit the application gracefully. Otherwise execution continues in the Intel SGX enabled path.



# *Figure 3: Dlopen usage screen shot*

# Dynamic Loading Vs Dynamic Linking

***Dynamic Loading*:** Dynamic loading refers to mapping an executable or library into a process's memory after is has started. Dynamic loading occurs when you call dlopen and dlsym, or their equivalent on other operating systems. The object file is loaded dynamically, and under the program’s control.

libdl.so

Mylib.so

|  |
| --- |
| Myfun1() |
| Myfun2() |
| Myfun3() |

dlopen

dlsym

dlclose

Int main()

{

Void \*handle;

dlopen(“mylib.so”, RTLD\_LAZY);

void (\*f)() = dlsym(handle, “Myfun1);

f();

dlclose();

}

dlerror

Ld-linux.so

***Figure 4: Dynamic Loading***

***Dynamic Linking:*** Dynamic linking refers to resolving symbols — associating their names with addresses or offsets — after compile time. During dynamic linking, symbols both in the calling program and in the library are resolved based on the process's possibly-unique memory layout at that time.

Libc.so

App(exe)

Printf()

Rand()

Libsgx\_urts.so

App.o

Libc.so

Libsgx\_urts.so

Sgx\_create\_enclave()

Sgx\_destroy\_enclave()

***Figure 5: Dynamic Linking***

# Technique’s for Dynamic Loading

In C, there is an API for opening a library, looking up symbols, handling errors, and closing the library. C users will need to include the header file <dlfcn.h> to use this API.

There are API’s which facilitates the dynamic loading functionality. A few of the relevant one’s are as follows:

**Dlopen:**

The dlopen() function opens a library and prepares it for use. In C its prototype is:

void \* dlopen(const char \*filename, int flag);

In dlopen(), the value of *flag* must be either RTLD\_LAZY, which means “resolve undefined symbols as code from the dynamic library is executed'', or RTLD\_NOW, which means “resolve all undefined symbols before dlopen() returns and fail if this cannot be done''. If the libraries depend on each other (e.g., X depends on Y), then you need to load the dependencies first (in this example, load Y first, and then X).

**Dlsym:**

The main routine for using a DL library is dlsym(3), which looks up the value of a symbol in a given (opened) library. This function is defined as:

void \* dlsym(void \*handle, char \*symbol);

The handle is the value returned from dlopen, and symbol is a NIL-terminated string

**dlclose:**

The converse of dlopen() is dlclose(), which closes a DL library. The dl library maintains link counts for dynamic file handles, so a dynamic library is not actually deallocated until dlclose has been called on it as many times as dlopen has succeeded on it.

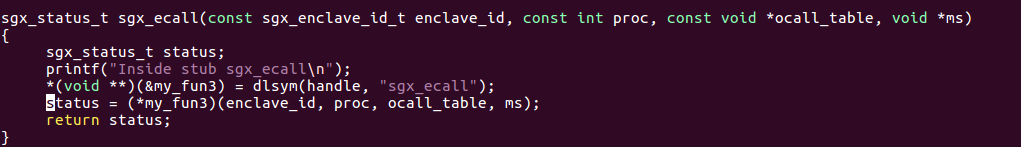
# Stub functions and dynamic loading problem

While using dlibrary calls for implementing dynamic loading, we ran in to issue of ‘undefined references’ error with below signature:

*/opt/intel/sgxsdk/SampleCode/SampleEnclave/App/Enclave\_u.c:36: undefined reference to `sgx\_ecall'  
collect2: error: ld returned 1 exit status  
Makefile:210: recipe for target 'app' failed  
make: \*\*\* [app] Error 1*

sgx\_ecall is part of generated untrusted code and while compiling and linking, sgx\_ecall symbol was not resolved, hence undefined reference error is thrown. To overcome this kind of undefined reference errors, we are using stub functions that act as placeholders for the actual functions.

In the above case, we define stub functions for the core SGX functions which are referenced by the application. Sgx\_stub.c contains the stub definitions wherein we dynamically resolve the requested symbol using dlsym dlibrary call.

  
***Figure 6: Illustrating an SGX stub call***

# Summary

We have covered the technique of dynamic loading of shared objects to build SGX Applications in Linux. SGX Applications require untrusted libraries at runtime, but sometimes the Intel SGX PSW may not be installed on the system. Dynamic loading helps this situation by resolving symbols at runtime. The applications that require SGX can then exit gracefully if required libraries are not present in the system.

# References

1. <https://software.intel.com/en-us/articles/intel-software-guard-extensions-tutorial-part-1-foundation> - Intel Corporation.
2. <http://tldp.org/HOWTO/Program-Library-HOWTO/dl-libraries.html>

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL® ASSUMES NO LIABILITY WHATSOEVER AND INTEL® DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL® PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL® AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL® OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL® PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

No computer system can provide absolute security under all conditions. Built-in security features available on select Intel® processors may require additional software, hardware, services and/or an Internet connection. Results may vary depending upon configuration. Consult your system manufacturer for more details.

Intel®, the Intel® Logo, Intel® Inside, Intel® Core™, Intel® Atom™, and Intel® Xeon® are trademarks of Intel Corporation in the U.S. and/or other countries. Other names and brands may be claimed as the property of others.

\* Other names and brands may be claimed as properties of others.

Copyright © 2017 Intel® Corporation