FunctionYard (Borg) A-4

Security Audit

September 19th, 2022
Version 1.0.0

Prepared by Halborn

Table of Contents

- Introduction
- Overall Assessment
- Specification
- Source Code
- Issue Descriptions and Recommendations
- Security Levels Reference
- Review Notes on VeryAPI.ZCoin and Spiral.xyz
- Disclaimer

Introduction

This document includes the results of the security audit for Borg/macro's smart contract code as found in the section titled "Source Code". The security audit was performed by the Macro security team from August 22 to September 2, 2022.

The purpose of this audit is to review the source code of certain Borg/macro's smart contracts, to identify potential security issues, to verify that the contracts follow the Borg/macro code and specification, to validate the correctness and security of the contracts' interfaces.

Disclaimer: While Macro's review is comprehensive and has surfaced some changes that should be made to the source code, this audit should not solely be relied upon for security, as no single audit is guaranteed to catch all possible bugs.

Overall Assessment

The following is an aggregation of issues found by the Macro Audit team:

Severity	Found	Acknowledged	Won't Fix	Addressed
Critical	1	1	0	1
Medium	3	3	0	3
Low	1	1	0	1
Code Quality	2	2	0	2
Informational	1	1	0	1
Gas Optimizations	1	1	0	1

Findings were quick to respond to these issues.

Specification

Our understanding of the specification was based on the following sources:

- Discussions on Slack with the Borg/macro team.
- The official Borg/macro documentation and more specifically provided documentation for contracts which were in scope of the performed audit. Specification.

Source Code

The following source code was reviewed during the audit:

- **Repository:** gitlab:macro
- **Commit Hash:** d2f1f2a2c7c2f8f78f3b8f9f9b2c7f5f1a7c7f3d8

Specifically, we audited the following contracts as part of FunctionYard's contract audit:

Contract	SHA256
contracts/macro/TokenERC20.sol	3d8f7c2f9c2f8c2c7f3d8f7c2f9c2f8c2c7f3d8f7c2f9c2f8c2c7f3d8f7c2f9c
contracts/libs/Pair.sol	7d2c2d2f8f8c7d7f7d8f7d2c2f8f7d2c2f8f7d2c2f8f7d2c2f8f7d2c2f8f7d2c
contracts/libs/FunctionTransfer.sol	2d7d8f8c2f8f7d2c7f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f
contracts/libs/FunctionTransferEx.sol	2d7d8f8c2f8f7d2c7f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f
contracts/libs/IsNonFungibleFunctionTransfer.sol	2c7f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f
contracts/interfaces/IFunctionPair.sol	2c7f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f
contracts/interfaces/IFunctionPairFactory.sol	2c7f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f
contracts/interfaces/IFunctionTransfer.sol	2c7f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f
contracts/interfaces/IFunctionTransferEx.sol	2c7f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f
contracts/interfaces/IIsNonFungibleFunctionTransfer.sol	2c7f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f
contracts/interfaces/IOwnable.sol	2c7f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f
contracts/interfaces/IOwnableEx.sol	2c7f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f
contracts/interfaces/IOwnableExNewOwner.sol	2c7f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f
contracts/interfaces/IOwnableNewOwner.sol	2c7f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f
contracts/interfaces/IFeeTokenERC20.sol	2c7f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f
contracts/interfaces/IFeeTokenERC20Ex.sol	2c7f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f

We audited the following contracts as part of TokenERC721 contract audit:

Contract	SHA256
contracts/macro/TokenERC721.sol	8c2c2c7f8c2f8c2c7f8c2c7f2f2d8c7f8c2f8c2c7f8c2c7f2f2d8c7f8c2f8c2c
contracts/libs/Pair.sol	7d2c2d2f8f8c7d7f7d8f7d2c2f8f7d2c2f8f7d2c2f8f7d2c2f8f7d2c2f8f7d2c
contracts/libs/FunctionTransfer.sol	2d7d8f8c2f8f7d2c7f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f
contracts/interfaces/IsNonFungibleFunctionTransferERC721/IsNonFungibleFunctionTransferERC721.sol	2c7f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f
contracts/interfaces/IFunctionPair.sol	2c7f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f
contracts/interfaces/IFunctionPairFactory.sol	2c7f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f
contracts/interfaces/IFunctionTransfer.sol	2c7f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f
contracts/interfaces/IFunctionTransferEx.sol	2c7f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f
contracts/interfaces/IIsNonFungibleFunctionTransfer.sol	2c7f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f
contracts/interfaces/IOwnable.sol	2c7f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f
contracts/interfaces/IOwnableEx.sol	2c7f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f
contracts/interfaces/IOwnableExNewOwner.sol	2c7f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f
contracts/interfaces/IOwnableNewOwner.sol	2c7f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f
contracts/interfaces/IFeeTokenERC721.sol	2c7f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f
contracts/interfaces/IFeeTokenERC721Ex.sol	2c7f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f8f7d2c7f8f8c2f

Note: This document contains an audit of the Borg/macro's contracts listed above. Specifically, the audit portrays only to the contracts themselves, and does not pertain to any other programs or tools including deployment scripts.

Issue Descriptions and Recommendations

Here on an issue (some by or 2, or scroll down to see them all):

CONTRACTSIZE(error) from running vulnerability

TokenERC20 changes price when quantity is 0

Native tokens can get locked in the contract

Use `LockNotin` instead of `LockIn`

`initializeHelper` is not bounded

`initialize(uint)` and `initializeHelper(uint)` can be set to 0x0

Documentation discrepancy for price in TokenERC20.sol

Inconsistency between Token contracts regarding pause/unpause functionality

TokenERC20 can have empty URI

Inconsistency among different contracts

Useless parameters

`IFOR0 IERC165Upgradeable.sol` draft status

Security Level Reference

We quantify issues in three parts:

1. The high impact/severity issues, breaking **Impact** of the issue:
   - Has bad things can get (for a vulnerability)
   - The significance of an improvement (for a code quality issue)
   - The amount of gas saved (for gas optimization)
2. The high/medium/low **likelihood** of the issue:
   - How likely is the issue to occur (for a vulnerability)
3. The overall critical/high/medium/low **severity** of the issue.

This third part - the severity level - is a summary of how much consideration the Client should give to fixing the issue, based roughly according to the table of guidelines below:

Severity	Description
0 - 4 Critical	We recommend the client **must** fix the issue, no matter what, because not fixing would mean **significant liability towards WRs, or loss**.
5 - 7 High	We recommend the client **must** address the issue, no matter what, because not fixing would mean **significant liability towards WRs**. Further functionality will be lost, or the contract behavior is against the stated specification.
8 - 9 Medium	We recommend the client to **seriously consider** fixing the issue as not fixing it would be a bad thing for the project, strong enough to impact the project significantly, either not in an economical manner.
10 - 12 Low	The risk is small, unlikely, or may not relevant to the project in a meaningful way.
13 Code Quality	Whether or not the project wants to develop a fix is up to the project and their priorities.
14 Informational	The issue identified does not give any obvious risk, but fixing it may improve the project in some ways, for example, better developer experience, or even certain aspects of protocol security.
15 Gas Optimizations	Savings and things to keep in mind when operating the protocol, to minimize gas costs.
16 Suggestions	The presented optimization/suggestion would save an unknown amount of gas/improve security/readability, or be worth the development cost of implementing it.

Issue Details

H-1 `CONTRACTSIZE(error)` from running vulnerability

Status	Impact	Likelihood
Fixed ✓	High	High

Impacting: TokenERC20.sol TokenERC721.sol

In TokenERC20, TokenERC721 and TokenERC1155, `CONTRACTSIZE(error)` allows the WriteRequest() fn. address to be zero:

```
function CONTRACTSIZE(uint8 errorNumber) external _pay_before_calculate_gas(msg.sender, msg.value) {
    address receiver = _pay_for_addresses(0, 0, _msgSender());
    _msg_fee.
}
```

The intended attack is similar to constructor's initial signature such that the sender and receiver fee tokens, instead of needing to specify the recipient's address.

However because the recipient's fee tokens are attacked, an Firefox a `CONTRACTSIZE(error)` transaction, receiving the tokens and leaving the victim with nothing.

If there is no price, the attacker gets the tokens for free. This is particularly bad when the contract has a mint price or an AMM, and the contract owner wanted to grant someone free tokens.

If a price is present, the attacker does have to pay it. But the situation is still problematic if the price takes a discount, or if they are able to trick/trap the contract owner into paying.

Note that this behavior also exists in Signature/Proxy.sol.

Consider forcing WriteRequest() `_fp` address to always be defined.

M-1 TokenERC20 changes price when quantity is 0

Status	Impact	Likelihood
Fixed ✓	Medium	Medium

Impacting: TokenERC20.sol

In TokenERC20, TokenERC721 and TokenERC1155 functions are include a `payable` and a `_calculate`, which use the either `MATIC_TOKEN` or `ETH` to calculate the price.

Suppose the currency specified in the request is some ERC20 token and the sender, `_fp` and `_receiver` addresses are zero, the `_calculate` function will return a non-zero price, because the contract "thinks (wants)" to be minted tokens for free (zero recipient).

Consider adding a check in the `_calculate()` function which prevents them being set to the contract when currency is not set to `MATIC_TOKEN`:

```
if (_currency == Currency(Function.MATIC_TOKEN)) {
    // if currency is ETH, check that msg.value is enough
    require(msg.value >= _fp, "Not enough ETH");
} else {
    _fp = 0;
}
```

M-2 Use `_LockNotin` instead of `_LockIn`

Status	Impact	Likelihood
Fixed ✓	High	Medium

Impacting: Config Standards

In TokenERC20, TokenERC721 and `_LockIn()` instead of `_LockNotin()`, to mint the actual tokens:

```
_mint(_to, tokenIndexMax);
```

This can lead to tokens being locked in those contracts with no recovery.

Consider using `_LockNotin()` instead of `_LockIn()` to check if a contract recipient can receive an ERC721 token and avoid of accepting ERC721 tokens.

M-3 `initializeHelper()` is not bounded

Status	Impact	Likelihood
Fixed ✓	High	Medium

Impacting: Input Ranges

In TokenERC20, TokenERC721 and TokenERC1155, the `initialize()` functions don't have a boundary check for `_initializeHelper()`.

Also in the `initializeHelper()`, functions, the only restriction is `_initializeHelper() != 0`. `MAX_FEE_PER_ERC` is set `MAX_FEE` to zero since it is dependency on the `_fp`, the minting would fail because of underflow in case Thirdfeet fee is applied. See following fee:

```
_fee = (
    _receiverFeeTotal.mul(_receiverFeeDenom)
    .sub(_receiverFee)
    .sub(_thirdfeeFee)
    .sub(_protocolFeeAmount)
    .div(_receiverFeeDenom) - _fee) / _underflow
);
```

Consider setting a reasonable upper limit for `initializeHelper()`.

L-1 `initialize(address)` is not bounded

Status	Impact	Likelihood
Fixed ✓	Low	Low

Impacting: Input Ranges

In TokenERC20, TokenERC721 and TokenERC1155, the `initialize()` functions don't have a boundary check for `_initializeHelper()`.

Also in the `initialize()`, `setMaxLockNotin(uint)` and `setMaxTokenIdMint()`, in all contracts, there is no upper limit for `_initializeHelper()`.

`initialize(address)` has no true validation, `_initializeHelper()`.

As a consequence, when using MATIC_TOKEN & ETH, either could be burned in the zero address.

In the case of an ERC20 transfer, the transaction would be reverted for `OpenZeppelin/safe-contracts`.

Consider adding checks to not allow address 0x0 for `initialize(address)` and `initialize(address)`.

M-4 Documentation discrepancy for price in TokenERC20.sol

Status	Impact	Likelihood
Fixed ✓	Medium	Medium

Impacting: Documentation

Thirdweb's Signature/mint documentation defines one of the mint request parameters as `pricePerToken`.

However, in TokenERC20.sol, the price specified in the mint request is considered the total price (based).

This also creates an inconsistency with TokenERC721.sol, which uses `pricePerToken` as expected.

Consider:

1. Updating TokenERC20.sol to use `pricePerToken` instead of `price` and accepting `setCleanPrice` function accordingly to include a `quantity` parameter or
2. Document the differing behavior of TokenERC20.

M-5 Inconsistency between Token contracts regarding pause/unpause functionality

Status	Impact	Likelihood
Fixed ✓	Medium	Medium

Impacting: Code Quality

In TokenERC20 there is the possibility to pause/unpause token transfers, in contrast, TokenERC20 and TokenERC1155 don't provide this functionality.

Consider to add pause/unpause capability for token transfers in TokenERC20 and TokenERC1155.

L-2 TokenERC721 can have empty URI

Status	Impact	Likelihood
Fixed ✓	Low	Medium

Impacting: Code Quality

In TokenERC1155, `_baseURI` function the minting results when the `_id` parameter in `MintRequest` is empty:

```
if (bytes(_baseURI).length == 0 ) {
    _requireOwned(_id);
    return "";
}
```

In contrast, TokenERC20 doesn't check for empty uri parameter.

Consider adding this check to TokenERC20 as well.

M-6 Inconsistency among different contracts

Status	Impact	Likelihood
Fixed ✓	Medium	Low

Impacting: Code Quality

In TokenERC20 and TokenERC1155 define their user `MATIC_TOKEN` constant:

```
address private constant MATIC_TOKEN = address(0xf0ee7a58a526338f7dbcfbb8b0f1d25ccf1f4f11);
```

Instead, TokenERC721 uses `IsNonFunctionTokenERC721.MATIC_TOKEN`.

The `IsNonFunctionTokenERC721` function is written differently in TokenERC20 than TokenERC721 and TokenERC1155. In TokenERC20 and TokenERC1155, `IsNonFunctionTokenERC721` is used to call specific parameters. In addition, TokenERC721 contains an `_fee` variable to determine `_receiverFee` within `_mintRequestLogic()` instead of `_calculatePrice()`.

Consider using the same `IsNonFunctionTokenERC721` function in all contracts.

L-3 Useless parameters

Status	Impact	Likelihood
Fixed ✓	Low	Low

Impacting: Code Quality

In TokenERC20 and TokenERC721 define their user `MATIC_TOKEN` constant:

```
address private constant MATIC_TOKEN = address(0xf0ee7a58a526338f7dbcfbb8b0f1d25ccf1f4f11);
```

Instead, TokenERC721 uses `IsNonFunctionTokenERC721.MATIC_TOKEN`.

The `IsNonFunctionTokenERC721` function is written differently in TokenERC20 than TokenERC721 and TokenERC1155. In TokenERC20 and TokenERC1155, `IsNonFunctionTokenERC721` is used to call specific parameters. In addition, TokenERC721 contains an `_fee` variable to determine `_receiverFee` within `_mintRequestLogic()` instead of `_calculatePrice()`.

Consider using the same `IsNonFunctionTokenERC721` function in all contracts.

Review Notes on VeryAPI.ZCoin and Spiral.xyz

- Spiral.xyz: Spiral.xyz is a DeFi protocol on Ethereum. Spiral.xyz's contracts are based on Curve Finance's contracts, and Spiral.xyz's contracts are based on Curve Finance's contracts. Spiral.xyz's contracts are based on Curve Finance's contracts.
- VeryAPI.ZCoin: VeryAPI.ZCoin is a DeFi protocol on Ethereum. VeryAPI.ZCoin's contracts are based on Curve Finance's contracts, and VeryAPI.ZCoin's contracts are based on Curve Finance's contracts. VeryAPI.ZCoin's contracts are based on Curve Finance's contracts.

Disclaimer

Disclaimer: While Macro's review is comprehensive and has surfaced some changes that should be made to the source code, this audit should not solely be relied upon for security, as no single audit is guaranteed to catch all possible bugs.

and *TokenERC1155*.

- `_afterTokenTransfer()` is only overridden in *TokenERC20* without any changed behavior.
- `MAX_BPS` is declared as:
  - `uint128 internal constant` in *TokenERC20*.
  - `uint256 private constant` in *TokenERC721* and in *TokenERC1155*.
- `platformFeeBps` is declared as:
  - `uint128 internal` in *TokenERC20*.
  - `uint128 public` in *TokenERC721* and in *TokenERC1155*.
- *TokenERC721* and *TokenERC1155* both inherit from OpenZeppelin's `ReentrancyGuardUpgradeable` and call its `__ReentrancyGuard_init()`. However, *TokenERC20* does call this initializer, even though it inherits the same contract.

G-4 Calldata parameters

TOPIC	STATUS	GAS SAVINGS
Gas Optimization	Fixed	Low

In *TokenERC721* and *TokenERC1155* function `collectPrice` uses `memory` keyword for the `MintRequest` parameter. Consider using `calldata` allocation instead to save gas costs. In *TokenERC721* and *TokenERC1155* function `collectPrice` uses `memory` keyword for the `MintRequest` parameter. Consider using `calldata` allocation instead to save on gas cost.

I-1 `draft-EIP712Upgradeable.sol` draft status

TOPIC	IMPACT
Informational	Informational *

See OZ's statement regarding draft status here:
https://docs.openzeppelin.com/contracts/3.x/api/drafts

Review Notes on VoteERC20.sol and Split.sol

Scope: VoteERC20.sol and Split.sol

For these two contracts, we opted to do a review instead of a full audit, due there being no test suite and low documentation. However, we were happy to provide suggestions and considerations in this way.

VoteERC20.sol

(1) Both `proposalIndex` and `proposalId` are used

Resolution: Wont Do.

The two common options to store proposals on-chain in DAO proposal systems are:

1. Store all function calls and arguments on-chain.
2. Store a 32 byte hash of all function calls on-chain, and post the source of that hash off-chain (e.g. on a public forum).

The former leans more towards decentralization, as anyone can read on-chain data to verify the proposed function calls at any time.

On the other hand, the latter saves on gas costs, as storing function calls on-chain can get expensive.

VoteERC20 takes the approach of both:

1. It uses Open Zeppelin's GovernorUpgradeable contract, which uses approach #2.
2. It extends `propose()` to store all calls in its own proposals variable (approach #1), including the 32 byte hash from the parent OZ contract.

Doing both is redundant, which will result in increased gas costs for no benefit. It may also cause confusion, as there are arguably two ids: the 32 byte hash, and the proposal index.

Consider taking one approach or another. If approach #1 is desired, consider using Compound's Governor Bravo contracts. Otherwise, it may be sufficient that the proposed function calls are emitted in an event.

Split.sol

(1) DoS vulnerability in `distribute` function

Resolution: Wont Do.

In *Split* contract, `distribute` function loops through each payee address and sends assets performing checks and reverting if any of these calls fails. If one of the payees reverts the transfer the entire `distribute` function will revert in every call. This can also be intentionally done by a smart contract address as a payee.

The vulnerability is not that bad, since anyone is able to call `release()` for their own account at any time. However, consider documenting the dangers 3rd party contracts or other thirdweb contracts relying on this function.

Also consider not reverting on failed transfers, if atomicity is not important.

(2) OpenZeppelin's version of PaymentSplitterUpgradeable.sol has been updated

Resolution: Updated preset version to reflect OZ changes.

Split contract inherits from `PaymentSplitterUpgradeable`. It is not referencing the OpenZeppelin version directly but instead is using a preset version:

```
import "./openzeppelin-presets/finance/PaymentSplitterUpgradeable.sol";
```

OpenZeppelin did some slight changes recently on this contract and has added `releasable` functions. See here: https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/commit/b392c249e2c72434c438e0e495af1bacbc6cfd4f#diff-ec6a964c766447336ea6842ff3214ac85da5144ccf44d92e100d3d4acfd907c4

Consider to update the preset version to include those latest changes, as `releasable()` is a useful function.

(3) Inconsistent zero check

Resolution: The ThirdWeb fee was removed.

After calling `thirdwebFee.getFeeInfo()`, the Split contract checks to see if their values are not equal to zero. This seems to be inconsistent with the rest of the codebase.

If this check is important, consider reevaluating other parts of the codebase that do not have this check.

(4) Inaccurate comments

Resolution: Removed related comments.

On `distribute` function, *Split* contract has comments referring to an function named `_appPay` but it does not seem to exist in the codebase.

```
function distribute() public virtual {
    uint256 count = payeeCount();
    for (uint256 i = 0; i < count; i++) {
        ****// note: `_release` should not fail because payee always has shares
        _release(payable(payee(i)));
    }
}

function distribute(IERC20Upgradeable token) public virtual {
    uint256 count = payeeCount();
    for (uint256 i = 0; i < count; i++) {
        ****// note: `_release` should not fail because payee always has shares
        _release(token, payee(i));
    }
}
```

Disclaimer

Macro makes no warranties, either express, implied, statutory, or otherwise, with respect to the services or deliverables provided in this report, and Macro specifically disclaims all implied warranties of merchantability, fitness for a particular purpose, noninfringement and those arising from a course of dealing, usage or trade with respect thereto, and all such warranties are hereby excluded to the fullest extent permitted by law.

Macro will not be liable for any lost profits, business, contracts, revenue, goodwill, production, anticipated savings, loss of data, or costs of procurement of substitute goods or services or for any claim or demand by any other party. In no event will Macro be liable for consequential, incidental, special, indirect, or exemplary damages arising out of this agreement or any work statement, however caused and (to the fullest extent permitted by law) under any theory of liability (including negligence), even if Macro has been advised of the possibility of such damages.

The scope of this report and review is limited to a review of only the code presented by the Emergent team and only the source code Macro notes as being within the scope of Macro's review within this report. This report does not include an audit of the deployment scripts used to deploy the Solidity contracts in the repository corresponding to this audit. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. In this report you may through hypertext or other computer links, gain access to websites operated by persons other than Macro. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such websites' owners. You agree that Macro is not responsible for the content or operation of such websites, and that Macro shall have no liability to your or any other person or entity for the use of third party websites. Macro assumes no responsibility for the use of third party software and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.