Reef Chain

FRAME Pallet Audit

Prepared by: **Halborn**

Date of Engagement: **July 5th, 2021 - July 23rd, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	07/05/2021	Piotr Cielas
0.2	Document Edits	07/12/2021	Piotr Cielas
1.0	Final Version	07/26/2021	Piotr Cielas
1.1	Remediation Plan	09/01/2021	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Piotr Cielas	Halborn	Piotr.Cielas@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Reef Chain is a smart contracts blockchain that is backwards-compatible with Ethereum EVM (and the Solidity programming language) and uses NPoS/PoC consensus.

Reef engaged Halborn to conduct a security assessment on their FRAME pallets beginning on July 5th, 2021 and ending July 23rd, 2021. This security assessment was scoped to the FRAME pallet source code in Rust.

Though this security audit's outcome is satisfactory, only the most essential aspects were tested and verified to achieve objectives and deliverables set in the scope due to time and resource constraints. It is essential to note the use of the best practices for secure pallet development.

1.2 AUDIT SUMMARY

The team at Halborn was provided three weeks for the engagement and assigned one full time security engineer to audit the security of the assets in scope. The engineer is a blockchain and smart contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit to achieve the following:

- Ensure that all FRAME pallet functions are intended.
- Identify potential security issues with the assets in scope.

In summary, Halborn identified some security risks that were mostly addressed by the Reef team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process,and implementation; automated testing techniques help enhance coverage of the FRAME pallets code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture, purpose and use of the pallet.
- Pallet manual code review and walkthrough.
- Manual Assessment of use and safety for the critical pallet variables and functions in scope to identify any arithmetic related vulnerability classes
- Manual testing with custom Javascript. (polkadot{.js} and reef.js).
- Static Security Analysis of security for the pallet in scope and imported functions. (cargo-clippy)
- Scanning of Rust files for vulnerabilities. (cargo-audit)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident, and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. It’s quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.

2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

This review was scoped to the currencies, evm-accounts and evm-bridge FRAME pallets.

1. currencies pallet

   (a) Repository: currencies

   (b) Commit ID: 393d0c0821cc25ea5c6912d9cac8f61a9232c9a3

2. evm-accounts pallet

   (a) Repository: evm-accounts

   (b) Commit ID: 393d0c0821cc25ea5c6912d9cac8f61a9232c9a3

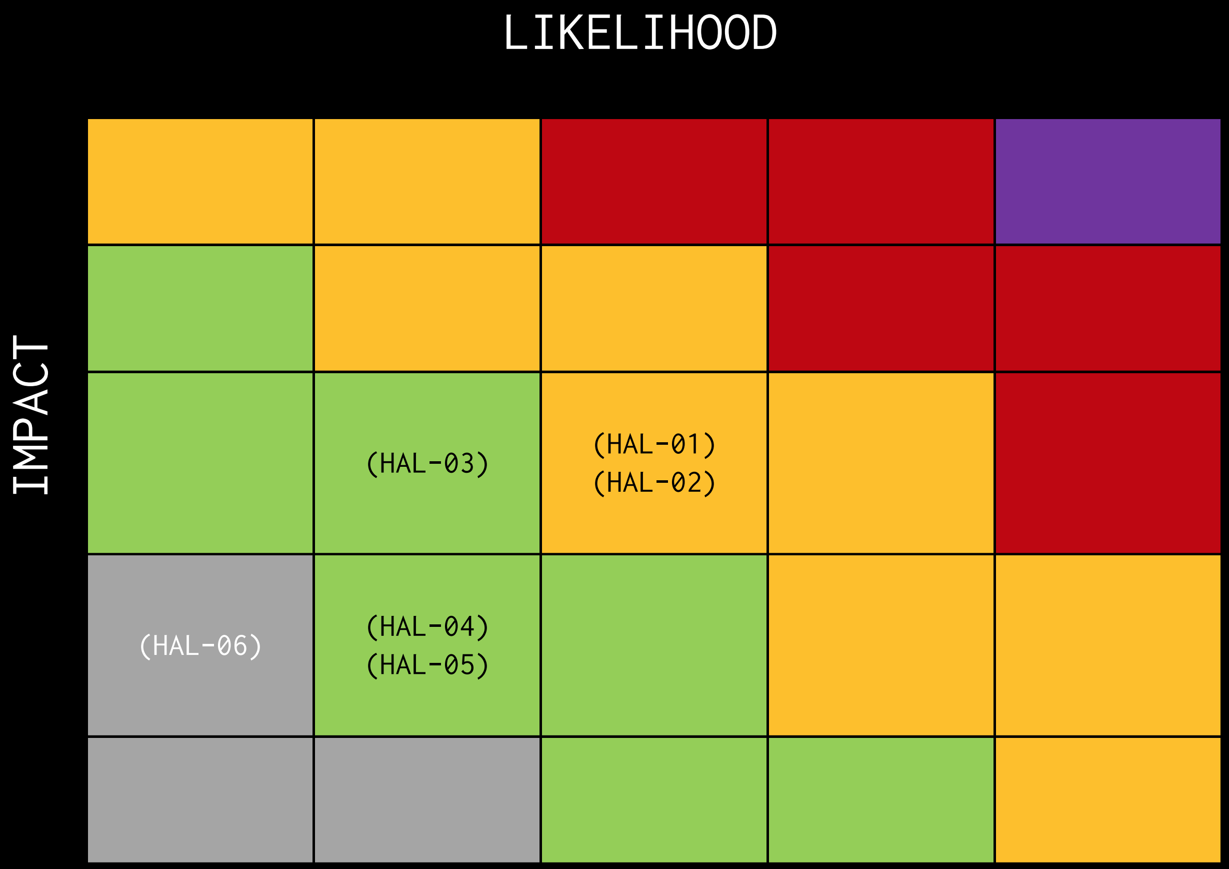
3. evm-bridge pallet

   (a) Repository: evm-bridge

   (b) Commit ID: 26ed9e88e773f5d628c01d558945cd38cd5a7d5a

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	2	3	1



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) INTEGER OVERFLOW	Medium	SOLVED - 08/19/2021
(HAL-02) TOTAL ISSUANCE NOT UPDATED ON MINT	Medium	ACKNOWLEDGED - 08/31/2021
(HAL-03) CASTING OVERFLOW	Low	SOLVED - 08/19/2021
(HAL-04) SLASH AMOUNT SANITY CHECK MISSING	Low	SOLVED - 08/19/2021
(HAL-05) CURRENCY ID VALIDATION MISSING	Low	ACKNOWLEDGED - 08/31/2021
(HAL-06) VECTOR CAPACITY VALIDATION MISSING	Informational	SOLVED - 08/19/2021

FINDINGS & TECH DETAILS

3.1 (HAL-01) INTEGER OVERFLOW - MEDIUM

Description:

An overflow happens when an arithmetic operation reaches the maximum size of a type. For instance in the ethereum_signable_message() method, if statement is summing up few u32 values which may end up overflowing the integer. In computer programming, an integer overflow occurs when an arithmetic operation attempts to create a numeric value that is outside of the range that can be represented with a given number of bits -- either larger than the maximum or lower than the minimum representable value.

Code Location:

Listing 1: modules/evm-accounts/src/lib.rs (Lines 182)

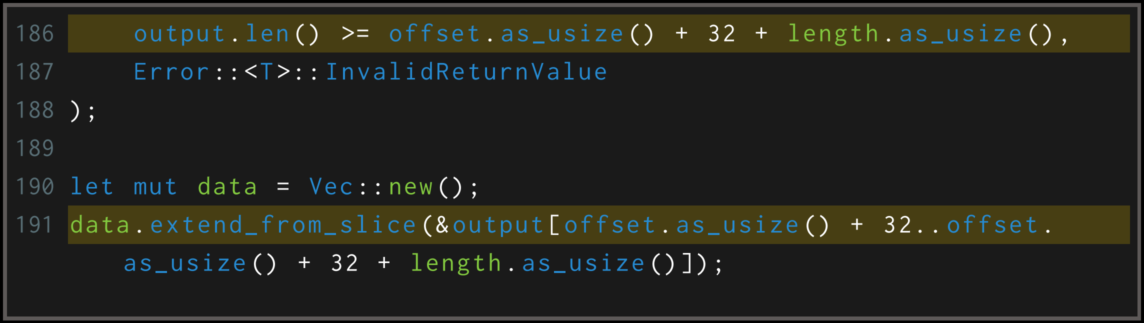
```
180 pub fn ethereum_signable_message(what: &[u8], extra: &[u8]) -> Vec
        <u8> {
181     let prefix = b”reef evm:”;
182     let mut l = prefix.len() + what.len() + extra.len();
183     let mut rev = Vec::new();
```

Listing 2: modules/evm-accounts/src/lib.rs (Lines 313,314)

```
312 pub fn to_ascii_hex(data: &[u8]) -> Vec<u8> {
313     let mut r = Vec::with_capacity(data.len() * 2);
314     let mut push_nibble = |n| r.push(if n < 10 { b'0' + n } else {
            b'a' - 10 + n });
```

Listing 3: modules/evm-bridge/src/lib.rs (Lines 183,186,191)

```
182 let offset = U256::from_big_endian(&output[0..32]);
183 let length = U256::from_big_endian(&output[offset.as_usize()..
        offset.as_usize() + 32]);
184 ensure!(
185     // output is 32-byte aligned. ensure total_length >= offset +
            string length + string data length.
```



```
    output.len() >= offset.as_usize() + 32 + length.as_usize(),
    Error::<T>::InvalidReturnValue
);

let mut data = Vec::new();
data.extend_from_slice(&output[offset.as_usize() + 32..offset.as_usize() + 32 + length.as_usize()]);
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendations:

It is recommended to use vetted safe math libraries for arithmetic operations consistently throughout the smart contract system. Consider replacing the addition and multiplication operators with Rust’s checked_add and checked_mul methods.

Remediation:

SOLVED: Reef fixed the issue in commit 6e4153498a28d03b8600739709cb200065c88781.

3.2 (HAL-02) TOTAL ISSUANCE NOT UPDATED ON MINT - MEDIUM

Description:

The update_balance dispatchable defined in modules/currencies/src/lib.rs does not update the total issuance of the currency (identified by user-supplied ID) which is minted to the target address. This may lead to discrepancies in token data.

Code Location:

Listing 4: modules/currencies/src/lib.rs (Lines 168)

```
159 #[pallet::weight(T::WeightInfo::update_balance_non_native_currency())]
160 pub fn update_balance(
161     origin: OriginFor<T>,
162     who: <T::Lookup as StaticLookup>::Source,
163     currency_id: CurrencyIdOf<T>,
164     amount: AmountOf<T>,
165 ) -> DispatchResultWithPostInfo {
166     ensure_root(origin)?;
167     let dest = T::Lookup::lookup(who)?;
168     <Self as MultiCurrencyExtended<T::AccountId>>::update_balance(currency_id, &dest, amount)?;
169     Ok(().into())
170 }
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendations:

Total issuance should be updated every time tokens are minted or burned.

Remediation Plan:

ACKNOWLEDGED: Reef states that the affected function is sudo only and will be deprecated in a future release.

3.3 (HAL-03) CASTING OVERFLOW - LOW

Description:

When converting or casting between types, an “overflow”/wrapping may occur and result in logic bugs leading to thread panic.

The decode_string utility method defined in modules/evm-bridge/src/lib.rs does not validate if the values of the offset and length variables can be cast to the usize type. Although the method is not exported and available externally, the method is vulnerable still and the risk could increase in the future if the method is used before it’s patched.

Code Location:

Listing 5: modules/evm-bridge/src/lib.rs (Lines 183,186,191)

```
182 let offset = U256::from_big_endian(&output[0..32]);
183 let length = U256::from_big_endian(&output[offset.as_usize()..offset.as_usize() + 32]);
184 ensure!(
185     // output is 32-byte aligned. ensure total_length >= offset + string length + string data length.
186     output.len() >= offset.as_usize() + 32 + length.as_usize(),
187     Error::<T>::InvalidReturnValue
188 );
189
190 let mut data = Vec::new();
191 data.extend_from_slice(&output[offset.as_usize() + 32..offset.as_usize() + 32 + length.as_usize()]);
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendations:

Check the value against maximum type value before casting.

Listing 6

```
if (x <= usize::MAX) {
    // logic...
}
```

Remediation:

SOLVED: Reef fixed the issue in commit 313439bb7940afa0f0d5060fbcbbe26d5a3e5298

3.4 (HAL-04) SLASH AMOUNT VALIDATION MISSING - LOW

Description:

The slash_reserved method defined in modules/currencies/src/lib.rs does not validate if the value of the user-supplied value parameter exceeds the actual balance of the account owned by the address that is to have its ERC20 tokens slashed.

Code Location:

Listing 7: modules/currencies/src/lib.rs (Lines 396)

```
394 fn slash_reserved(currency_id: Self::CurrencyId, who: &T::
    AccountId, value: Self::Balance) -> Self::Balance {
395     match currency_id {
396         CurrencyId::ERC20(_) => value,
397         CurrencyId::Token(TokenSymbol::REEF) => T::NativeCurrency
                ::slash_reserved(who, value),
398         _ => T::MultiCurrency::slash_reserved(currency_id, who,
                value),
399     }
400 }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendations:

The slashed amount should always be lesser or equal to the account balance that is to be slashed.

Remediation:

SOLVED: Reef fixed the issue in commit (bd43bec58890be763b32bfdfd18ba85a8c0ef9e5)[http

3.5 (HAL-05) CURRENCY ID VALIDATION MISSING - LOW

Description:

Many dispatchables and helper methods defined in modules/currencies/src/lib.rs do not check if the user-supplied currency ID matches any of the existing ones before calling the possibly resource-intensive underlying utility functions.

Code Location:

Listing 8: modules/evm-accounts/src/lib.rs (Lines 125)

```
121 #[pallet::weight(T::WeightInfo::transfer_non_native_currency())]
122 pub fn transfer(
123     origin: OriginFor<T>,
124     dest: <T::Lookup as StaticLookup>::Source,
125     currency_id: CurrencyIdOf<T>,
126     #[pallet::compact] amount: BalanceOf<T>,
127 ) -> DispatchResultWithPostInfo {
128     let from = ensure_signed(origin)?;
129     let to = T::Lookup::lookup(dest)?;
130     <Self as MultiCurrency<T::AccountId>>::transfer(currency_id, &
        from, &to, amount)?;
131     Ok(().into())
132 }
```

List of all the functions that fail to validate the currency ID:

Listing 9: (Lines 2,3)

```
1 auditor@halborn:~/projects/reef/reef-chain/modules/currencies$ \
2 > grep -ne 'fn.*CurrencyId' src/lib.rs \
3 > | cut -d '-' -f 1
4 178:     fn minimum_balance(currency_id: Self::CurrencyId)
5 186:     fn total_issuance(currency_id: Self::CurrencyId)
6 199:     fn total_balance(currency_id: Self::CurrencyId, who: &T::
       AccountId)
```

```
217:     fn free_balance(currency_id: Self::CurrencyId, who: &T::AccountId)
235:     fn ensure_can_withdraw(currency_id: Self::CurrencyId, who: &T::AccountId, amount: Self::Balance)
290:     fn deposit(currency_id: Self::CurrencyId, who: &T::AccountId, amount: Self::Balance)
303:     fn withdraw(currency_id: Self::CurrencyId, who: &T::AccountId, amount: Self::Balance)
316:     fn can_slash(currency_id: Self::CurrencyId, who: &T::AccountId, amount: Self::Balance)
324:     fn slash(currency_id: Self::CurrencyId, who: &T::AccountId, amount: Self::Balance)
336:     fn update_balance(currency_id: Self::CurrencyId, who: &T::AccountId, by_amount: Self::Amount)
376:     fn remove_lock(lock_id: LockIdentifier, currency_id: Self::CurrencyId, who: &T::AccountId)
386:     fn can_reserve(currency_id: Self::CurrencyId, who: &T::AccountId, value: Self::Balance)
394:     fn slash_reserved(currency_id: Self::CurrencyId, who: &T::AccountId, value: Self::Balance)
402:     fn reserved_balance(currency_id: Self::CurrencyId, who: &T::AccountId)
423:     fn reserve(currency_id: Self::CurrencyId, who: &T::AccountId, value: Self::Balance)
445:     fn unreserve(currency_id: Self::CurrencyId, who: &T::AccountId, value: Self::Balance)
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendations:

It is recommended to validate all user-supplied input in order to avoid executing unnecessary operations and mitigate the risk of resource exhaustion.

Remediation Plan:

ACKNOWLEDGED: Reef states that there is only 1 currency id in use, and there likely won’t be more going forward.

3.6 (HAL-06) VECTOR CAPACITY VALIDATION MISSING - INFORMATIONAL

Description:

The to_ascii_hex utility function defined in modules/evm-accounts/src/lib.rs when creating a new Vec<u8> from the user-supplied data slice with a Vec::with_capacity method does not validate if the capacity of the new vector exceeds the maximum allowed capacity.

Code Location:

Listing 10: modules/currencies/src/lib.rs (Lines 313)

```rust
312 pub fn to_ascii_hex(data: &[u8]) -> Vec<u8> {
313     let mut r = Vec::with_capacity(data.len() * 2);
314     let mut push_nibble = |n| r.push(if n < 10 { b'0' + n } else {
            b'a' - 10 + n });
315     for &b in data.iter() {
316         push_nibble(b / 16);
317         push_nibble(b % 16);
318     }
319     r
320 }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendations:

Validate if the new capacity (data.len()* 2) does not exceed isize::MAX bytes.

Remediation:

SOLVED: Reef fixed the issue in commit (6b826f7ca16d1a30f3fa55f0606d0b94b69b2b3a)[http

FUZZING

Introduction:

Fuzzing or fuzz testing is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a computer program. The program is then monitored for exceptions such as crashes, failing built-in code assertions, or potential memory leaks.

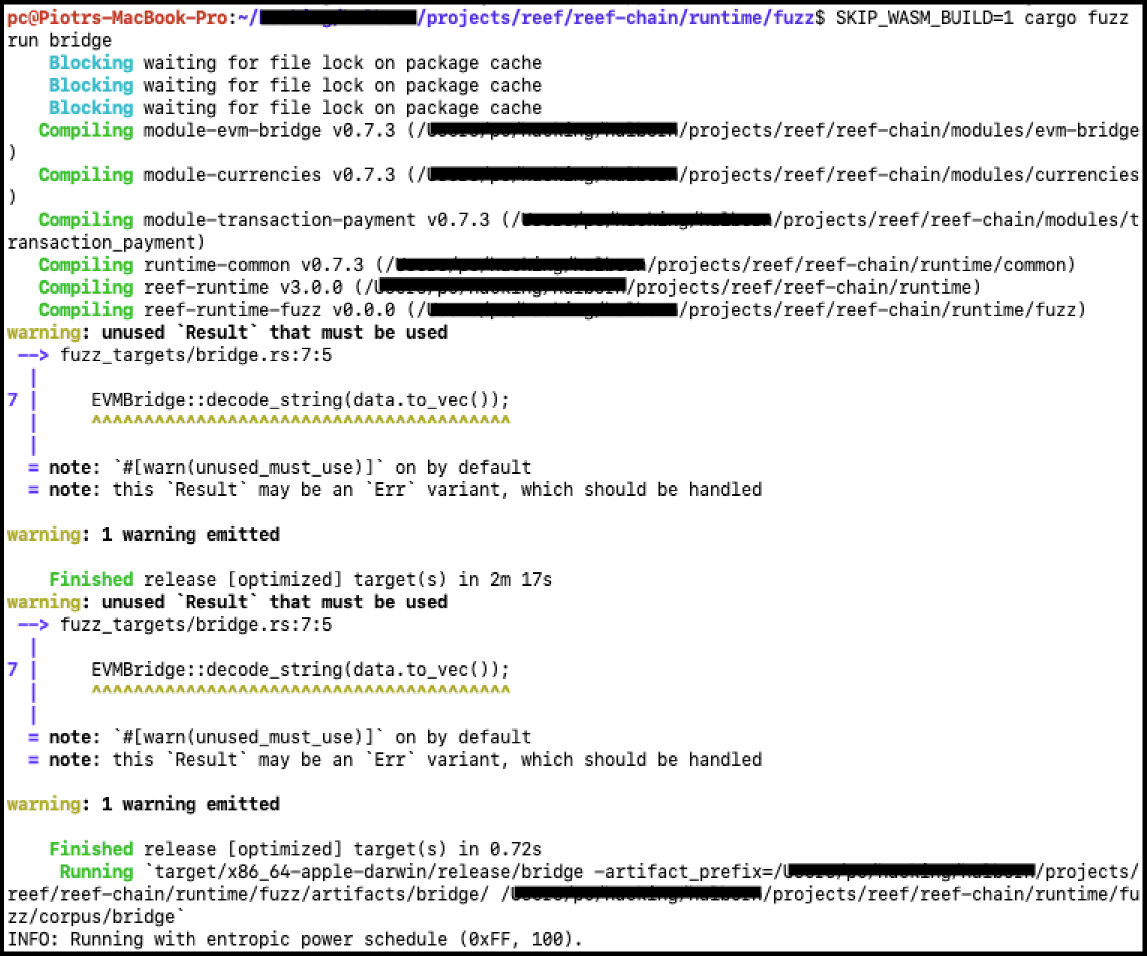
Halborn custom-built scripts leverage libFuzzer and cargo-fuzz for in-process, coverage-guided fuzz testing.

The fuzzer tracks which areas of the code are reached and generates mutations on the corpus of input data to maximize the code coverage. The code coverage information is provided by LLVM's SanitizerCoverage instrumentation.

Description:

Halborn used custom fuzzing scripts, tailored to the specifics of Substrate and the Reef protocol. The methods targeted were the ones accepting vectors of bytes as input because they are potentially most likely to be vulnerable to memory-related and indexing issues.

PoC:



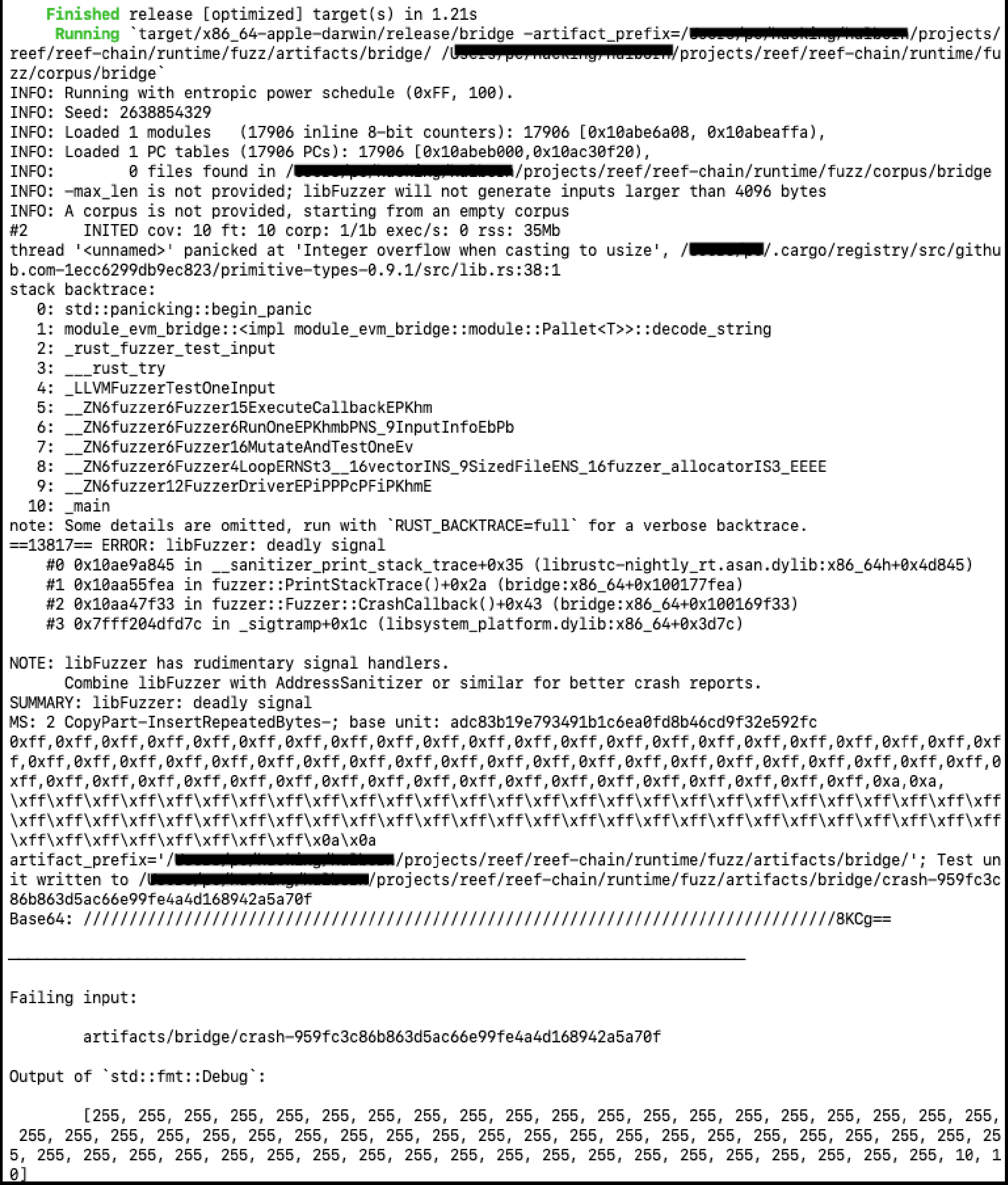
```
pc@Piotrs-MacBook-Pro:~/[redacted]/projects/reef/reef-chain/runtime/fuzz$ SKIP_WASM_BUILD=1 cargo fuzz
run bridge
    Blocking waiting for file lock on package cache
    Blocking waiting for file lock on package cache
    Blocking waiting for file lock on package cache
   Compiling module-evm-bridge v0.7.3 (/[redacted]/projects/reef/reef-chain/modules/evm-bridge
)
   Compiling module-currencies v0.7.3 (/[redacted]/projects/reef/reef-chain/modules/currencies
)
   Compiling module-transaction-payment v0.7.3 (/[redacted]/projects/reef/reef-chain/modules/t
ransaction_payment)
   Compiling runtime-common v0.7.3 (/[redacted]/projects/reef/reef-chain/runtime/common)
   Compiling reef-runtime v3.0.0 (/[redacted]/projects/reef/reef-chain/runtime)
   Compiling reef-runtime-fuzz v0.0.0 (/[redacted]/projects/reef/reef-chain/runtime/fuzz)
warning: unused `Result` that must be used
 --> fuzz_targets/bridge.rs:7:5
  |
7 |     EVMBridge::decode_string(data.to_vec());
  |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  |
  = note: `#[warn(unused_must_use)]` on by default
  = note: this `Result` may be an `Err` variant, which should be handled

warning: 1 warning emitted

    Finished release [optimized] target(s) in 2m 17s
warning: unused `Result` that must be used
 --> fuzz_targets/bridge.rs:7:5
  |
7 |     EVMBridge::decode_string(data.to_vec());
  |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  |
  = note: `#[warn(unused_must_use)]` on by default
  = note: this `Result` may be an `Err` variant, which should be handled

warning: 1 warning emitted

    Finished release [optimized] target(s) in 0.72s
     Running `target/x86_64-apple-darwin/release/bridge -artifact_prefix=/[redacted]/projects/
reef/reef-chain/runtime/fuzz/artifacts/bridge/ /[redacted]/projects/reef/reef-chain/runtime/fu
zz/corpus/bridge`
INFO: Running with entropic power schedule (0xFF, 100).
```



```
     Finished release [optimized] target(s) in 1.21s
      Running `target/x86_64-apple-darwin/release/bridge -artifact_prefix=/[redacted]/projects/
reef/reef-chain/runtime/fuzz/artifacts/bridge/ /[redacted]/projects/reef/reef-chain/runtime/fu
zz/corpus/bridge`
INFO: Running with entropic power schedule (0xFF, 100).
INFO: Seed: 2638854329
INFO: Loaded 1 modules   (17906 inline 8-bit counters): 17906 [0x10abe6a08, 0x10abeaffa),
INFO: Loaded 1 PC tables (17906 PCs): 17906 [0x10abeb000,0x10ac30f20),
INFO:        0 files found in /[redacted]/projects/reef/reef-chain/runtime/fuzz/corpus/bridge
INFO: -max_len is not provided; libFuzzer will not generate inputs larger than 4096 bytes
INFO: A corpus is not provided, starting from an empty corpus
#2      INITED cov: 10 ft: 10 corp: 1/1b exec/s: 0 rss: 35Mb
thread '<unnamed>' panicked at 'Integer overflow when casting to usize', /[redacted]/.cargo/registry/src/githu
b.com-1ecc6299db9ec823/primitive-types-0.9.1/src/lib.rs:38:1
stack backtrace:
   0: std::panicking::begin_panic
   1: module_evm_bridge::<impl module_evm_bridge::module::Pallet<T>>::decode_string
   2: _rust_fuzzer_test_input
   3: ___rust_try
   4: _LLVMFuzzerTestOneInput
   5: __ZN6fuzzer6Fuzzer15ExecuteCallbackEPKhm
   6: __ZN6fuzzer6Fuzzer6RunOneEPKhmbPNS_9InputInfoEbPb
   7: __ZN6fuzzer6Fuzzer16MutateAndTestOneEv
   8: __ZN6fuzzer6Fuzzer4LoopERNSt3__16vectorINS_9SizedFileENS_16fuzzer_allocatorIS3_EEEE
   9: __ZN6fuzzer12FuzzerDriverEPiPPPcPFiPKhmE
  10: _main
note: Some details are omitted, run with `RUST_BACKTRACE=full` for a verbose backtrace.
==13817== ERROR: libFuzzer: deadly signal
    #0 0x10ae9a845 in __sanitizer_print_stack_trace+0x35 (librustc-nightly_rt.asan.dylib:x86_64h+0x4d845)
    #1 0x10aa55fea in fuzzer::PrintStackTrace()+0x2a (bridge:x86_64+0x100177fea)
    #2 0x10aa47f33 in fuzzer::Fuzzer::CrashCallback()+0x43 (bridge:x86_64+0x100169f33)
    #3 0x7fff204dfd7c in _sigtramp+0x1c (libsystem_platform.dylib:x86_64+0x3d7c)

NOTE: libFuzzer has rudimentary signal handlers.
      Combine libFuzzer with AddressSanitizer or similar for better crash reports.
SUMMARY: libFuzzer: deadly signal
MS: 2 CopyPart-InsertRepeatedBytes-; base unit: adc83b19e793491b1c6ea0fd8b46cd9f32e592fc
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xf
f,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0
xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xa,0xa,
\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff
\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff
\xff\xff\xff\xff\xff\xff\xff\xff\x0a\x0a
artifact_prefix='/[redacted]/projects/reef/reef-chain/runtime/fuzz/artifacts/bridge/'; Test un
it written to /[redacted]/projects/reef/reef-chain/runtime/fuzz/artifacts/bridge/crash-959fc3c
86b863d5ac66e99fe4a4d168942a5a70f
Base64: //////////////////////////////////////////////////////////////////////////////////8KCg==

────────────────────────────────────────────────────────────────────────────────────


Failing input:

        artifacts/bridge/crash-959fc3c86b863d5ac66e99fe4a4d168942a5a70f

Output of `std::fmt::Debug`:

        [255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 25
5, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 10, 1
0]
```

Results:

Between the time constraints and lack of advanced memory manipulation in the source code one vulnerability was detected-casting overflow in the decode_string utility method in modules/evm-bridge/src/lib.rs.

AUTOMATED TESTING

5.1 VULNERABILITIES AUTOMATIC DETECTION

Description:

Halborn used automated security scanners to assist with detection of well known security issues and vulnerabilities. Among the tools used was cargo audit, a security scanner for vulnerabilities reported to the RustSec Advisory Database. All vulnerabilities published in https://crates.io are stored in a repository named The RustSec Advisory Database. cargo audit is a human-readable version of the advisory database which performs a scanning on Cargo.lock. To better assist the developers maintaining this code, the auditors are including the output with the dependencies tree, and this is included in the cargo audit output to better know the dependencies affected by unmaintained and vulnerable crates.

Results:

ID	**Package**	**Categories**
RUSTSEC-2021-0067	cranelift-codegen	code-execution, memory-corruption
RUSTSEC-2021-0066	evm-core	denial-of-service
RUSTSEC-2021-0070	nalgebra	memory-corruption, memory-exposure
RUSTSEC-2021-0070	nalgebra	memory-corruption, memory-exposure
RUSTSEC-2021-0013	raw-cpuid	memory-corruption, denial-of-service

THANK YOU FOR CHOOSING

// HALBORN