

IT Security Report

Source Code Audit of QAN's Fence



*Author: Tamás Dávid Pethes /six/
Date: 2020.08.05.*

Prepared for:



Confidential

Contents

Executive summary.....	3
Risk classification.....	4
Audit results.....	5
High - Denial of service by calling functions with invalid values.....	5
High - Function running for indefinite time, possible denial of service.....	6
Important - GLP secrets might not be zeroized.....	7
Important - Lack of documentation.....	8
Recommendation - Code format improvements.....	9
Recommendation - Miscellaneous recommendation.....	10
Worklog.....	11
List of tasks during the audit.....	11
Auditor contact: Awalcon.....	12

Executive summary

The source code audit and fuzzing of QAN's Fence project reveals multiple parts of the code where security improvement should be applied. The audit started on 2020. July 27., lasted until 2020. August 5. and included 5 working days.

Considering that Fence is a library and it will be used by not just QAN, but also external clients, the threat model includes cases where the external developers might make mistakes in using the library.

The threat model specifies attackers who either access Fence directly or through a program which uses Fence (eg. a client buys Fence and bulilds software around it). The latter is more likely to happen, but in this case it is important to point out for the client who uses Fence what his responsibilities are regarding security. what they take care of before passing data to Fence and what Fence's security is responsible for. At the time of the source code audit such "README" file was not available, but it is highly recommended to provide in the future.

Two high risk vulnerabilities have been found that can lead to DoS (denial of service). On the other hand, no RCE (Remote Code Execution) vulnerability has been found which is a very positive trait.

One of these is related to memory allocation and the program stops in this case.

The other high risk vulnerability is related to a logic issue where the program runs without time limit, but never actually finishes. This can lead to denial of service on the long run.

Applying the recommendations from this report will fix the vulnerabilities found and generally improve the code's security and quality.

The report is to be published by QAN after the issues have been mitigated.

Risk classification

High risk: function calls or logic errors that can break the intended use of Fence.

Important improvement: improvements that should be done to avoid possible security bugs or misused when calling into Fence.

Recommended: Further recommendations that improve the quality and readability of the code. These are not related only to security, but if the code is cleaner and more understandable, there are less chances for bugs.

Audit results

High - Denial of service by calling functions with invalid values

Description: Values passed to the core functions are not validated and there are three possible outcomes when such input is sent:

1. The program exits with "panic"
2. Fails to allocate memory and exists
3. Runs for an indefinite time (see next vulnerability for details)

Impact: Denial of Service.

Proof of concept:

```
/*const N: usize = 4096*99999999; // memory allocation of 819199991808 bytes failedAborted
const L: u16 = 59393u16;
let samples:Vec<u16> = dck::sample(L, N);
for i in 0..N {
    assert!(samples[i] == 0 || samples[i] == 1 || samples[i] == L-1);
}*/
```

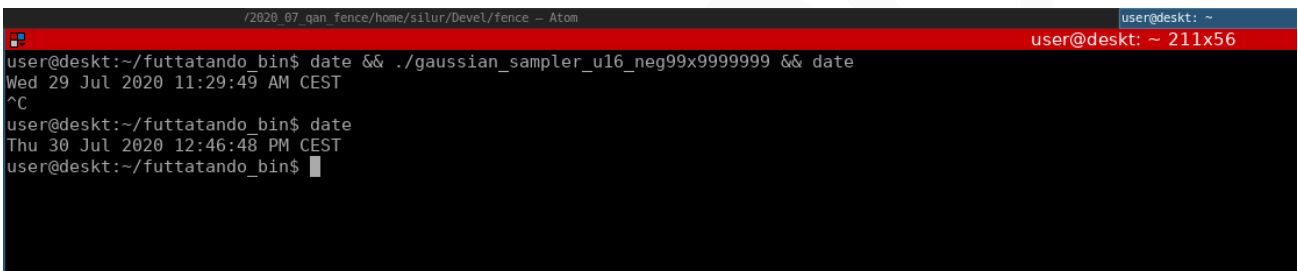
Proposed solution: Implement validation. Make sure the values passed to the functions are positive numbers and are in a valid range.

High - Function running for indefinite time, possible denial of service

Description: Similar to the vulnerability before, but in this case the program does not stop. It runs as long as it is not killed manually.

Impact: Fence gets stuck at that point and in case there is a program written around it and the attacker can make a call that spawns Fence, he/she repeats it as long as the computer runs out of memory, causing a Denial of Service for the whole system.

Screenshot:



```
/2020_07_gan_fence/home/silur/Devel/fence - Atom user@deskt: ~
user@deskt:~/futtatando_bin$ date && ./gaussian_sampler_u16_neg99x9999999 && date
Wed 29 Jul 2020 11:29:49 AM CEST
^C
user@deskt:~/futtatando_bin$ date
Thu 30 Jul 2020 12:46:48 PM CEST
user@deskt:~/futtatando_bin$
```

Proposed solution: Validate input. If the first vulnerability is fixed by validating the input, this will also be fixed.

Important - GLP secrets might not be zeroized

Description: Memory is not zeroized when returning secrets.

Impact: secrets could be leaking in memory.

Proposed solution: Force volatile memory to get zeroized. You can use the "zeroize" crate for that.

Example for a decent solution - <https://crates.io/crates/secretcy>

References:

<https://github.com/dalek-cryptography/curve25519-dalek/issues/11>

Important - Lack of documentation

Description: Fence does not have a README file.

For example, clients do not know the correct way to call functions and might cause panics.

```
// thread 'main' panicked at 'attempt to multiply with overflow',  
/rustc/5c1f21c3b82297671ad3ae1e8c942d2ca92e84f2/src/libcore/ops/arith.rs:316:45  
/*const N: usize = 128;  
const L: u16 = 59393u16;  
let u16max = u16::max_value();  
let u16sampler = GaussianSampler::<u16>::new(&u16max*99,(u16max*99) as f64,false);  
u16sampler.sample(N, false);*/
```

Impact: anyone using the project has no guidance and it is not clarified how a client developer should use the library.

Proposed solution: Provide a README file.

Recommendation - Code format improvements

Description: format improvements for the code. you can use fmt check and clippy.

```
$ cargo fmt -- --check
```

```
$ cargo clippy
```

```
Checking util v0.1.0 (/home/six/dx/Vlk/0v_itS/0_munkak/2020_07_qan_fence/home/silur/Devel/fence/util)
```

```
warning: unneeded `return` statement
```

```
--> util/src/modalg.rs:12:5
```

Proposed solution: Run and apply the recommendations of these tools.

Recommendation - Miscellaneous recommendation

Description: Silur's gmail addresses have been found in the code.

Impact: this is a reference to a non-company address, therefore it does not look official.

Proposed solution: Remove a gmail addresses.

Worklog

List of tasks during the audit

- Research on post-quantum cryptography based on Fence's paper
- Manual review of the provided source code. Methodology partially includes:
 - [OSSTMM](#)
 - [NIST SP800-115](#)
 - https://scrapbox.io/layerx/Rust_code_audit
 - [Bugs You'll Probably Only Have in Rust](#)
- Fuzzing the core functions (based on auditor interpretation and consultation with Silur) using the tool called AFL
- Analyzing the code using:
 - cargo test
 - cargo audit
 - cargo clippy
 - rg
- Analyzing compiled SWIFFTX binary using:
 - Valgrind

Worklog sha512:

03f2e43ab3462298c72111804cccee49a6accfe42211a03cfbda3212c12f3db592dda26d23eb57ff114973113c4a9883cd56a3c9e06a0e7de39f8e5cf754a09f

Auditor contact: Awalcon

Tamás Dávid Pethes /six/

Mobile and Signal: +36 20 256 4090

E-mail: ptdavid@awalcon.org

PGP: B1F7 B1D6 8838 98B4 2212 1D90 CA71 D1E4 078E 99C5

Website: <https://awalcon.org/>