SAULIDITY

FANFURY

2022 | SMART CONTRACT SECURITY ANALYSIS

PREPARED BY

Saulidity

PRESENTED TO

Fanfury

2022

SAULIDITY

SECURITY REPORT

Smart Contract Audit

saulidity.com

Saulidity

@Saulidity

DISCLAIMER

This report does not constitute financial advice, and Saulidity is not accountable or liable for any negative consequences resulting from this report, nor may Saulidity be held liable in any way. You agree to the terms of this disclaimer by reading any part of the report. If you do not agree to the terms, please stop reading this report immediately and delete and destroy any and all copies of this report that you have downloaded and/or printed. This report was entirely based on information given by the audited party and facts that existed prior to the audit. Saulidity and/or its auditors cannot be held liable for any outcome, including modifications (if any) made to the contract(s) for the audit that was completed. No modifications have been made to the contract(s) by the Saulidity team unless it is indicated explicitly. The audit does not include the project team, website, logic, or tokenomics, but if it does, it will be indicated explicitly. The security is evaluated only on the basis of smart contracts only. There were no security checks performed on any apps or activities. There has not been a review of any product codes. It is assumed by Saulidity that the information and materials given were not tampered with, censored, or misrepresented. Even if this report exists and Saulidity makes every effort to uncover any security flaws, you should not rely completely on it and should conduct your own independent research. Saulidity hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Saulidity, for any amount or kind of loss or damage that may result to you or any other person or any kind of company, community, association and institution. Saulidity is the exclusive owner of this report, and it is published by Saulidity. Without Saulidity's express written authorization, use of this report for any reason other than a security interest in the individual contacts, or use of sections of this report, is forbidden.

Table of Contents

Saulidity

Saulidity is a renowned cybersecurity firm specializing in the analysis and development of Smart contracts. Saulidity, as a full-service security organization, can help with a variety of audits and project development.

In a market where confidence and trust are key, a genuine project may simply increase its user base enormously with an official audit performed by Saulidity.

Introduction

For a thorough understanding of the audit, please read the entire document.

The goal of the audit was to find any potential smart contract security problems and vulnerabilities.
The information in this report should be used to understand the smart contract's risk exposure and as a guide to improving the smart contract's security posture by addressing the concerns that were discovered.

During our audit, we conducted a thorough inquiry using automated analysis and manual review approaches.

The security specialists did a complete study independently of one another in order to uncover any security issues in the contracts as comprehensively as feasible. For optimum security and professionalism, all of our audits are undertaken by at least two independent auditors.

The project's website, logic, or tokenomics have not been vetted by the Saulidity team.

Information

We analyze smart contracts for both well-known and more specific vulnerabilities.

Here are some of the most well-known vulnerabilities that are taken into account but not limited to:

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Style guide violation
- Transfer forwards all gas
- API violation
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

Information

Security experts of Saulidity performed a manual review of the code and conducted security scans in relation to the scope of the smart contract audit by prioritizing accuracy, efficiency and practicality.

The most essential aspects were tested and verified by conducting:

- Manual code review and walkthrough.
- Scanning of files for vulnerabilities.
- Identifying undefined behaviour.
- Identifying unsafe code.

The purpose of this audit to identify potential security issues with the smart contracts.

Appendix

Vulnerabilities can be divided into four threat levels: Critical, High, Medium and Low. The classification is mainly based on the impact, likelihood of utilization and other factors.

Critical flaws can result in the loss of assets or the alteration of data and are often simple to exploit.

High-level vulnerabilities are challenging to exploit, but they can have a big influence on how smart contracts are executed, such as giving the public access to key features.

Although medium-level vulnerabilities should be fixed, they generally cannot result in the loss of assets or the manipulation of data.

Low-level flaws are typically caused by code fragments that are out-of-date, useless, etc. and cannot significantly affect execution.

SC Weakness Registry

ITEM	DESCRIPTION
Default Visibility	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.
Integer Overflow and Underflow	If unchecked math is used, all math operations should be safe from overflows and underflows.
Outdated Compiler Version	It is recommended to use a recent version of the Solidity compiler.
Floating Pragma	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.
Unchecked Call Return Value	The return value of a message call should be checked.
Access Control & Authorization	Ownership takeover should not bepossible. All crucial functions should be protected. Users could not affect data that belongs to other users.
Selfdestruct	The contract should not be destroyed until it has funds belonging to users.
Check-Effect-Interaction	CEI pattern should be followed if the code performs any external call.

SC Weakness Registry

ITEM	DESCRIPTION
Uninitialized Storage Pointer	Storage type should be set explicitly if the compiler version is < 0.5.0.
Assert Violation	Properly functioning code should never reach a failing assert statement.
Deprecated Solidity Functions	Deprecated built-in functions should never be used.
Delegatecall to Untrusted Callee	Delegatecalls should only be allowed to trusted addresses.
Denial of Service	Execution of the code should never be blocked by a specific contract state unless it is required.
Race Conditions	Race Conditions and Transactions Order Dependency should not be possible.
Authorization through tx.origin	tx.origin should not be used for authorization.
Block values as a proxy for time	Block numbers should not be used for time calculations.

SC Weakness Registry

ITEM	DESCRIPTION
Signature Unique Id	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.
Shadowing State Variable	State variables should not be shadowed.
Weak Sources of Randomness	Random values should never be generated from Chain Attributes.
Incorrect Inheritance Order	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.
Calls Only to Trusted Addresses	All external calls should be performed only to trusted addresses.
Presence of unused variables	The code should not contain unused variables if this is not justified by design.

Audit & Project Information

Project Name	Fanfury
Contract(s)	https://github.com/FanFury/liquidity-contracts
Report ID	rsSAUL01 V1.1
Website	fury.fan
Contact	Adrian Foxcraft
Contact Information	Telegram: @nimrostafarian @Foxcraft_FURY
Code language	Rust

Analysis

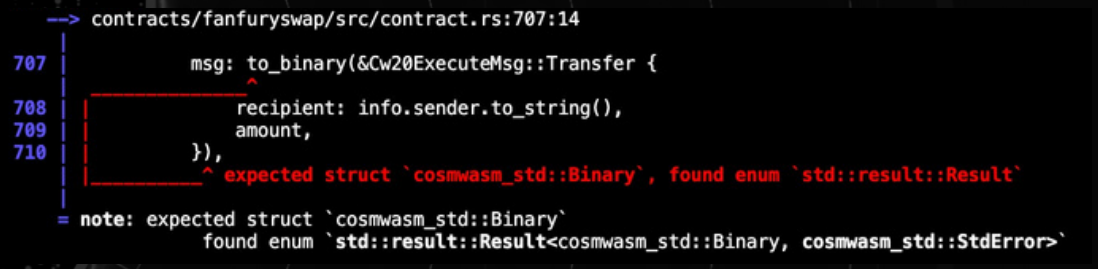
Compiler Error

Issue/Title:

Compiler error.

Description:

Expected type did not match the received type.



Analysis

Compiler Error

example:

```
fn plus_one(x: i32) -> i32 {
    x + 1
}

plus_one("Not a number");
//       ^^^^^^^^^^^^^^^ expected `i32`, found `&str`

if "Not a bool" {
// ^^^^^^^^^^^^ expected `bool`, found `&str`
}

let x: f32 = "Not a float";
//     ---   ^^^^^^^^^^^^^ expected `f32`, found `&str`
//     |
//     expected due to this
```

Commment:

This error typically take places if an expression was used in a place where the compiler expected an expression of a different type.

It can occur in several cases, the most common being when calling a function and passing an argument which has a different type than the matching type in the function declaration.

Status: Mitigated

Analysis

Arithmetic Error

Issue/Title:

Arithmetic Error

Severity:

Low

Description:

Integer overflow/underflow may happen when an arithmetic operation tries 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).

```
{
    if amount_remaining > Uint128::zero() {
        if bond.1.bonded_amount > amount_remaining {
            unbonded_amount += amount_remaining;
            updated_bond.bonded_amount -= amount_remaining;
            amount_remaining = Uint128::zero();
            updated_bonds.push(updated_bond);
        } else {
            unbonded_amount += bond.1.bonded_amount;
            amount_remaining -= bond.1.bonded_amount;
        }
```

Analysis

Arithmetic Error

Commment:

It is recommended to use vetted safe math libraries for arithmetic operations consistently throughout the smart contract system. Consider using Rust safe arithmetic functions for primitives rather than standard arithmetic operators. Even if integer overflows and underflows may not cause Rust to panic in the release mode, the consequences could be dire if the result of those operations is used in financial calculations.

Status: Mitigated

Analysis

Unsafe Code Usage

Issue/Title:

Unsafe Code Usage

Severity:

Lowest/Optimization/Informational

Description:

Statistics related to the usage of unsafe code in a core Rust codebase and its dependencies.

Analysis

Unsafe Code Usage

Appendix

🔒 = No 'unsafe' usage found, declares
#![forbid(unsafe_code)]

❓ = No 'unsafe' usage found, missing
#![forbid(unsafe_code)]

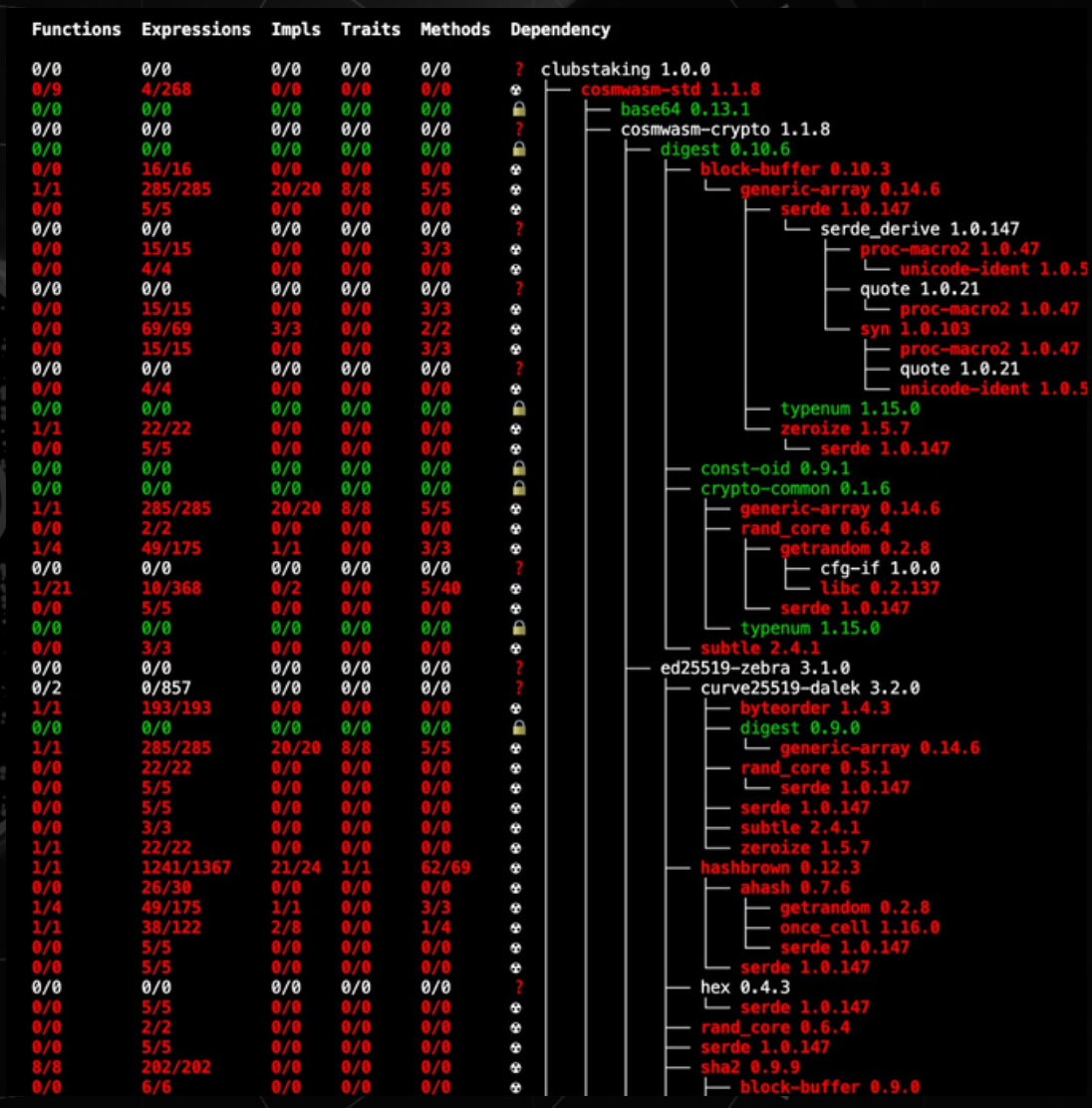
☢ = 'unsafe' usage found

Metric output format: x/y
x = unsafe code used by the build
y = total unsafe code found in the crate

Analysis

Unsafe Code Usage

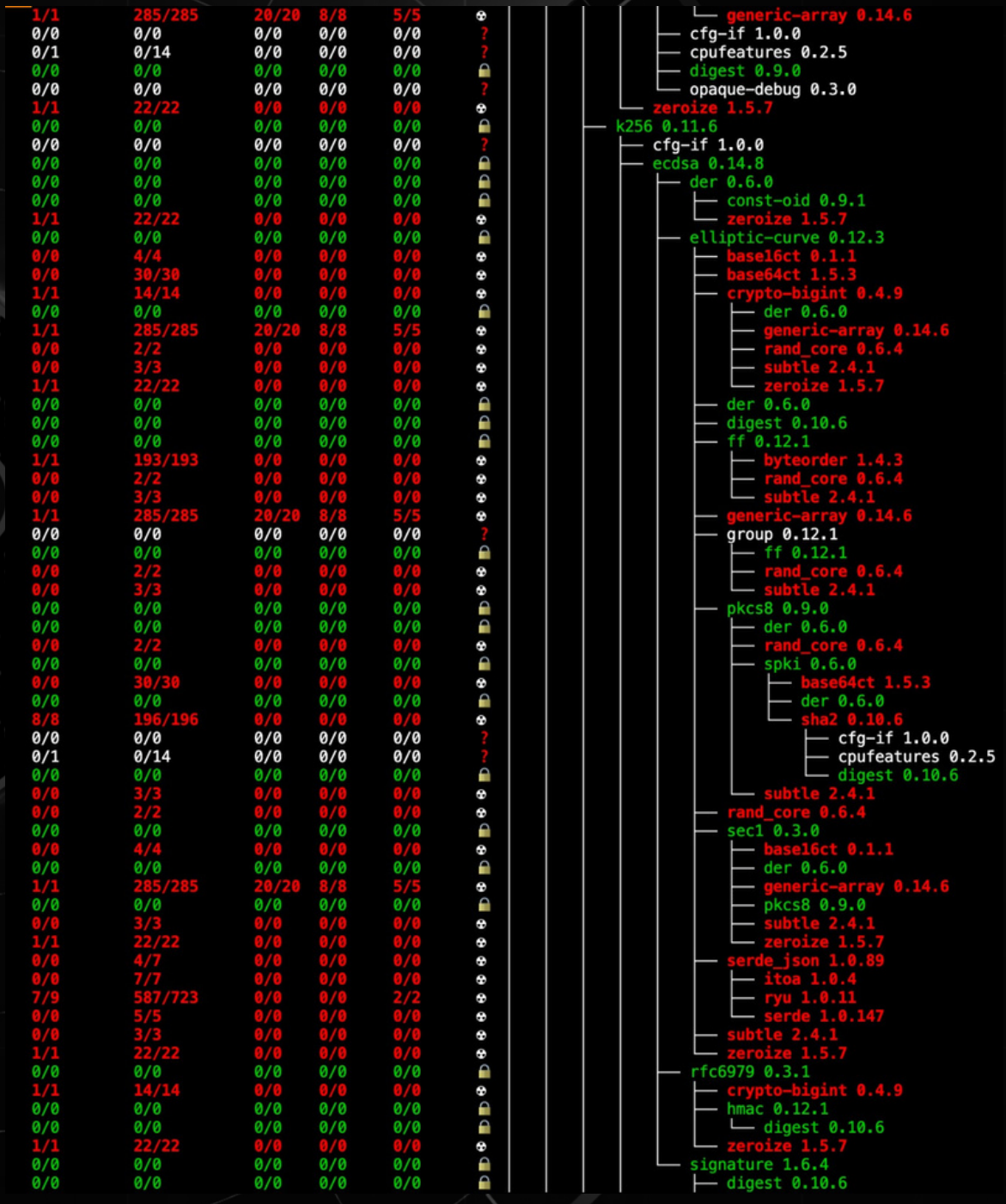
Results:



Functions	Expressions	Impls	Traits	Methods	Dependency
0/0	0/0	0/0	0/0	0/0	? clubstaking 1.0.0
0/9	4/268	0/0	0/0	0/0	☢ ├── cosmwasm-std 1.1.8
0/0	0/0	0/0	0/0	0/0	🔒 │ ├── base64 0.13.1
0/0	0/0	0/0	0/0	0/0	? │ ├── cosmwasm-crypto 1.1.8
0/0	0/0	0/0	0/0	0/0	🔒 │ │ ├── digest 0.10.6
0/0	16/16	0/0	0/0	0/0	☢ │ │ │ ├── block-buffer 0.10.3
1/1	285/285	20/20	8/8	5/5	☢ │ │ │ │ └── generic-array 0.14.6
0/0	5/5	0/0	0/0	0/0	☢ │ │ │ │ ├── serde 1.0.147
0/0	0/0	0/0	0/0	0/0	? │ │ │ │ │ └── serde_derive 1.0.147
0/0	15/15	0/0	0/0	3/3	☢ │ │ │ │ │ ├── proc-macro2 1.0.47
0/0	4/4	0/0	0/0	0/0	☢ │ │ │ │ │ │ └── unicode-ident 1.0.5
0/0	0/0	0/0	0/0	0/0	? │ │ │ │ │ ├── quote 1.0.21
0/0	15/15	0/0	0/0	3/3	☢ │ │ │ │ │ │ └── proc-macro2 1.0.47
0/0	69/69	3/3	0/0	2/2	☢ │ │ │ │ │ └── syn 1.0.103
0/0	15/15	0/0	0/0	3/3	☢ │ │ │ │ │ ├── proc-macro2 1.0.47
0/0	0/0	0/0	0/0	0/0	? │ │ │ │ │ ├── quote 1.0.21
0/0	4/4	0/0	0/0	0/0	☢ │ │ │ │ │ └── unicode-ident 1.0.5
0/0	0/0	0/0	0/0	0/0	🔒 │ │ │ │ ├── typenum 1.15.0
1/1	22/22	0/0	0/0	0/0	☢ │ │ │ │ └── zeroize 1.5.7
0/0	5/5	0/0	0/0	0/0	☢ │ │ │ │ └── serde 1.0.147
0/0	0/0	0/0	0/0	0/0	🔒 │ │ │ ├── const-oid 0.9.1
0/0	0/0	0/0	0/0	0/0	🔒 │ │ │ ├── crypto-common 0.1.6
1/1	285/285	20/20	8/8	5/5	☢ │ │ │ │ ├── generic-array 0.14.6
0/0	2/2	0/0	0/0	0/0	☢ │ │ │ │ ├── rand_core 0.6.4
1/4	49/175	1/1	0/0	3/3	☢ │ │ │ │ │ ├── getrandom 0.2.8
0/0	0/0	0/0	0/0	0/0	? │ │ │ │ │ │ ├── cfg-if 1.0.0
1/21	10/368	0/2	0/0	5/40	☢ │ │ │ │ │ │ └── libc 0.2.137
0/0	5/5	0/0	0/0	0/0	☢ │ │ │ │ │ └── serde 1.0.147
0/0	0/0	0/0	0/0	0/0	🔒 │ │ │ │ └── typenum 1.15.0
0/0	3/3	0/0	0/0	0/0	☢ │ │ │ └── subtle 2.4.1
0/0	0/0	0/0	0/0	0/0	? │ │ └── ed25519-zebra 3.1.0
0/2	0/857	0/0	0/0	0/0	? │ │ ├── curve25519-dalek 3.2.0
1/1	193/193	0/0	0/0	0/0	☢ │ │ │ ├── byteorder 1.4.3
0/0	0/0	0/0	0/0	0/0	🔒 │ │ │ ├── digest 0.9.0
1/1	285/285	20/20	8/8	5/5	☢ │ │ │ │ └── generic-array 0.14.6
0/0	22/22	0/0	0/0	0/0	☢ │ │ │ ├── rand_core 0.5.1
0/0	5/5	0/0	0/0	0/0	☢ │ │ │ │ └── serde 1.0.147
0/0	5/5	0/0	0/0	0/0	☢ │ │ │ ├── serde 1.0.147
0/0	3/3	0/0	0/0	0/0	☢ │ │ │ ├── subtle 2.4.1
1/1	22/22	0/0	0/0	0/0	☢ │ │ │ └── zeroize 1.5.7
1/1	1241/1367	21/24	1/1	62/69	☢ │ │ ├── hashbrown 0.12.3
0/0	26/30	0/0	0/0	0/0	☢ │ │ │ ├── ahash 0.7.6
1/4	49/175	1/1	0/0	3/3	☢ │ │ │ │ ├── getrandom 0.2.8
1/1	38/122	2/8	0/0	1/4	☢ │ │ │ │ ├── once_cell 1.16.0
0/0	5/5	0/0	0/0	0/0	☢ │ │ │ │ └── serde 1.0.147
0/0	5/5	0/0	0/0	0/0	☢ │ │ │ └── serde 1.0.147
0/0	0/0	0/0	0/0	0/0	? │ │ ├── hex 0.4.3
0/0	5/5	0/0	0/0	0/0	☢ │ │ │ └── serde 1.0.147
0/0	2/2	0/0	0/0	0/0	☢ │ │ ├── rand_core 0.6.4
0/0	5/5	0/0	0/0	0/0	☢ │ │ ├── serde 1.0.147
8/8	202/202	0/0	0/0	0/0	☢ │ │ ├── sha2 0.9.9
0/0	6/6	0/0	0/0	0/0	☢ │ │ │ ├── block-buffer 0.9.0

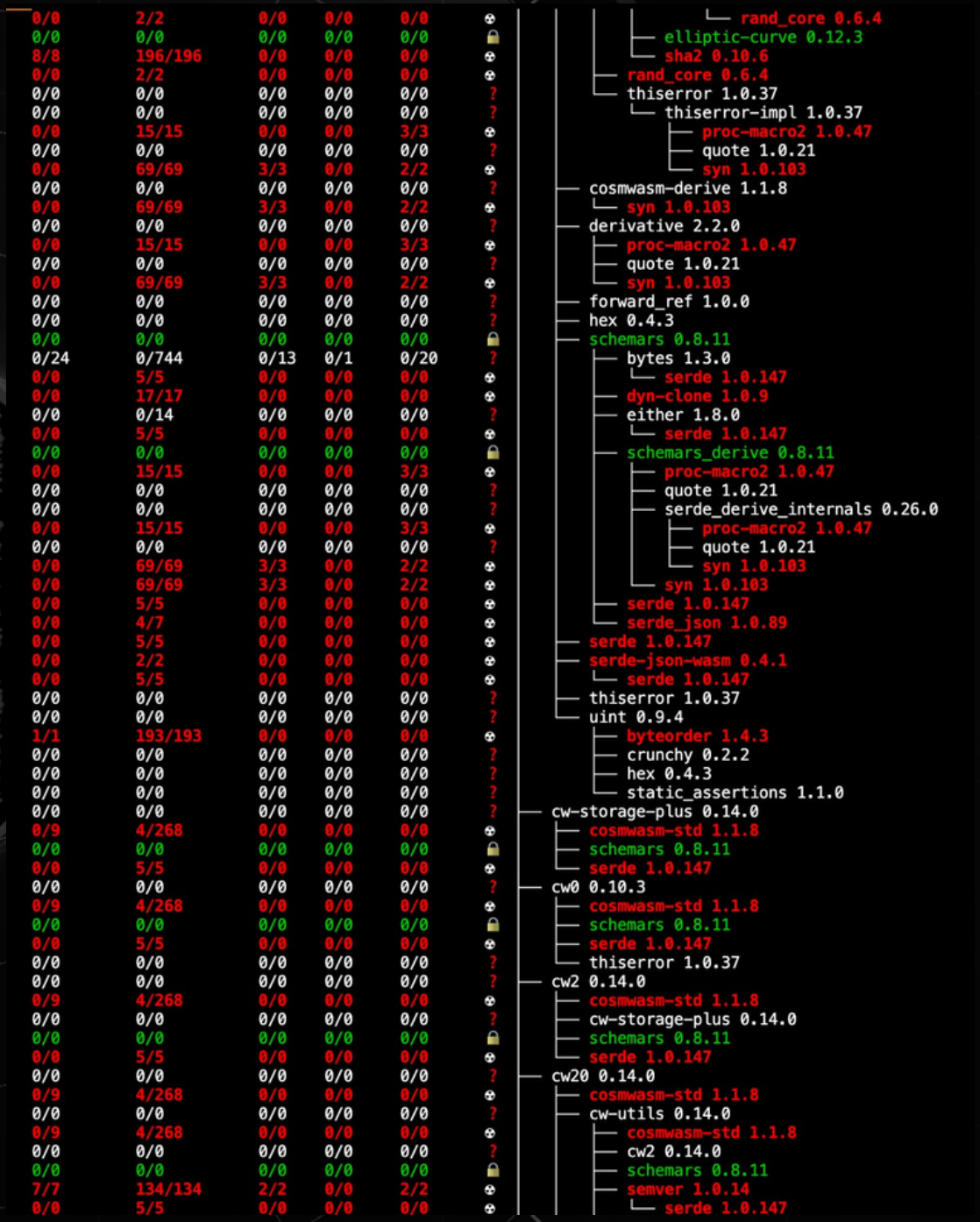
Analysis

Unsafe Code Usage



Analysis

Unsafe Code Usage

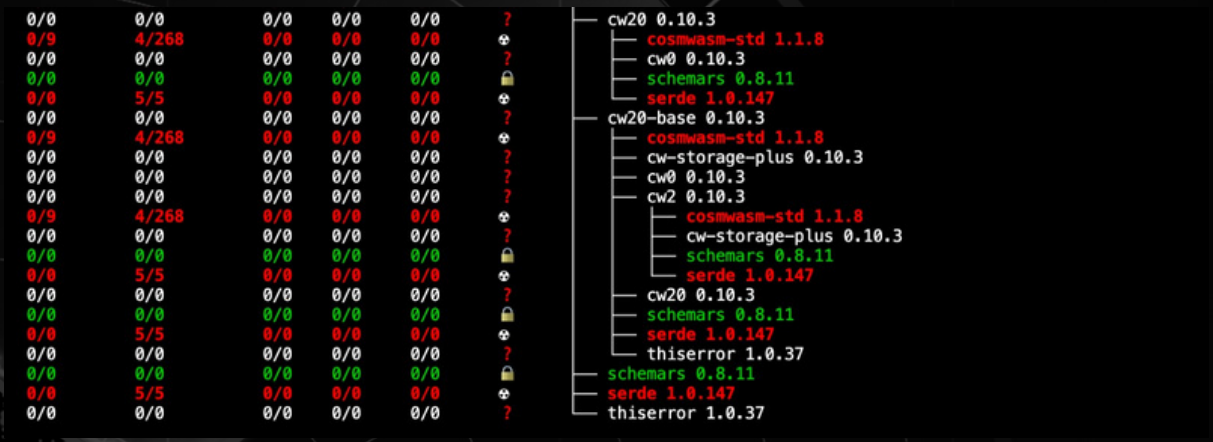


0/0	2/2	0/0	0/0	0/0	☢
0/0	0/0	0/0	0/0	0/0	🔒
8/8	196/196	0/0	0/0	0/0	☢
0/0	2/2	0/0	0/0	0/0	☢
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	?
0/0	15/15	0/0	0/0	3/3	☢
0/0	0/0	0/0	0/0	0/0	?
0/0	69/69	3/3	0/0	2/2	☢
0/0	0/0	0/0	0/0	0/0	?
0/0	69/69	3/3	0/0	2/2	☢
0/0	0/0	0/0	0/0	0/0	?
0/0	15/15	0/0	0/0	3/3	☢
0/0	0/0	0/0	0/0	0/0	?
0/0	69/69	3/3	0/0	2/2	☢
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	🔒
0/24	0/744	0/13	0/1	0/20	?
0/0	5/5	0/0	0/0	0/0	☢
0/0	17/17	0/0	0/0	0/0	☢
0/0	0/14	0/0	0/0	0/0	?
0/0	5/5	0/0	0/0	0/0	☢
0/0	0/0	0/0	0/0	0/0	🔒
0/0	15/15	0/0	0/0	3/3	☢
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	?
0/0	15/15	0/0	0/0	3/3	☢
0/0	0/0	0/0	0/0	0/0	?
0/0	69/69	3/3	0/0	2/2	☢
0/0	69/69	3/3	0/0	2/2	☢
0/0	5/5	0/0	0/0	0/0	☢
0/0	4/7	0/0	0/0	0/0	☢
0/0	5/5	0/0	0/0	0/0	☢
0/0	2/2	0/0	0/0	0/0	☢
0/0	5/5	0/0	0/0	0/0	☢
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	?
1/1	193/193	0/0	0/0	0/0	☢
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	?
0/9	4/268	0/0	0/0	0/0	☢
0/0	0/0	0/0	0/0	0/0	🔒
0/0	5/5	0/0	0/0	0/0	☢
0/0	0/0	0/0	0/0	0/0	?
0/9	4/268	0/0	0/0	0/0	☢
0/0	0/0	0/0	0/0	0/0	🔒
0/0	5/5	0/0	0/0	0/0	☢
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	?
0/9	4/268	0/0	0/0	0/0	☢
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	🔒
0/0	5/5	0/0	0/0	0/0	☢
0/0	0/0	0/0	0/0	0/0	?
0/9	4/268	0/0	0/0	0/0	☢
0/0	0/0	0/0	0/0	0/0	?
0/9	4/268	0/0	0/0	0/0	☢
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	🔒
7/7	134/134	2/2	0/0	2/2	☢
0/0	5/5	0/0	0/0	0/0	☢

```
│   │   │   │   └── rand_core 0.6.4
│   │   │   ├── elliptic-curve 0.12.3
│   │   │   └── sha2 0.10.6
│   │   ├── rand_core 0.6.4
│   │   └── thiserror 1.0.37
│   │       └── thiserror-impl 1.0.37
│   │           ├── proc-macro2 1.0.47
│   │           ├── quote 1.0.21
│   │           └── syn 1.0.103
│   ├── cosmwasm-derive 1.1.8
│   │   └── syn 1.0.103
│   ├── derivative 2.2.0
│   │   ├── proc-macro2 1.0.47
│   │   ├── quote 1.0.21
│   │   └── syn 1.0.103
│   ├── forward_ref 1.0.0
│   ├── hex 0.4.3
│   ├── schemars 0.8.11
│   │   ├── bytes 1.3.0
│   │   │   └── serde 1.0.147
│   │   ├── dyn-clone 1.0.9
│   │   ├── either 1.8.0
│   │   │   └── serde 1.0.147
│   │   ├── schemars_derive 0.8.11
│   │   │   ├── proc-macro2 1.0.47
│   │   │   ├── quote 1.0.21
│   │   │   ├── serde_derive_internals 0.26.0
│   │   │   │   ├── proc-macro2 1.0.47
│   │   │   │   ├── quote 1.0.21
│   │   │   │   └── syn 1.0.103
│   │   │   └── syn 1.0.103
│   │   ├── serde 1.0.147
│   │   └── serde_json 1.0.89
│   ├── serde 1.0.147
│   ├── serde-json-wasm 0.4.1
│   │   └── serde 1.0.147
│   ├── thiserror 1.0.37
│   └── uint 0.9.4
│       ├── byteorder 1.4.3
│       ├── crunchy 0.2.2
│       ├── hex 0.4.3
│       └── static_assertions 1.1.0
├── cw-storage-plus 0.14.0
│   ├── cosmwasm-std 1.1.8
│   ├── schemars 0.8.11
│   └── serde 1.0.147
├── cw0 0.10.3
│   ├── cosmwasm-std 1.1.8
│   ├── schemars 0.8.11
│   ├── serde 1.0.147
│   └── thiserror 1.0.37
├── cw2 0.14.0
│   ├── cosmwasm-std 1.1.8
│   ├── cw-storage-plus 0.14.0
│   ├── schemars 0.8.11
│   └── serde 1.0.147
├── cw20 0.14.0
│   ├── cosmwasm-std 1.1.8
│   ├── cw-utils 0.14.0
│   │   ├── cosmwasm-std 1.1.8
│   │   ├── cw2 0.14.0
│   │   ├── schemars 0.8.11
│   │   ├── semver 1.0.14
│   │   │   └── serde 1.0.147
```

Analysis

Unsafe Code Usage



Commment:

Code that uses the unsafe keyword is considered unsafe since all of the memory safety guarantees of Rust are not enforced there. It means that the code might be prone to vulnerabilities that would've been prevented by the compiler such as Buffer overflow, Double free, Use After free, and more.The results show that many core components may contain unsafe Rust code. We recommended to always double check unsafe code in your own codebase and monitor any core dependencies that contain unsafe Rust in case of any found vulnerabilities.

Status: Mitigated

Analysis

Test Coverage (Low)

Issue/Title:

Test Coverage (Low)

Severity:

Lowest/Optimization/Informational

Description:

Code coverage by unit testing or functional testing is a good practice to be sure all lines of the code work correctly.

Analysis

Test Coverage (Low)

Results:

Running 15 tests

```
test contract::tests::test_buying_of_club ... ok
test contract::tests::test_multiple_buying_of_club ... ok
test contract::tests::test_buying_of_club_after_releasing_by_prev_owner ... ok
test contract::tests::test_assign_a_club ... ok
test contract::tests::test_assign_stakes_to_a_club ... ok
test contract::tests::test_claim_rewards_with_no_auto_stake ... ok
test contract::tests::test_multiple_staking_on_club_by_same_address ... ok
test contract::tests::test_immediate_partial_withdrawals_from_club ... ok
test contract::tests::test_owner_claim_rewards ... ok
test contract::tests::test_claim_previous_owner_rewards ... ok
test contract::tests::test_immediate_complete_withdrawals_from_club ... ok
test contract::tests::test_non_immediate_complete_withdrawals_from_club_with_scheduled_refunds ... ok
test contract::tests::test_non_immediate_complete_withdrawals_from_club ... ok
test contract::tests::test_non_immediate_partial_withdrawals_from_club ... ok
test contract::tests::test_distribute_rewards ... ok
```

ok. 15 passed; 0 failed.

59.86% Coverage, 780/1303 lines covered.

Analysis

Test Coverage (Low)

Results:

Running 41 tests

```
test contract::tests::instantiate::marketing::invalid_marketing ... ok
test contract::tests::instantiate::marketing::basic ... ok
test contract::tests::instantiate::basic ... ok
test allowances::tests::no_self_allowance ... ok
test contract::tests::burn ... ok
test contract::tests::instantiate::mintable ... ok
test contract::tests::instantiate::mintable_over_cap ... ok
test contract::tests::can_mint_by_minter ... ok
test contract::tests::marketing::clear_description ... ok
test contract::tests::marketing::clear_marketing ... ok
test contract::tests::marketing::clear_project ... ok
test contract::tests::marketing::update_description ... ok
test contract::tests::instantiate_multiple_accounts ... ok
test contract::tests::marketing::update_logo_png_invalid ... ok
test contract::tests::marketing::update_logo_png ... ok
test contract::tests::marketing::update_logo_png_oversized ... ok
test allowances::tests::burn_from_respects_limits ... ok
test contract::tests::marketing::update_logo_svg_invalid ... ok
test contract::tests::marketing::update_logo_svg ... ok
test contract::tests::marketing::update_logo_url ... ok
test contract::tests::marketing::update_logo_svg_oversized ... ok
test contract::tests::marketing::update_project ... ok
test allowances::tests::transfer_from_respects_limits ... ok
test contract::tests::marketing::update_marketing ... ok
test allowances::tests::send_from_respects_limits ... ok
test allowances::tests::increase_decrease_allowances ... ok
test contract::tests::marketing::update_marketing_invalid ... ok
test contract::tests::marketing::update_unauthorised ... ok
test contract::tests::no_one_mints_if_minter_unset ... ok
test contract::tests::minter_can_update_minter_but_not_cap ... ok
test contract::tests::others_cannot_mint ... ok
test contract::tests::others_cannot_update_minter ... ok
test contract::tests::queries_work ... ok
test allowances::tests::allowances_independent ... ok
test contract::tests::unset_minter ... ok
test enumerable::tests::query_all_accounts_works ... ok
test contract::tests::send ... ok
test contract::tests::transfer ... ok
test enumerable::tests::query_all_owner_allowances_works ... ok
test enumerable::tests::query_all_spender_allowances_works ... ok
test contract::tests::migration::test_migrate ... ok
```

ok. 41 passed; 0 failed.

88.79% coverage, 483/544 lines covered.

Analysis

Test Coverage (Low)

Commment:

We recommend to increase the code coverage as much possible in order to increase the possible tests to check all the functionalities. This will help the production release functions as intended.

Status: Mitigated

Analysis

Undefined Behaviour

Issue/Title:

Undefined Behaviour

Description:

Detection of different classes of undefined behaviour with an experimental interpreter for Rust's mid-level intermediate representation.

Analysis

Undefined Behaviour

Results:

Running 15 tests

```
test contract::tests::test_assign_a_club ... ok
test contract::tests::test_assign_stakes_to_a_club ... ok
test contract::tests::test_buying_of_club ... ok
test contract::tests::test_buying_of_club_after_releasing_by_prev_owner ... ok
test contract::tests::test_claim_previous_owner_rewards ... ok
test contract::tests::test_claim_rewards_with_no_auto_stake ... ok
test contract::tests::test_distribute_rewards ... ok
test contract::tests::test_immediate_complete_withdrawals_from_club ... ok
test contract::tests::test_immediate_partial_withdrawals_from_club ... ok
test contract::tests::test_multiple_buying_of_club ... ok
test contract::tests::test_multiple_staking_on_club_by_same_address ... ok
test contract::tests::test_non_immediate_complete_withdrawals_from_club ... ok
test contract::tests::test_non_immediate_complete_withdrawals_from_club_with_scheduled_refunds ... ok
test contract::tests::test_non_immediate_partial_withdrawals_from_club ... ok
test contract::tests::test_owner_claim_rewards ... ok
```

ok. 15 passed; 0 failed.

Analysis

Undefined Behaviour

Results:

Running 41 tests

```
test allowances::tests::allowances_independent ... ok
test allowances::tests::burn_from_respects_limits ... ok
test allowances::tests::increase_decrease_allowances ... ok
test allowances::tests::no_self_allowance ... ok
test allowances::tests::send_from_respects_limits ... ok
test allowances::tests::transfer_from_respects_limits ... ok
test contract::tests::burn ... ok
test contract::tests::can_mint_by_minter ... ok
test contract::tests::instantiate::basic ... ok
test contract::tests::instantiate::marketing::basic ... ok
test contract::tests::instantiate::marketing::invalid_marketing ... ok
test contract::tests::instantiate::mintable ... ok
test contract::tests::instantiate::mintable_over_cap ... ok
test contract::tests::instantiate_multiple_accounts ... ok
test contract::tests::marketing::clear_description ... ok
test contract::tests::marketing::clear_marketing ... ok
test contract::tests::marketing::clear_project ... ok
test contract::tests::marketing::update_description ... ok
test contract::tests::marketing::update_logo_png ... ok
test contract::tests::marketing::update_logo_png_invalid ... ok
test contract::tests::marketing::update_logo_png_oversized ... ok
test contract::tests::marketing::update_logo_svg ... ok
test contract::tests::marketing::update_logo_svg_invalid ... ok
test contract::tests::marketing::update_logo_svg_oversized ... ok
test contract::tests::marketing::update_logo_url ... ok
test contract::tests::marketing::update_marketing ... ok
test contract::tests::marketing::update_marketing_invalid ... ok
test contract::tests::marketing::update_project ... ok
test contract::tests::marketing::update_unauthorised ... ok
test contract::tests::migration::test_migrate ... ok
test contract::tests::minter_can_update_minter_but_not_cap ... ok
test contract::tests::no_one_mints_if_minter_unset ... ok
test contract::tests::others_cannot_mint ... ok
test contract::tests::others_cannot_update_minter ... ok
test contract::tests::queries_work ... ok
test contract::tests::send ... ok
test contract::tests::transfer ... ok
test contract::tests::unset_minter ... ok
test enumerable::tests::query_all_accounts_works ... ok
test enumerable::tests::query_all_owner_allowances_works ... ok
test enumerable::tests::query_all_spender_allowances_works ... ok
```

ok. 41 passed; 0 failed.

Analysis

Undefined Behaviour

Commment:

All predefined tests were passed without any issues raised.

Tests to detect memory leaks were conducted as well as certain classes of undefined behavior, such as:

- Out-of-bounds memory accesses and use-after-free
- Invalid use of uninitialized data
- Violation of intrinsic preconditions
- Not sufficiently aligned memory accesses and references
- Violation of some basic type invariants (a bool that is not 0 or 1, for example, or an invalid enum discriminant)
- Experimental: Violations of the Stacked Borrows rules governing aliasing for reference types
- Experimental: Data races (but no weak memory effects)

Status: Mitigated

Analysis

Fuzzing

Issue/Title:

Fuzzing

Description:

As unsafe code was detected in the usage of some core dependencies, Saulidity conducted fuzzing to some of those dependencies for a certain period of time.

```
#![no_main]

use libfuzzer_sys::fuzz_target;
use serde_json::*;

fn json_fuzz(data: &str) -> serde_json::Result<()> {
    let v: serde_json::Value = serde_json::from_str(data)?;

    ok(())
}
fuzz_target!(|data: &[u8]| {
    // fuzzed code goes here
});
```

Analysis

Fuzzing

```rust
#[macro_use]
extern crate honggfuzz;
extern crate anyhow;

use anyhow::{ Result, anyhow};

fn err (data: &str) -> Result<()> {
    anyhow::ensure!(data.len() > 0, "only user {} is allowed", data);

    return Err(anyhow!("some error {:?}", data));

}

▶ Run | Debug
fn main() -> Result<()>{
    println!("Starting fuzzer");

    loop {

        fuzz!(|data: &str| {

            err(data);

        });

    }
}
```

Analysis

Fuzzing

Commment:

Due to compiler error, only handpicked dependencies were fuzzed for a certain amount of time. Saulidity recommends fixing the compiler error. All fuzzing tests were positive ie. no issues were detected at this time.

Status: Mitigated

Testing Standards

The goal of the audit was to find any potential smart contract security problems and vulnerabilities.
The information in this report should be used to understand the smart contract's risk exposure and as a guide to improving the smart contract's security posture by addressing the concerns that were discovered.

The blockchain platform is used to deploy and execute smart contracts. The platform, its programming language, and other smart contract-related applications all have vulnerabilities that may be exploited. As a result, the audit cannot ensure the audited smart contract(s) explicit security.Audits can't make statements or warranties on security of the code.It also cannot be deemed an adequate assessment of the code's utility and safety, bug-free status, or any statements of the smart contract.While we did our best in completing the study and publishing this report, it is crucial to emphasize that you should not rely only on it; we advocate all projects doing many independent audits and participating in a public bug bounty program to assure smart contract security.

Testing Standards

1. Gather all relevant data.

2. Perform a preliminary visual examination of all documents and contracts.

3. Find security holes with specialist tools & manual review with independent experts.

4. Create and distribute a report.

SAULIDITY

Smart Contract
Audit

saulidity.com
Saulidity
@Saulidity