Holoride - MultiverseX Bridge
Security Assessment Findings Report

Date: September 9, 2023
Project: HMB
Version 1.1

Contents

1 Confidentiality statement **3**

2 Disclaimer **3**

3 About Sub7 **4**

4 Project Overview **4**

5 Executive Summary **5**

5.1 Scope 5
5.2 Timeline 5
5.3 Summary of Findings Identified 6
5.4 Methodology 8

6 Findings and Risk Analysis **9**

6.1 High level of centralization on both sides of the bridge 9
6.2 addWrappedToken could be misused 10
6.3 Lack of good-practice init function implementations 11
6.4 Lack of variable update functionality 11
6.5 Possibility of maliciously increasing service fees 12
6.6 depositLiquidity is misleading and could lead to lose funds by the user 13
6.7 Lack of tests and documentation 14
6.8 Manual token assertions 14
6.9 Lack of safe math operations and overflow-checks enabled 15
6.10 The service fee calculation returns zero for <200 tokens 16
6.11 _validateAddress function is not neccesary 16
6.12 TODO in the codebase 17
6.13 State variable is never used 18
6.14 Outdated dependencies 18

1 Confidentiality statement

2 Disclaimer

A smart contract security audit is considered a snapshot in time. The findings and recommendations reflect the information gathered during the assessment and not any changes or modifications made outside of that period.

Time-limited engagements do not allow for a full evaluation of all security controls. Sub7 Security prioritized the assessment to identify the weakest security controls an attacker would exploit. Sub7 Security recommends conducting similar assessments on an annual basis by internal or third-party assessors to ensure the continued success of the controls

3 About Sub7

Sub7 is a Web3 Security Agency, offering Smart Contract Auditing Services for blockchain-based projects in the DeFi, Web3 and Metaverse space.

Learn more about us at https://sub7.xyz

4 Project Overview

Holoride is turning vehicles into moving theme parks while revolutionizing in-car entertainment for you. Transit time will never be the same again. Buckle up, put on your VR headset and explore games and experiences with the holoride app, motion-synced with the car.

RIDE is the utility token of the holoride platform that sits at the heart of its ecosystem, built on the MultiversX Network. RIDE is designed to supercharge the connection between holoride users, creators, and business partners by providing additional benefits and enhanced user engagement.

RIDE is being built on top of the proprietary core technology provided by holoride GmbH. The RIDE token is issued by holoride AG, Liechtenstein.

Holoride intends to expand access to the RIDE token as well as increase it's holder base by deploying a cross-chain bridge between MultiversX and Ethereum for the RIDE token. The open-source bridge built by the MultiversX team was used as the basis for the RIDE bridge. The bridge was developed in conjunction with Solidant Ltd.

5 Executive Summary

Sub7 Security has been engaged to what is formally referred to as a Security Audit of Solidity Smart Contracts, a combination of automated and manual assessments in search for vulnerabilities, bugs, unintended outputs, among others inside deployed Smart Contracts.

The client has chosen for a Light service audit

Our Light Audit service is a cost-effective alternative to our comprehensive Full Audit service, created especially for clients with tight budget constraints. This service employs a single expert auditor who carries out a meticulous review of your Solidity code to identify vulnerabilities, bugs, errors, and manipulation issues. The process culminates in a thorough report outlining all potential risks and providing clear recommendations for improvements.

1 (One) Security Researcher/Consultant was engaged in this activity

5.1 Scope

https://github.com/solidant/multiverseX/tree/feat/tech-docs

Commit Hash: 79ca320ea4a4d4f5ce037dd0a00046dd8978b613

Multiversx side:

- rs/bridged-tokens-wrapper/src/*
- rs/common/fee-estimator-module/src/*
- rs/esdt-safe/src/*
- rs/multi-transfer-esdt/src/*
- rs/multisig/src/*
- rs/price-aggregator/src/*
- rs/wesdt-swap/src/*
- deposit-wrapper/src/lib.rs

Ethereum Side:

- sol/contracts/Bridge.sol
- sol/contracts/ERC20Safe.sol
- sol/contracts/GenericERC20.sol
- sol/contracts/SharedStructs.sol

5.2 Timeline

24 July 2023 to 11 August 2023

5.3 Summary of Findings Identified

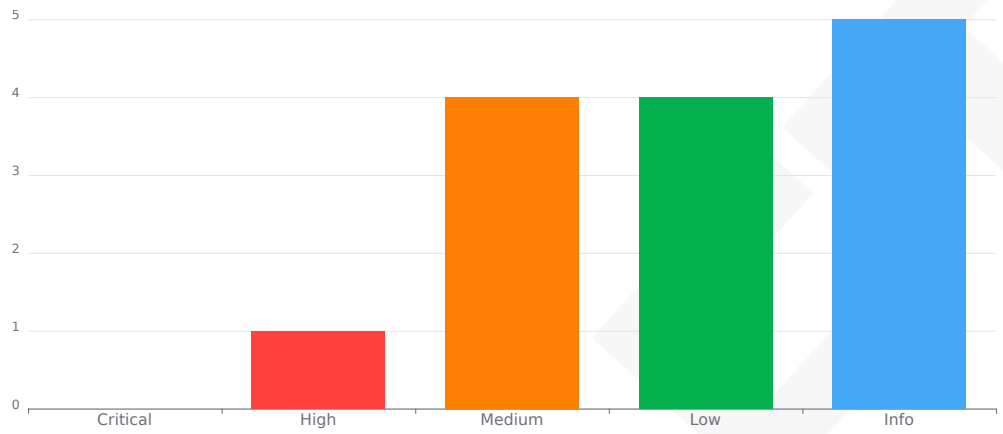

Figure 1: Executive Summary

 # 1 High High level of centralization on both sides of the bridge – ***Acknowledged***

 # 2 Medium addWrappedToken could be misused – ***Fixed***

 # 3 Medium Lack of good-practice init function implementations – ***Fixed***

 # 4 Medium Lack of variable update functionality – ***Acknowledged***

 # 5 Medium Possibility of maliciously increasing service fees – ***Acknowledged***

 # 6 Low depositLiquidity is misleading and could lead to lose funds by the user – ***Fixed***

 # 7 Low Lack of tests and documentation – ***Acknowledged***

 # 8 Low Manual token assertions – ***Acknowledged***

 # 9 Low Lack of safe math operations and overflow-checks enabled – ***Fixed***

 # 10 Informational The service fee calculation returns zero for <200 tokens – ***Acknowledged***

 # 11 Informational _validateAddress function is not neccesary – ***Fixed***

 # 12 Informational TODO in the codebase – ***Fixed***

 # 13 Informational State variable is never used – ***Fixed***

 # 14 Informational Outdated dependencies – ***Fixed***

5.4 Methodology

Sub7 Security follows a phased assessment approach that includes thorough application profiling, threat analysis, dynamic and manual testing. Security Auditors/Consultants utilize automated security tools, custom scripts, simulation apps, manual testing and validation techniques to scan for, enumerate and uncover findings with potential security risks.

As part of the process of reviewing solidity code, each contract is checked against lists of known smart contract vulnerabilities, which is crafted from various sources like SWC Registry , DeFi threat and previous audit reports.

The assessment included (but was not limited to) reviews on the following attack vectors:

Oracle Attacks | Flash Loan Attacks | Governance Attacks | Access Control Checks on Critical Function | Account Existence Check for low level calls | Arithmetic Over/Under Flows | Assert Violation | Authorization through tx.origin | Bad Source of Randomness | Block Timestamp manipulation | Bypass Contract Size Check | Code With No Effects | Delegatecall | Delegatecall to Untrusted Callee | DoS with (Unexpected) revert | DoS with Block Gas Limit | Logical Issues | Entropy Illusion | Function Selector Abuse | Floating Point and Numerical | Precision | Floating Pragma | Forcibly Sending Ether to a Contract | Function Default Visibility | Hash Collisions With Multiple Variable Length Arguments | Improper Array Deletion | Incorrect interface | Insufficient gas griefing | Unsafe Ownership Transfer | Loop through long arrays | Message call with hardcoded gas amount | Outdated Compiler Version | Precision Loss in Calculations | Price Manipulation | Hiding Malicious Code with External Contract | Public burn() function | Race Conditions / Front Running | Re-entrancy | Requirement Violation | Right-To-Left-Override control character (U+202E) | Shadowing State Variables | Short Address/Parameter Attack | Signature Malleability | Signature Replay Attacks | State Variable Default Visibility | Transaction Order Dependence | Typographical Error | Unchecked Call Return Value | Unencrypted Private Data On-Chain | Unexpected Ether balance | Uninitialized Storage Pointer | Unprotected Ether Withdrawal | Unprotected SELFDESTRUCT Instruction | Unprotected Upgrades | Unused Variable | Use of Deprecated Solidity Functions | Write to Arbitrary Storage Location | Wrong inheritance | Many more…

6 Findings and Risk Analysis

6.1 High level of centralization on both sides of the bridge

Severity: High
Status: Acknowledged

Description

Almost all contracts appears to suffer from a significant level of centralization. Namely, the owner is able to perform very wide range of actions that impacts other users. The most significant one are being able to:

- Slash the stake of other user
- Reduce the quorum to the minimal value (1) and run a validator
- Freeze the specific token and then withdraw whole balance of it from the contract

Having in mind slashing for example, although `slash\_board\_member` function's comment states that this function is supposed to be executed only if other user acts maliciously, there is nothing in the code that would prevent the execution of this functions anytime the owner pleases. This leads to funds losses, if the owner decides to become malicious.

Location

Entire codebase - mostly functions marked as `#\[only\_owner\]` on MultiverseX side, and as {{onlyAdmin}}on Ethereum side.

Samples:

- `setQuorum`
- `setServiceFeePercentage`
- `setMaxServiceFee`
- `setServiceFeeReceiver`
- `recoverLostFunds`
- `slash\_board\_member`
- `change\_quorum`

Recommendation

We suggest that an in-depth analysis of the risks arising from the use of such a high level of centralization in the protocol is performed. The following issues should be taken into account:

- How trustworthy and impartial will the people who have control over the Owner/Administrator wallet be?
- Will Multisig offer the appropriate level of control over particularly critical functions? How many people should be included in its quorum?
- What will be the approach of the community and potential investors and participants using the bridge? Will this level of centralization in the context of the general pro-decentralization trend not cause dislike for the protocol?
- How likely is it that an action will be performed incorrectly, which will disrupt the operation of the protocol and expose the protocol to financial/reputational losses?

Client Comments

None

6.2 addWrappedToken could be misused

Severity: Medium
Status: Fixed

Description

It was found that the `add\_wrapped\_token` function, ultimately used to add new tokens in the `wrapped` version, i.e. minted in place of those received from the opposite side of the bridge, does not verify whether a token with the same `TokenIdentifier` already exists. As a consequence, it is possible to overwrite the token with another `num\_decimals` value. This may have negative consequences for the protocol if some of the tokens are already minted and they have a different decimals value than after `update`. State may then be inconsistent.

The same applies to the `updateWrappedToken` endpoint - when using it, make sure that it will not have any negative consequences for the protocol.

Location

- rs/bridged-tokens-wrapper/src/lib.rs:19-27

Recommendation

We suggest validating whether the `universal\_bridged\_token\_ids` being added already exists, for example by adding:

```
require!(
  !self.universal\_bridged\_token\_ids()
    .contains(&universal\_bridged\_token\_ids),
  "Universal token was already added"
);
```

Client Comments

None

6.3 Lack of good-practice init function implementations

Severity: Medium
Status: Fixed

Description

It was observed that `#\[init\]` functions are using `.set()` functions to set the storage variables. However, it is considered a good practice to use `.set\_if\_empty()` functions that will only change those values if they do not exist. This is to make sure that any contract upgrades do not break the functionality.

It is worth to point out, that usage of `.set\_if\_empty()` method is also mentioned in the official MultiverseX documentation - https://docs.multiversx.com/developers/developer-reference/upgrading-smart-contracts/

Location

- Global issue in almost all contracts on `Rust` side

Recommendation

It is recommended to change the `.set()` calls to `.set\_if\_empty()` calls in initialization functions.

Client Comments

None

6.4 Lack of variable update functionality

Severity: Medium
Status: Acknowledged

Description

It was observed that the `depost-wrapper` contract does not implement setters for its storage variables. Meaning that updating the addresses of specific contracts would be possible only by upgrading the contract itself or deploying a new instance.

Location

- `deposit-wrapper` contract's logic

Recommendation

It is recommended to implement a setter functions that would allow to update the storage variables. Additionally, such functions should implement an Access Control mechanism that would prevent an anonymous change of said variables.

Client Comments

None

6.5 Possibility of maliciously increasing service fees

Severity: Medium
Status: Acknowledged

Description

It was observed that the `FeeEstimatorModule` defines both `service\_fee\_percentage` and `max\_service\_fee` as variables controlled by owner. This means that a malicious owner with access to mempool could execute a sandwich attack. I.e. the malicious owner would keep the fees at a low value and sandwich the discovered transaction with a fee increase and then a decrease to a low value. Such behaviour would maliciously increase own fees while looking innocent to users. The same situation applies to the Ethereum side on the bridge, with `setServiceFeePercentage` and `setMaxServiceFee` functions.

Location

- sol/contracts/ERC20Safe.sol:90-93, `setServiceFeePercentage()`
- sol/contracts/ERC20Safe.sol:95-97, `setMaxServiceFee()`
- rs/common/fee-estimator-module/src/lib.rs:30-34, `set\_service\_fee\_percentage()`
- rs/common/fee-estimator-module/src/lib.rs:36-40, `set\_max\_service\_fee()`

Recommendation

It is recommended to change a `max\_service\_fee` and `maxServiceFee` from a variables to a constants values. Furthermore, a "gracing period" for changing the `service\_fee\_percentage` and `serviceFeePercentage` should be considered. This way, a change in the service fee would impact users with a slight delay making it impossible to surprise users with maliciously increased fees without notice. Additionally, some responsible min/max bounds should be considered to introduce one more prevention mechanism from potentially dangerous changes of fees.

Client Comments

None

6.6 depositLiquidity is misleading and could lead to lose funds by the user

Severity: Low
Status: Fixed

Description

It was found that the `depositLiquidity` entry-point allows any user to deposit liquidity for any token. The problem is that due to the fact that this function is not documented anywhere, its name may suggest the existence of a mechanism for depositing liquidity, earning interest, and then withdrawing liquidity.

Unfortunately, if the user makes a mistake and uses this function with tokens included, he will lose control over them, and the protocol itself will not even record who sent them. No withdraw option is also possible.

This is quite problematic and adds unnecessary risk to the protocol.

Location

- rs/bridged-token-wrapper/src/lib.rs:127-133

Recommendation

The suggested solution is to change the name of the function, document it or mark the endpoint with `#\[only\_owner\]` macro, if the business logic provides that only the owner will deposit initial liquidity to the newly added token.

Client Comments

None

6.7 Lack of tests and documentation

Severity: Low
Status: Acknowledged

Description

It was observed that tests and documentation could be improved upon. Given that the protocol in scope of the engagement is a bridge, which by the very definition will handle a substantial amount and volume of assets, the documentation seems to lack in details. Additionally, the testing suite could be improved to introduce more tests (checking both "happy paths" and error-generating scenarios), using more user-friendly framework than Mandos, for example simple Rust unit tests.

Furthermore, it is worth to mention that no tests are present for the `deposit-wrapper` contract.

Location

- Global issue, affecting multiple contracts

Recommendation

It is recommended to improve the quality of documentation and introduce more tests for increased test coverage.

Client Comments

None

6.8 Manual token assertions

Severity: Low
Status: Acknowledged

Description

It was observed that the `deposit-wrapper` contract is meant to accept transfers from only one token (saved as `esdt\_token\_id` storage variable). The `deposit` function is marked with `payable("\*")` macro and manually asserts if the token sent with the call corresponds to the one saved in storage. Such operation increases the code complexity and price associated with deploying and calling the contract.

The MultiversX technology allows to specify accepted tokens on the protocol level, via the `payable` macro itself.

Location

- rs/deposit-wrapper/src/lib.rs:40, `deposit()`

Recommendation

It is recommended to specify the token in the `payable` macro instead of manually asserting it in the function logic. It will reduce the code complexity and take full advantage of the framework's capabilities.

Client Comments

None

6.9 Lack of safe math operations and overflow-checks enabled

Severity: Low
Status: Fixed

Description

It was observed that arithmetic operations do not use the safe math that is overflow or underflow aware. This, even if made sure that no overflow is possible, is considered a bad practice. Although, for instance in `price-aggregator` the `oracle\_status` is modified in a way that could overflow.

In addition, it has been noticed that in the configuration of Cargo.toml `overflow-checks` for release mode are not `enabled`, which consequently makes the alternative protection disabled.

Location

- rs/price-aggregator/src/lib.rs:83-88, `submit\_unchecked()`
- Global issue on `Rust` side

Recommendation

It is recommended to use safe math operations instead of overflowable default operands. For instance, use `saturating\_add()` or `checked\_add()` functions instead of the default `+`. Additionally, it is recommended to `enable` the `overflow-checks` for the release mode.

Client Comments

None

6.10 The service fee calculation returns zero for <200 tokens

Severity: Informational
Status: Acknowledged

Description

It was noticed that the `calculate\_service\_fee` function, used to calculate and return the service fee to be paid for using the protocol services based on the service fee percentage set in the system, due to the use of simple arithmetic operations, rounds the `service\_fee\_percentage\_amount` to zero for the number of tokens less than 200.
This is because by default `SERVICE\_FEE\_PERCENTAGE` is set to 50, which is 0.5%. In operation, the number of tokens is multiplied by 50 and then divided by 10,000 (100%). Thus, if the first part of the operation returns a value less than 10000, the whole thing rounds up to zero.

The risk related to the vulnerability is negligible, because token decimals usually operate around 15-18, effectively making transfers of such a small amount of tokens pointless, but it is worth keeping this in mind and protecting the function against unplanned side-effects as a standard.

Location

- rs/common/fee-estimator-module/src/lib.rs:79

Recommendation

We suggest adapting the service fee calculation function to this specific scenario in order to correctly calculate the fee for a smaller number of tokens.

Client Comments

None

6.11 _validateAddress function is not neccesary

Severity: Informational
Status: Fixed

Description

The `\_validateAddress` function is used to check whether the `account` parameter passed to it is not a zero address. If it is not, the function is transparent, if it is - an error is returned.

However, it is only used once, within the `\_addRelayer` function. Due to the fact that validation consists in one use of `require` - the existence of a special function for this type of operation is redundant and generates unnecessary costs in storage.

Location

- sol/contracts/access/RelayerRole.sol:72,82-85, `\_validateAddress`

Recommendation

We suggest removing the `\_validateAddress` function and replacing its only use with the line:

```
require(account != address(0), "RelayerRole: account cannot be the 0 address");
```

Client Comments

None

6.12 TODO in the codebase

Severity: Informational
Status: Fixed

Description

Comments marked as `TODO` have been left in several places in the code. This is a common practice during the code development process, but it is nevertheless bad security practice if the code in this version is to be used in production.

They can provide a potential attacker with ideas and attack scenarios on functionalities that have not yet been implemented or require some improvement.

Location

- sol/contracts/access/RelayerRole.sol:47
- rs/multisig/src/setup.rs:31

Recommendation

We recommend removing all `TODO` comments or implementing the solutions described in them.

Client Comments

None

6.13 State variable is never used

Severity: Informational
Status: Fixed

Description

It has been noticed that one of the variables defined in the `Contract state` section of the `Bridge` contract is not used anywhere, edited and has no effect on anything. Its existence is therefore redundant and generates an unnecessary cost of storage.

```
1  string private constant action = "CurrentPendingBatch";
```

Location

- sol/contracts/Bridge.sol:30

Recommendation

We suggest removing the indicated variable, or implementing the contract logic that uses it.

Client Comments

None

6.14 Outdated dependencies

Severity: Informational
Status: Fixed

Description

It was observed that the contracts in scope of the engagement are using dependencies in outdated or not supported versions. Newer versions contain bug fixes, performance optimizations and may contain security fixes.

First example is related to the `elrond-wasm\*` dependencies. Version used is 0.32.0, while 0.38.0 is considered recommended.

Second example is `wee\_alloc` crate with `0.4.5` version, which is marked as unmaintained.

Location

- `Cargo.toml` files in individual contracts

Recommendation

It is recommended to upgrade the dependencies to the latest available version, or replace them with the maintained and secure alternatives.

Client Comments

None