// HALBORN

ORAICHAIN-COSMWASM

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **February 12, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	02/12/2021	Gabi Urrutia
0.2	Document Edits	02/15/2021	Gabi Urrutia
1.0	Final Version	02/16/2021	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

EXECUTIVE SUMMARY

1.1 INTRODUCTION

Oraichain engaged Halborn to conduct a security assessment on smart contracts for the Cosmos ecosystem beginning on February 12th, 2021 and ending February 16th, 2021. The security assessment was scoped to the contract built by CosmWasm in Rust language and an audit of the security risk and implications regarding the changes introduced by the development team at Oraichain prior to its production release shortly following the assessments deadline.

Overall, the smart contracts code does NOT contain any obvious exploitation vectors that Halborn was able to leverage within the timeframe of testing allotted. In addition, Contracts are written in Rust by CosmWasm as a module that can plug into the Cosmos SDK.
Halborn recommends performing further testing to validate extended safety and correctness in context to the whole set of contracts. External threats, such as economic attacks, oracle attacks, and inter-contract functions and calls should be validated for expected logic and state.

1.2 TEST APPROACH & METHODOLOGY

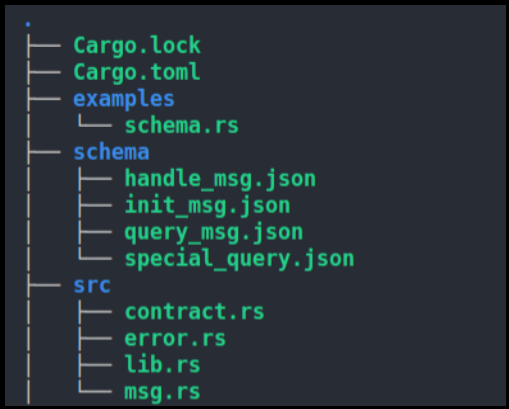
Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the smart contract audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts 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 smart contracts.
- Smart Contract manual code read and walkthrough.
- Manual Assessment of use and safety for the critical Rust variables and functions in scope to identify any arithmetic related vulnerability classes.
- Scanning of Rust files for vulnerabilities. (cargo-audit)

- Common mistakes and improvement Rust code (cargo-clippy)

1.3 SCOPE

IN-SCOPE:
Code into datasource-eth folder:



Specific commit of contract: commit

0b5e0da4996c8ef36f01fe317e48d54ec879ba92

OUT-OF-SCOPE:
Other smart contracts in the repository and economics attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	3

EXECUTIVE SUMMARY

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
DOCUMENTATION	Informational	-
VULNERABILITY SCANNING	Informational	-
CODE IMPROVEMENTS	Informational	-

FINDINGS & TECH DETAILS

3.1 DOCUMENTATION - INFORMATIONAL

Description:

The documentation to set up the environment and interact with it requires many libraries and packages not included there. Furthermore, for simulating the smart contracts interactions by cosmwasm-simulate tool, the documentation was provided by Oraichain team but not included in the repository.

Recommendation:

Consider updating the documentation.

3.2 VULNERABILITY SCANNING - INFORMATIONAL

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. Security Detections are only in scope.

Reference: https://rustsec.org/advisories/

Results:

```
ziion@eltitourruts-virtual-machine:~/ORAI_COSMOS/orai-master/smart-contracts/datasource-eth$ cargo audit
    Fetching advisory database from `https://github.com/RustSec/advisory-db.git`
      Loaded 245 security advisories (from /home/ziion/.cargo/advisory-db)
    Updating crates.io index
    Scanning Cargo.lock for vulnerabilities (32 crate dependencies)
```

No vulnerabilities were founded.

3.3 CODE IMPROVEMENTS - INFORMATIONAL

Description:

Halborn used cargo-clippy code analysis tool to detect common mistakes and possible improvements in Rust code. More than 400 recommendations are stored in rust-lang.

Reference: https://rust-lang.github.io/rust-clippy/master/index.html

Results:

```
ziion@eltitourruts-virtual-machine:~/ORAI_COSMOS/orai-master/smart-contracts/datasource-eth$ cargo clippy
warning: single-character string constant used as pattern
  --> src/contract.rs:54:56
   |
54 |     let last = first + data.get(first..).unwrap().find("}").unwrap();
   |                                                        ^^^ help: try using a `char` instead: `'}'`
   |
   = note: `#[warn(clippy::single_char_pattern)]` on by default
   = help: for further information visit https://rust-lang.github.io/rust-clippy/master/index.html#single_char_pattern

warning: field assignment outside of initializer for an instance created with Default::default()
  --> src/msg.rs:18:59
   |
18 | #[derive(Serialize, Deserialize, Clone, Debug, PartialEq, JsonSchema)]
   |                                                           ^^^^^^^^^^
   |
   = note: `#[warn(clippy::field_reassign_with_default)]` on by default
note: consider initializing the variable with `schemars::schema::Metadata { description: JsonSchema, ..Default::default() }` and removing relevant reassignments
  --> src/msg.rs:18:59
   |
18 | #[derive(Serialize, Deserialize, Clone, Debug, PartialEq, JsonSchema)]
   |                                                           ^^^^^^^^^^
   = help: for further information visit https://rust-lang.github.io/rust-clippy/master/index.html#field_reassign_with_default
   = note: this warning originates in a derive macro (in Nightly builds, run with -Z macro-backtrace for more info)

warning: 2 warnings emitted

    Finished dev [unoptimized + debuginfo] target(s) in 0.01s
```

Recommendation:

It’s faster using a char than using a str for string methods that receive a single-character str as an argument. So, it’s better to use ‘x’ instead of “x” because multi-byte unicode characters could be not cached. Otherwise, it’s widely accepted to use `#[derive(Serialize, Deserialize, Clone, Debug, PartialEq, JsonSchema)]` for initializing variables.

THANK YOU FOR CHOOSING

// HALBORN