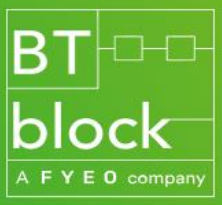

Security Assessment for Vega Multisig Control V2

Vega

June 2022
Version 2.0

Presented by:
BTblock LLC

Corporate Headquarters
FYEO Inc.
PO Box 147044
Lakewood CO 80214
United States

Security Level
Strictly Confidential

Table of Contents

List of Figures

List of Tables

Executive Summary

Overview

Vega engaged Btblock, a FYEO Company, to perform a Security Assessment for Vega Multisig Control Version 2.

The assessment was conducted remotely by the BTblock Security Team. Testing took place on May 05 - May 23, 2022, and focused on the following objectives:

- To provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the results of our tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the BTblock Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

Key Findings

The following issues have been identified during the testing period. These should be prioritized for remediation to reduce to the risk they pose:

- BT-VV2-01 – Signatures may be skipped when counting
- BT-VV2-02 – Missing zero address check
- BT-VV2-03 – Old Solidity compiler
- BT-VV2-04 – An optional limitation with the costly operation
- BT-VV2-05 – Public visibility is set for functions that are not called internally
- BT-VV2-06 – Uncovered test cases

During the test, the following positive observations were noted regarding the scope of the engagement:

- The team was very supportive and open to discuss the design choices made

Based on formal verification we conclude that the reviewed code implements the documented functionality.

Scope and Rules of Engagement

BTblock performed a Security Assessment for Vega Multisig Control Version 2. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied through a public repository at https://github.com/vegaprotocol/MultisigControl with the commit hash e3e3a775264586ed3911432b40ef5fe4a08e3b69. A re-review was performed on June 2, 2022, with the commit hash 89d1f6fdae8df6f11d0f5669a5e78a7836772613.

Files included in the code review
vega-v2/ ├── contracts/ │ ├── ERC20_Asset_Pool.sol │ ├── ERC20_Bridge_Logic_Restricted.sol │ ├── ETH_Asset_Pool.sol │ ├── ETH_Bridge_Logic.sol │ ├── IERC20.sol │ ├── IERC20_Bridge_Logic_Restricted.sol │ ├── IETH_Bridge_Logic.sol │ ├── IMultisigControl.sol │ ├── Migrations.sol │ └── MultisigControl.sol ├── test/ │ ├── Asset_Pool_Unit_Tests.js │ ├── ERC20_Bridge_Logic_Unit_Tests.js │ ├── ETH_Asset_Pool_Unit_Tests.js │ ├── ETH_Bridge_Logic_Tests.js │ └── MultisigControl_Unit_Tests.js └── truffle-config.js

Table 1: Scope

Technical Analyses and Findings

During the Security Assessment for Vega Multisig Control Version 2, we discovered:

- 1 finding with MEDIUM severity rating.
- 2 findings with LOW severity rating.
- 3 findings with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

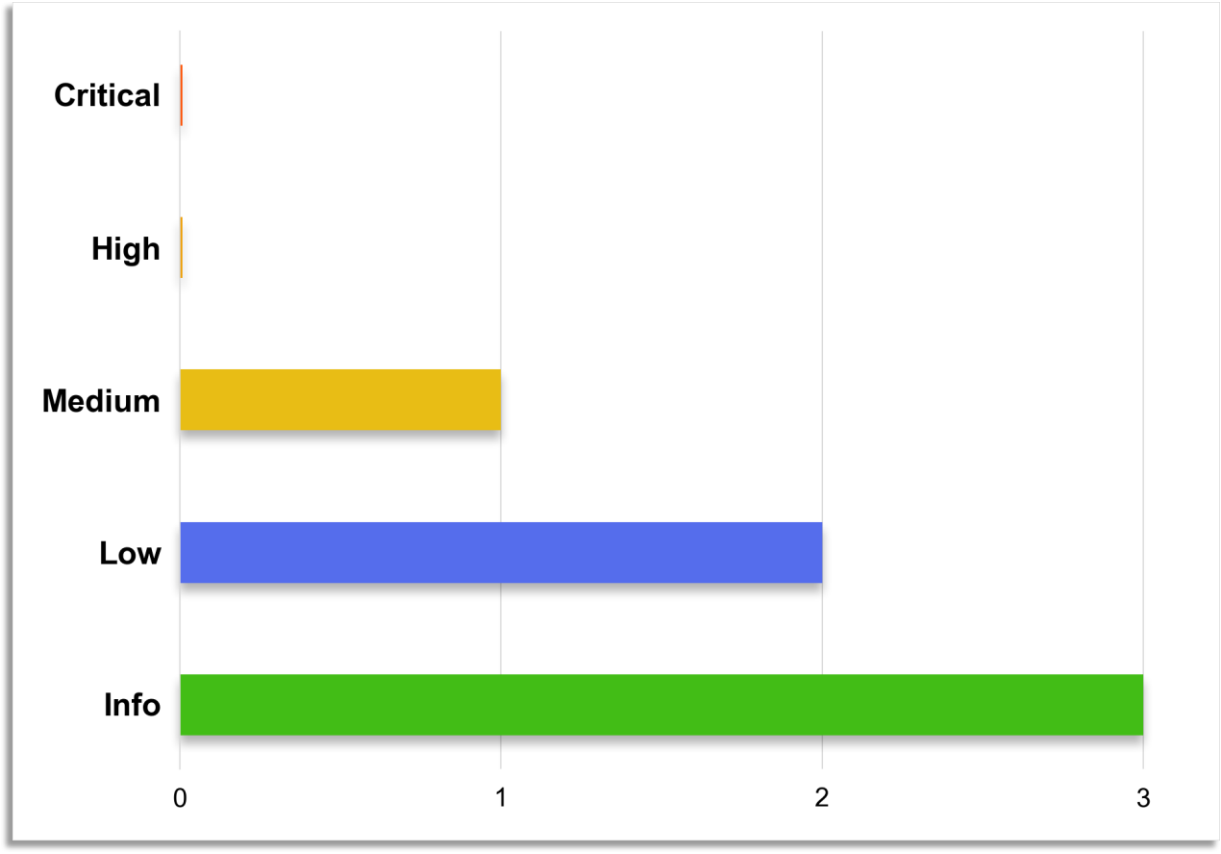

Figure 1: Findings by Severity

Findings

The Findings section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

Finding #	Severity	Description
BT-VV2-01	Medium	Signatures may be skipped when counting
BT-VV2-02	Low	Missing zero address check
BT-VV2-03	Low	Old Solidity compiler
BT-VV2-04	Informational	An optional limitation with the costly operation
BT-VV2-05	Informational	Public visibility is set for functions that are not called internally
BT-VV2-06	Informational	Uncovered test cases

Table 2: Findings Overview

Technical Analysis

The source code has been manually validated to the extent that the state of the repository allowed. The validation includes confirming that the code correctly implements the intended functionality.

Conclusion

Based on formal verification, we conclude that the code implements the documented functionality to the extent of the reviewed code.

Technical Findings

Project Overview & General Observations

The Vega protocol smart contracts are written in Solidity and they implement a bridge between Ethereum and Vega. Any user can deposit a registered Asset or ETH to the smart contract and it will be processed in the Vega Network by a generated event. In order to withdraw, a user needs to have a batch of signatures from the Vega Network that will validate the operation. Smart contracts support multi-signature logic, where in order to validate an operation, the number of signatures should be greater than some threshold.

The overall quality of the code is good. During the audit, no critical issues were found. Among the listed findings there are several recommendations for GAS usage optimization.

The code is fairly covered with unit tests, but several uncovered negative scenarios were mentioned in the findings.

SIGNATURES MAY BE SKIPPED WHEN COUNTING

Finding ID: BT-VV2-01
Severity: Medium
Status: Remediated

Description

If there are not enough signatures for the message on the first call, when performing a second call, signatures from the first one will be ignored.

Proof of Issue

File name: contracts/MultisigControl.sol

Line number: 118

```
if(signers[recovered_address] && !has_signed[message_hash][recovered_address]){
   has_signed[message_hash][recovered_address] = true;
   sig_count++;
}
```

Severity and Impact Summary

- If some of the signatures are skipped, there may not be enough signatures left for verification.
- Usage of storage to save temporary data leads to increased GAS costs.

Recommendation

It is recommended to change the logic that checks for duplicates. For example:

Add a map with a list of IDs assigned to each signer.

```
mapping(address => uint8) signers;
mapping(uint8 => address) ids;
```

IDs should be in the range 1..255.

When adding a signer, check that the ID is not used in the IDs map.

When deleting a signer, set the ID in signers to 0 and the address in IDs to address(0)

In `verify_signatures`, create memory array for used IDs:

```
bool[] memory used = new bool[](256);
```

Check that the signer exists and is not duplicated:

```
uint8 id = signers[recovered_address];
if (id > 0 && !used[id]) {
    used[id] = true;
    sig_count++;
}
```

Because of the additional IDs map, the operation of adding a new signer would be more expensive, but the more frequently used `verify_signatures` function would cost less.

MISSING ZERO ADDRESS CHECK

Finding ID: BT-VV2-02
Severity: Low
Status: Remediated

Description

A zero address Validation is a sanity check.

Proof of Issue

File name: contracts/ERC20_Asset_Pool.sol

Line number: 24

```
multisig_control_address = multisig_control
```

File name: contracts/ERC20_Bridge_Logic_Restricted.sol

Line number: 25

```
erc20_asset_pool_address = erc20_asset_pool
```

File name: contracts/ETH_Asset_Pool.sol

Line number: 60

```
(success) = target.call{value: amount}()
```

File name: contracts/ETH_Bridge_Logic.sol

Line number: 25

```
ETH_asset_pool_address = ETH_asset_pool
```

Severity and Impact Summary

Zero addresses may lead to errors or blocking functionality and may also cause a loss of funds.

Recommendation

It is recommended to add a zero address check to all mentioned cases.

OLD SOLIDITY COMPILER

Finding ID: BT-VV2-03
Severity: Low
Status: Accepted at Risk by Client

Description

Old compilers may have some open issues and are more likely to produce less optimized code.

Proof of Issue

File name: truffle-config.js

Line number: 92

```
compilers: {
   solc: {
       version: "0.8.8"
```

Severity and Impact Summary

Solc `0.8.8` is affected by the following issues - Nested Callata Array Abi Reencoding Size Validation - User-Defined Value Types Bug

Recommendation

It is recommended to use the latest stable build, currently `0.8.13`.

References

- https://docs.soliditylang.org/en/latest/bugs.html

Note from Vega team:

- The known bugs in compiler version 0.8.8 are not a risk for the current implementation of the Vega Protocol
- Moving to the very latest version also poses a risk that new features on Solidity have unknown bugs, thus the team has opted not to use the most up to date compiler version at this time.

AN OPTIONAL LIMITATION WITH THE COSTLY OPERATION

Finding ID: BT-VV2-04
Severity: **Informational**
Status: **Remediated**

Description

There is an optional variable that is saved to the storage and may be ignored.

Proof of Issue

File name: contracts/ERC20_Bridge_Logic_Restricted.sol

On `deposit_asset`, the user's lifetime deposits are saved into the storage:

Line number: 227

```
 user_lifetime_deposits[msg.sender][asset_source] += amount;
```

It is used at the beginning of the function to limit the maximum deposited amount:

Line number: 207

```
  require(exempt_depositors[msg.sender] ||
user_lifetime_deposits[msg.sender][asset_source] + amount <=
asset_deposit_lifetime_limit[asset_source], "deposit over lifetime limit");
```

But can be ignored if the user performs `exempt_depositor`:

Line number: 156

```
  function exempt_depositor() public override {
      require(!exempt_depositors[msg.sender], "sender already exempt");
      exempt_depositors[msg.sender] = true;
      emit Depositor_Exempted(msg.sender);
    }
```

Severity and Impact Summary

Usage of storage is the most costly operation in Solidity. Storing data that has no use (in the case when `exempt_depositor` was executed) leads to unnecessary GAS costs.

Recommendation

It is recommended to review this logic and decide whether storing the unused variable is required or can be simplified to reduce GAS cost.

PUBLIC VISIBILITY IS SET FOR FUNCTIONS THAT ARE NOT CALLED INTERNALLY

Finding ID: BT-VV2-05
Severity: **Informational**
Status: **Remediated**

Description

Public visibility is used for functions that should be accessible from other contracts, via transactions, and from the current contract. Functions that are not meant to be called internally should have external visibility.

Proof of Issue

File name: contracts/ERC20_Asset_Pool.sol

Line number: 38

```
    ERC20_Asset_Pool.set_multisig_control(address,uint256,bytes)
```

Line number: 56

```
    ERC20_Asset_Pool.set_bridge_address(address,uint256,bytes)
```

Line number: 68

```
    ERC20_Asset_Pool.withdraw(address,address,uint256)
```

File name: contracts/ERC20_Bridge_Logic_Restricted.sol

Line number: 43

```
ERC20_Bridge_Logic_Restricted.list_asset(address,bytes32,uint256,uint256,uint256,bytes
)
```

Line number: 61

```
    ERC20_Bridge_Logic_Restricted.remove_asset(address,uint256,bytes)
```

Line number: 91

```
ERC20_Bridge_Logic_Restricted.set_asset_limits(address,uint256,uint256,uint256,bytes)
```

Line number: 104

```
    ERC20_Bridge_Logic_Restricted.get_asset_deposit_lifetime_limit(address)
```

Line number: 111

```
    ERC20_Bridge_Logic_Restricted.get_withdraw_threshold(address)
```

Line number: 120

```
    ERC20_Bridge_Logic_Restricted.set_withdraw_delay(uint256,uint256,bytes)
```

Line number: 132

```
    ERC20_Bridge_Logic_Restricted.global_stop(uint256,bytes)
```

Line number: 145

```
    ERC20_Bridge_Logic_Restricted.global_resume(uint256,bytes)
```

Line number: 156

```
    ERC20_Bridge_Logic_Restricted.exempt_depositor()
```

Line number: 165

```
    ERC20_Bridge_Logic_Restricted.revoke_exempt_depositor()
```

Line number: 174

```
    ERC20_Bridge_Logic_Restricted.is_exempt_depositor(address)
```

Line number: 188

```
ERC20_Bridge_Logic_Restricted.withdraw_asset(address,uint256,address,uint256,uint256,b
ytes)
```

Line number: 205

```
    ERC20_Bridge_Logic_Restricted.deposit_asset(address,uint256,bytes32)
```

Line number: 235

```
    ERC20_Bridge_Logic_Restricted.is_asset_listed(address)
```

Line number: 240

```
    ERC20_Bridge_Logic_Restricted.get_multisig_control_address()
```

Line number: 246

```
    ERC20_Bridge_Logic_Restricted.get_vega_asset_id(address)
```

Line number: 252

```
    ERC20_Bridge_Logic_Restricted.get_asset_source(bytes32)
```

File name: contracts/ETH_Asset_Pool.sol

Line number: 33

```
    ETH_Asset_Pool.set_multisig_control(address,uint256,bytes)
```

Line number: 51

```
    ETH_Asset_Pool.set_bridge_address(address,uint256,bytes)
```

Line number: 62

```
    ETH_Asset_Pool.withdraw(address,uint256)
```

File name: contracts/ETH_Bridge_Logic.sol

Line number: 36

```
    ETH_Bridge_Logic.set_deposit_minimum(uint256,uint256,bytes)
```

Line number: 49

```
ETH_Bridge_Logic.set_deposit_maximum(uint256,uint256,bytes)
```

Line number: 64

```
ETH_Bridge_Logic.withdraw_asset(uint256,uint256,address,uint256,bytes)
```

Line number: 75

```
ETH_Bridge_Logic.deposit_asset(bytes32)
```

Line number: 85

```
ETH_Bridge_Logic.get_deposit_minimum()
```

Line number: 91

```
ETH_Bridge_Logic.get_deposit_maximum()
```

Line number: 96

```
ETH_Bridge_Logic.get_multisig_control_address()
```

Line number: 101

```
ETH_Bridge_Logic.get_vega_asset_id()
```

File name: contracts/MultisigControl.sol

Line number: 33

```
MultisigControl.set_threshold(uint16,uint256,bytes)
```

Line number: 47

```
MultisigControl.add_signer(address,uint256,bytes)
```

Line number: 62

```
MultisigControl.remove_signer(address,uint256,bytes)
```

Line number: 131

```
MultisigControl.get_valid_signer_count()
```

Line number: 136

```
MultisigControl.get_current_threshold()
```

Line number: 142

```
MultisigControl.is_valid_signer(address)
```

Line number: 148

```
MultisigControl.is_nonce_used(uint256)
```

Severity and Impact Summary

External functions may be more GAS efficient, especially with functions that receive a lot of data in parameters.

Recommendation

It is recommended to set external visibility for functions that are not called internally.

References

- https://docs.soliditylang.org/en/v0.7.4/contracts.html?highlight=external#visibility-and-getters

UNCOVERED TEST CASES

Finding ID: BT-VV2-06
Severity: **Informational**
Status: **Remediated**

Description

Some core functionality is not tested for negative scenarios.

Proof of Issue

Test case - try to withdraw asset with expired request

File name: contracts/ETH_Bridge_Logic.sol

Line number: 65

```
    require(expiry > block.timestamp, "withdrawal has expired");
```

Test case - try to withdraw an amount that is larger than the threshold using recently created request

File name: contracts/ERC20_Bridge_Logic_Restricted.sol

Line number: 190

```
    require(withdraw_thresholds[asset_source] > amount || creation +
default_withdraw_delay <= block.timestamp, "large withdraw is not old enough");
```

Test case - perform a failed deposit with non-standard ERC20

File name: contracts/ERC20_Bridge_Logic_Restricted.sol

Line number: 216

```
    case 0 {   // no return value but didn't revert
       result := true
    }
```

Test case - perform a failed withdraw with non-standard ERC20

File name: contracts/ERC20_Asset_Pool.sol

Line number: 76

```
    case 0 {   // no return value but didn't revert
       result := true
    }
```

Severity and Impact Summary

Negative scenarios are the potential vector of attack. In case of future code modifications, unit tests will be able to show if these scenarios are handled correctly.

Recommendation

It is recommended to cover negative scenarios with unit tests.

Our Process

Methodology

BTblock, a FYEO Company, uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

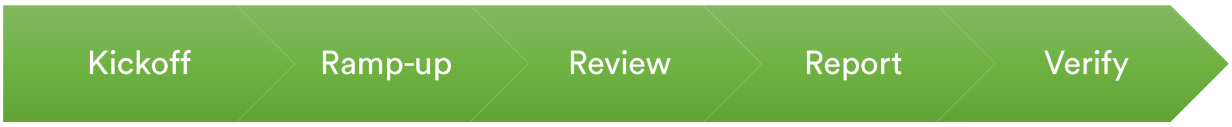

Figure 2: Methodology Flow

Kickoff

The project is kicked off as the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

Ramp-up

Ramp-up consists of the activities necessary to gain proficiency on the particular project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

Reporting

BTblock, a FYEO Company, delivers a draft report that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project. We may conclude that the overall risk is low but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We not only report security issues identified but also informational findings for improvement categorized into several buckets:

- Critical

- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

VERIFY

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes within a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

ADDITIONAL NOTE

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination.

THE CLASSIFICATION OF VULNERABILITIES

Security vulnerabilities and areas for improvement are weighted into one of several categories using, but is not limited to, the criteria listed below:

Critical – vulnerability will lead to a loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets
- The complexity to exploit is low
- The probability of exploit is high

High - vulnerability has potential to lead to a loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that cannot be found in the code
- All mismatches from the stated and actual functionality
- Unprotected key material
- Weak encryption of keys
- Badly generated key materials
- Txn signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - vulnerability hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-reviewed crypto functions
- Program crashes, leaves core dumps or writes sensitive data to log files

Low – vulnerability has a security impact but does not directly affect the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations