INDIAN INSTITUTE OF SCIENCE

भारतीय विज्ञान संस्थान

Group Members

RaJ Jha(17980) & Paras Lohani(18226)

E0:256

TPCSS PROJECT REPORT (RNG:ENTROPY)

23 Jan 2021

Guided By - Prof. Vinod Ganapathy

Department of Computer Science and Automation

Contents

1 Introduction

Good entropy is the fundamental basis for good cryptography and SSL or TLS. We should ensure that our entropy should not be weak or predictable because if it was so then a strong adversary can break our security.
All security protocols depend on good entropy:

- Public/Private key pairs/identities.
- Unique AES keys
- Unique initialization vectors

Therefore bad entropy also affects the security of:

- Session keys (SSL/TLS, SSH)
- Encrypted files

The entropy module allows CTR_DRBG to reseed using gathered entropy. We gathered an entropy :

- When an application calls the entropy gather function.
- During a call from CTR_DRBG

Gathered entropy is accumulated using an SHA-512 update function. Once a reseed is triggered, the SHA-512 function is finalized. The results of this first SHA-512 function are then hashed again and returned.

2 Module Covered

2.1 Description

In this project we worked on `ENTROPY` which is a sub module of `RNG` component of TLS library. We implemented the function gen_entropy.c and entropy.h
The previous implementation of mbedTLS in C/C++ have various security issued like memory leak, unsafe block etc. which can be easily exploited by adversary. So we have implemented this module using a memory safe language `RUST`.

2.2 Link to GitHub Repository:

The link to the code is given here

3 Implementation

The various functions which we have written in `RUST` and its implementation details are depicted below:

3.1 entropy_add_source:

MbedTLS uses a number of different sources for its entropy. Entropy Sources are non-blocking. A counter is maintained for every source, which counts the number of bits that were returned on each gather call. If we run mbedTLS on a different platform, such as an embedded platform, we have to add platform-specific or application-specific entropy sources.

It requires us to provide a callback **f_source** that we can call whenever the entropy pool tries to gather entropy, the data **p_source** that we need with our callback, and a threshold. This threshold indicates the minimum number of bytes the entropy pool should wait on from this callback before releasing entropy. If we chose a value that is higher than our callback then it will block entropy collection. Here we have also indicated that if the added source is strong or weak. The entropy module refuses to deliver entropy unless it has at least one strong source.

When collecting entropy for a request, the entropy pool does a maximum of 256 polls to each entropy source to retrieve entropy from them. If the threshold value for a source is higher than the entropy it can deliver in those 256 polls, then it will throw an error.
There might be a case that an entropy source provides some limited entropy, but not on every poll, then we selected a threshold value of 0. A zero-threshold does not cause the entropy pool to return an error if it cannot provide any entropy in 256 calls.

3.2 mbedtls_entropy_gather:

It takes as an input a structure mbedtls_context_struct which contains 4 data fields : Accumulator started, State_count and two pointers which points to two other structure mbedtls_entropy_source_state and mbedtls_sha512_context. It triggers an extra gather poll for the accumulator and returns 0 on success or returns an error if we are failed to gather an entropy.
Let's see how it works:
The entropy collector gathers entropy from multiple sources and integrate them to its internal entropy state. Each time entropy is collected, the internal state gets better. Unless additional entropy sources have been added to the default entropy collector sources, mostly OS dependent random sources are used. Each time the entropy collector gathers entropy from the system, the entropy gets extracted.

3.3 mbedtls_write_seed_file:

It takes as an input a structure mbedtls_context_struct which contains 4 data fields : (Accumulator started, State_count and two pointers which points to two other structure mbedtls_entropy_source_state and mbedtls_sha512_context) and name of the file.

This function returns 0 on successful write to seed file. If a hardware platform does not have a hardware entropy source to leverage into the entropy pool,then this might cause security error in it because a strong entropy source is crucial for security of cryptographic and TLS operations.

we can use the NV seed entropy source that mbedTLS provides for platforms that support non-volatile memory,mbedTLS use a fixed amount of entropy as a seed and update this seed each time entropy is gathered with an mbedTLS entropy collector for the first time. In a simple case it means that the seed is updated after reset at the start of the first TLS connection.

We have to define `mbedtls_entropy_NV_seed`. This ensures the entropy pool knows it can use the NV seed entropy source. We should set the following functions to make the entropy collector call them:

```
int (*mbedtls_nv_seed_write)( unsigned char *buf, size_t buf_len )
int (*mbedtls_nv_seed_read)( unsigned char *buf, size_t buf_len );
```

3.4 mbedtls_entropy_update_seed_file:

It takes as an input a structure `mbedtls_context_struct` which contains 4 data fields : (Accumulator started, State_count and two pointers which points to two other structure mbedtls_entropy_source_state and mbedtls_sha512_context) and name of the file.

The advantage of a seed file is that we can generate it on a high-entropy system and then update and use it on our low-entropy system.

It reads and updates a seed file. We update the seed for 2 reasons.First, `ENTROPY_MAX_SEED_SIZE` may have exceeded reseed_interval. This is rare, since reseed_interval is set to high values in practice; its maximum value is 248, meaning it would naturally update seed once every couple of million years.

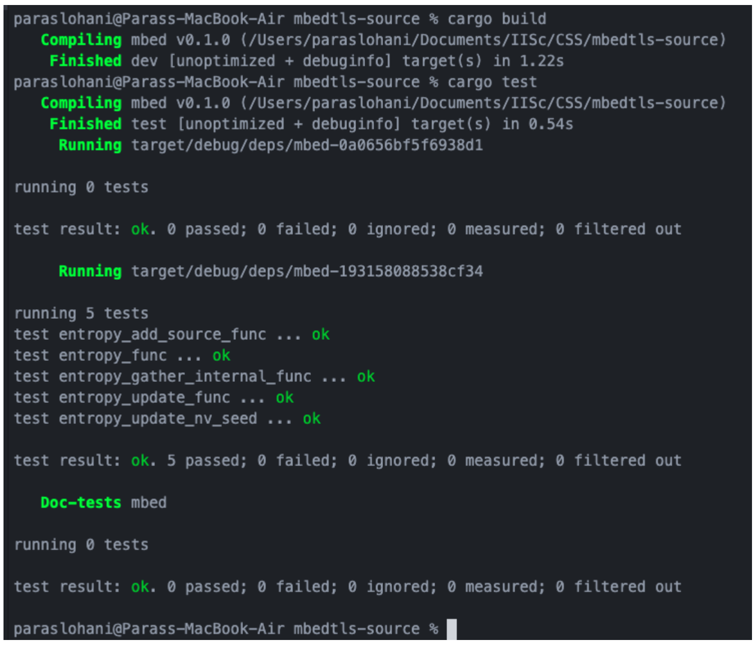
Indeed, in practice, one does not want a Random Bit Generator to update seed often. This would give an attacker more opportunities to compromise the entropy source. More commonly, Update seed is called when a Random Bit Generator state could have been compromised, and requires fresh entropy to be mixed in.

Reseed(secret key,v, entropy) :=
The v component holds the newest "block" of pseudorandom bits that the Random Bit Generator has generated
(secret key1,v1) ← Update (seed, secret key,v)
reseed_counter ← 1
return (secret key,v,reseed_counter)

3.5 mbedtls_entropy_update_manually:

It adds the data to the accumulator manually.It takes as an input a structure `context\_struct` which contains 4 data fields : (Accumulator started, State_count and two pointers which points to two other structure mbedtls_entropy_source_state and mbedtls_sha512_context) and data and length of data. It returns an error if the data provided to accumulator is corrupted.

4 Tests



```
paraslohani@Parass-MacBook-Air mbedtls-source % cargo build
   Compiling mbed v0.1.0 (/Users/paraslohani/Documents/IISc/CSS/mbedtls-source)
    Finished dev [unoptimized + debuginfo] target(s) in 1.22s
paraslohani@Parass-MacBook-Air mbedtls-source % cargo test
   Compiling mbed v0.1.0 (/Users/paraslohani/Documents/IISc/CSS/mbedtls-source)
    Finished test [unoptimized + debuginfo] target(s) in 0.54s
     Running target/debug/deps/mbed-0a0656bf5f6938d1

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out

     Running target/debug/deps/mbed-193158088538cf34

running 5 tests
test entropy_add_source_func ... ok
test entropy_func ... ok
test entropy_gather_internal_func ... ok
test entropy_update_func ... ok
test entropy_update_nv_seed ... ok

test result: ok. 5 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out

   Doc-tests mbed

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out

paraslohani@Parass-MacBook-Air mbedtls-source %
```

Figure 1: Unit Tests Result

5 Challenges

5.1 Complicated Macros

Macro rules in rust is too confusing. A simple macro in C++ needs a complicated implementation in Rust. Therefore in most of the cases we used either normal declaration of variables or implemented a function.

5.2 Strict Compilation

In Rust, when one declare some variable, it should be instantiated there itself with some value otherwise compiler will throw error saying the variable is not initialized. But this is not the case with C. Therefore whenever any variable is declared, we have initialised it with **Default::default()** which will give a default value to the variable according to its data type. The same is done for the struct declared in headers. For that a separate **implementation (impl)** is written or a macro #[**derive(Default)**] is added at the top.

One cannot use integer for boolean check. You have to use bool data type for that. C is flexible enough to allow you that.

5.3 Documentation

Documentation is not good as compared to languages like python, Java, C++. One has to waste too much of time in internet surfing for searching libraries which will be useful for a particular code. To solve this issue we tried to write code sometimes without using libraries i.e. we expanded the function logic and implemented it in hard way. It was very difficult to implement file part of code in Rust. Details in documentation lacks in explaining the functionality of std::fs.

5.4 Unsafe block of code

In rust some functions does not compile without being covered in a unsafe block. Function like memset and reference pointer value change. For some part we have changed the logic such that we do not require the block anymore. But some places still has unsafe block which is difficult to be removed.

5.5 Changed a constant reference variable in C

In C, one can change the value of constant reference value whereas it is not possible in rust. One constant cannot borrow the values of another constant in Rust. In rust one has to use mutable pointer for that. *Refer entropy_update function in entropy.c file for such case.*

5.6 Void data type

Rust do not have void type. For places in code where void is needed we have used std::ffi::c_void which worked the same way.

5.7 Size of mutable variable

One cannot use the function size_of in std::mem for finding size of mutable variable. The variable should be constant. We have hard-coded the size mentioned while declaring the variable.

6 Features of Rust used

6.1 Memory Safety

Rust achieves memory safety using the principles of ownership and borrowing.Here memory space is owned by the variable and temporarily borrowed by other variables which allows Rust to provide memory safety at compile time without depending upon garbage collector. In RUST we can create a pointer to any location, but we cannot dereference it. Any reference we create always is bound to an object. If a mutable `(&mut)` reference to an object exists, no other reference to this object can exist.

6.2 Type inference

Type inference is the process of automatically detection of the type of an expression and RUST provides this feature.This makes thing easier as when we initialized a variable with proper value then the type of the variable is automatically assigned by the RUST compiler.

6.3 Borrow Checker

Rust's ownership model feels like something in between. By keeping track of where data is used throughout the program and by following a set of rules, the borrow checker is able to determine where data needs to be initialized and where it needs to be freed (or dropped, in Rust terms)

6.4 Default implementation

One can implement default behavior of the struct or traits. In our code we have implemented it therefore we can use **Default::default()** to initialize variables of struct in one go.

6.5 Unsafe

Rust compiler tells us the part of code which is unsafe. Therefore if we desperately need to run that part we can add it inside unsafe block which will tell the compiler that the part of code is needed to run.

6.6 Data type

Rust provides differnt data types of bit of numbers. For example 8, 16, 32 etc. It is easy to differentiate which data type to use when. Its a feature which we have used in our code.

6.7 Type Casting

Type casting in rust is very simple to use. **as** keyword is used for that. For example if you want to use usize as i32, then simply type **as i32** after variable name.

7 Conclusion

We have successfully implemented the **ENTROPY** submodule using RUST programming language by exploiting the various features of RUST which in turn removes various security issues and vulnerabilities in C implementation of mbedTLS. We ensured that ENTROPY submodule are secure in their basic functions against a nonadaptive adversary.

8 References

[1] "`Random Number Generator (RNG) Module Level Design`" https://tls.mbed.org/module-level-design-rng
[2]"`RUST language documentation`" https://rust-lang.github.io/unsafe-code-guidelines/layout/function-pointers.html
[3] "`Random Number Generation Using Deterministic Random Bit Generators`" https://nvlpubs.nist.gov/
90Ar1.pdf
[4]"`Guide to port C to RUST`" https://locka99.gitbooks.io/a-guide-to-porting-c-to-rust/content/setting$_u p_r u$
[5]"`mbedTLS wikipedia`" https://en.wikipedia.org/wiki/Mbed$_T LS$