Zellic

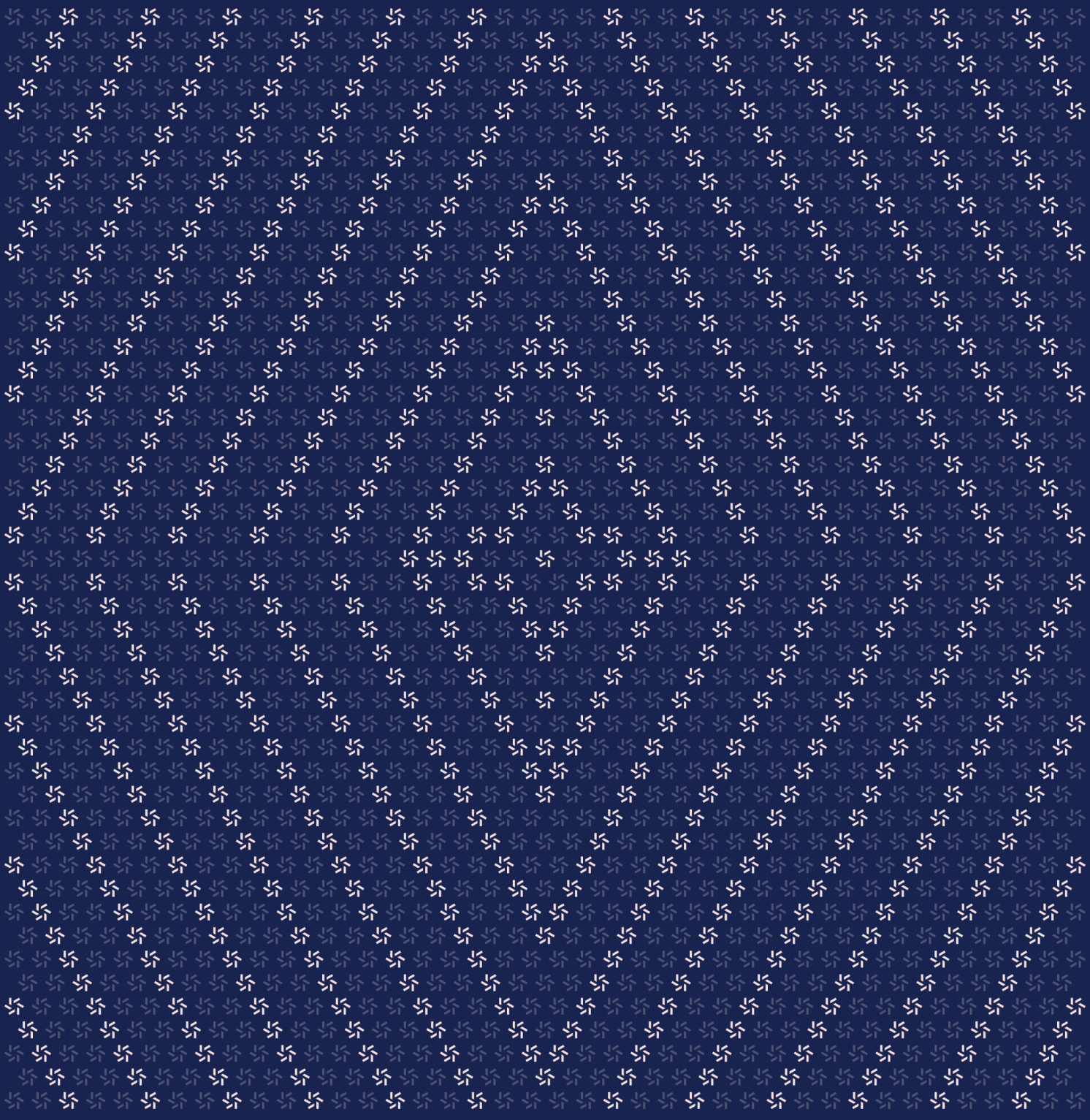
Prepared for
Mihai Dancaescu
warp.green

Prepared by
Jaeeu Kim
Jisub Kim
Zellic

June 13, 2024

warpdotgreen-cli

Smart Contract Security Assessment



Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for warp.green from June 10th to June 12th, 2024. During this engagement, Zellic reviewed warpdotgreen-cli's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Could a malicious user steal funds from the bridge contracts?
- Could a malicious user make the bridge contracts unusable?
- Could a malicious user execute unexpected actions on the bridge contracts?
- Could a malicious user mint or burn tokens in unexpected ways?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody
- Chia side of the bridge

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

This project is a bridge project for cross-chain purposes. Therefore, it involves elements outside the contract that utilize the contract's data — for example, events. However, these elements are not within the scope of this review and therefore require careful attention.

1.4. Results

During our assessment on the scoped warpdotgreen-cli contracts, there were no security vulnerabilities discovered.

Zellic recorded its notes and observations from the assessment for warp.green's benefit in the

Discussion section (3. ↗).

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	0
Informational	0

2. Introduction

2.1. About warpdotgreen-cli

warp.green contributed the following description of warpdotgreen-cli:

> The warp.green protocol facilitates the communication of messages across supported blockchains (Chia and Ethereum/Base) through a trusted set of validators.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood.

We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped contracts itself. These observations — found in the Discussion (3. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

warpdotgreen-cli Contracts

Type	Solidity
Platform	EVM-compatible

Target: cli

Repository	https://github.com/warpdotgreen/cli ↗
Version	`291710292739bfc4c08e5f5ee0b8ac98a400e662`

Programs	`ERC20Bridge.sol`
	`Portal.sol`
	`MilliETH.sol`
	`WrappedCAT.sol`

2.4. Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of four person-days. The assessment was conducted over the course of three calendar days.

Contact Information

The following project manager was associated with the engagement:

Chad McDonald
Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

Jaeeu Kim
Engineer
jaeeu@zellic.io

Jisub Kim
Engineer
jisub@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

June 10, 2024	Start of primary review period
June 12, 2024	End of primary review period

3. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

3.1. Anyone can emit the `SendMessage` event directly without transferring tokens

It is possible for anyone to directly call the `SendMessage` function of the portal, allowing arbitrary `MessageSent` events to be emitted without actually transferring tokens. In `eth_follower`, the `getEventByIntNonce` function does not check the `msg.sender` of the event, which means that a `MessageSent` event created by an arbitrary user can be signed through the `messageSigner`. In this case, the `msg.sender` will be the user's address.

The warp.green team has acknowledged this. This is intended behavior because the portal is for sending and receiving messages, so it should be callable by any caller. Another bridge contract could use this portal to send/receive messages for cross-chain communication. For this design, destination bridge contracts have the responsibility of checking messages sent from the source bridge.

For this reason, they check the message sources on the Chia side of the bridge app ↗.

3.2. Reentrancy in `sendMessage` by a miner

The `toll` will be sent to the miner, but a miner that is `block.coinbase` can be a contract address.

The concern was raised that if a malicious miner were to reenter the `sendMessage` emitting the `MessageSent` event, it could potentially exploit a reentrancy vulnerability in the `sendMessage` function, leading to out-of-order event processing.

For example, if the expected event sequence is `1, 2, 3, 4, 5, 6`, a reentrancy attack could result in an event queue like `1, 3, 2, 4, 5, 6`, disrupting the intended sequential order.

The warp.green team provided the following response:

> We considered this vector during the design phase. For applications that require strict sequential processing of messages, we recommend including a sequence number within the message payload itself. It's important to note that Ethereum nonces are incremental primarily for convenience. If you examine the messages from XCH, you'll notice that the nonces are non-sequential. As long as the nonces are unique for a given chain, the system will function correctly.

3.3. Mismatched `transferTip` condition between token minting and burning

There is a mismatched condition between token minting and burning. The `receiveMessage` function has a stricter condition for minting tokens, as it fails when the `transferTip` is `0`. On the other hand, the `bridgeBack` function allows burning tokens even when the `transferTip` is `0`.

The warp.green team provided the following response:

> The difference is because `mojoToTokenRatio` is either 10^6 (Warped XCH) or 10^12 (any other Warped CAT), so transfer tip will be higher than 0. When bridging back, you are cutting decimals, so we wanted to enforce a minimum tip.

4. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

4.1. Module: ERC20Bridge.sol

Function: `bridgeEtherToChia(byte[32] _receiver, uint256 _maxMessageToll)`

This function is used to bridge native Ether to Chia by first wrapping it into milliETH (or WETH). This function wraps Ether into an ERC-20, sends the portal tip, and sends a message to mint tokens on Chia.

Inputs

- `_receiver`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Receiver puzzle hash for the tokens.
- `_maxMessageToll`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of the maximum message toll.

Branches and code coverage

Intended branches

- Call the deposit function of IWETH to wrap the Ether.
  - ☑ Test coverage
- Call the decimals function of IWETH and update the factor.
  - ☑ Test coverage
- Invoke the `_handleBridging` function.
  - ☑ Test coverage

Negative behavior

- Check if the message toll is less than the maximum message toll.
  - ☑ Negative test
- Check if the amount after toll is greater than `wethToEthRatio` and divisible by `weth-`

ToEthRatio.

☑ Negative test

Function: `bridgeToChiaWithPermit(address _assetContract, byte[32] _receiver, uint256 _amount, uint256 _deadline, uint8 _v, byte[32] _r, byte[32] _s)`

This function is used to bridge ERC-20 tokens to Chia with a permit allowing token spending. This function uses an ERC-20 permit for gas-efficient token approval and transfer in a single transaction.

Inputs

- `_assetContract`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the ERC-20 token to bridge.
- `_receiver`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Receiver puzzle hash for the wrapped tokens.
- `_amount`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Amount to bridge to Chia, in Mojos.
- `_deadline`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of the permit deadline.
- `_v`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of the permit signature `v`.
- `_r`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of the permit signature `r`.
- `_s`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of the permit signature `s`.

Branches and code coverage

Intended branches

- Call the decimals function of the asset contract and update the factor.
  - ☑ Test coverage
- Call the permit function of target asset contract.
  - ☑ Test coverage
- Invoke the `_handleBridging` function.
  - ☑ Test coverage

Negative behavior

- Revert if `msg.value` is not equal to the message toll.
  - ☑ Negative test

Function call analysis

- `ERC20Decimals(_assetContract).decimals()`
  - **What is controllable?** `_assetContract` — it is not whitelisted.
  - **If the return value is controllable, how is it used and how can it go wrong?** If the decimals of the asset contract are manipulated, the factor will be set arbitrarily.
- `IERC20Permit(_assetContract).permit(msg.sender, address(this), _amount * factor, _deadline, _v, _r, _s)`
  - **What is controllable?** `_assetContract` — it is not whitelisted.

Function: `bridgeToChia(address _assetContract, byte[32] _receiver, uint256 _mojoAmount)`

This function is used to bridge ERC-20 tokens to Chia via the warp.green protocol. This function transfers tokens to this contract, redirects the portal tip, and sends a message for tokens to be minted on Chia.

Inputs

- `_assetContract`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the ERC-20 token to bridge.
- `_receiver`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Receiver puzzle hash for the tokens.

- `_mojoAmount`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Amount to bridge to Chia, in Mojos.

Branches and code coverage

Intended branches

- Invoke the `_handleBridging` function.
  - ☑ Test coverage

Negative behavior

- Revert if msg.value is not equal to the value of the message toll.
  - ☑ Negative test

Function: `initializePuzzleHashes(byte[32] _burnPuzzleHash, byte[32] _mintPuzzleHash)`

This function initializes the puzzle hashes used. It can only be called once. This function is expected to be called during deployment.

Inputs

- `_burnPuzzleHash`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of the burn puzzle hash.
- `_mintPuzzleHash`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of the mint puzzle hash.

Branches and code coverage

Intended branches

- Update the burn puzzle hash and mint puzzle hash.
  - ☑ Test coverage

Negative behavior

- Revert if the burn puzzle hash or mint puzzle hash is already set.

- ☑ Negative test

Function: `receiveMessage(byte[32], byte[3] _source_chain, byte[32] _source, byte[32][] _contents)`

This function is used to receive messages from the portal contract and process them. This function transfers tokens to the receiver and the portal tip to the portal. This function is called by the portal contract.

Inputs

- `nonce`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Not used in the function.
- `_source_chain`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Source chain ID.
- `_source`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Source puzzle hash (address of sender).
- `_contents`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Message contents (asset contract, receiver, and amount).

Branches and code coverage

Intended branches

- Update the amount using the decimals of the asset contract.
  - ☑ Test coverage
- Calculate the transfer tip.
  - ☑ Test coverage
- If the asset contract is not IWETH, transfer the amount to the receiver and the tip to the portal.
  - ☑ Test coverage
- If the asset contract is IWETH, withdraw the amount and send it to the receiver and the tip to the portal.
  - ☑ Test coverage

Negative behavior

- Revert if the message sender is not the portal.
  - ☑ Negative test
- Revert if the source is not the `burnPuzzleHash` of the contract.
  - ☑ Negative test
- Revert if the source chain is not the `otherChain` of the contract.
  - ☑ Negative test
- Revert if the amount is less than the transfer tip.
  - ☑ Negative test

Function call analysis

- `ERC20Decimals(assetContract).decimals()`
  - **What is controllable?** `assetContract` — it is not whitelisted.
  - **If the return value is controllable, how is it used and how can it go wrong?** If the decimals of the asset contract are manipulated, the amount will be calculated incorrectly.

Function: `_handleBridging(address _assetContract, bool _transferAsset, byte[32] _receiver, uint256 _amount, uint256 _messageToll, uint256 _mojoToTokenFactor)`

This function is used to build message contents and send a message to the portal contract.

Inputs

- `_assetContract`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the ERC-20 token to bridge.
- `_transferAsset`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Whether to transfer the asset to this contract or not.
- `_receiver`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Receiver puzzle hash for the wrapped tokens (receiver address).
- `_amount`
  - **Control**: Arbitrary.
  - **Constraints**: None.

    - **Impact**: Amount of CAT tokens to be minted on Chia, in Mojos.
- `_messageToll`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Value of the message toll.
- `_mojoToTokenFactor`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: A power of 10 to convert from CAT amount (Mojos) to the ERC-20 token's smallest unit.

Branches and code coverage

Intended branches

- Calculate the transfer tip.
    - ☑ Test coverage
- Build message contents.
    - ☑ Test coverage
- If `_transferAsset` is true, transfer the asset to this contract from the sender.
    - ☑ Test coverage
- Transfer the tip to the portal.
    - ☑ Test coverage
- Send a message to the portal.
    - ☑ Test coverage

Negative behavior

- Revert if the amount is less than the transfer tip.
    - ☑ Negative test

4.2. Module: MilliETH.sol

Function: `deposit()`

This function is used to mint milliETH (1,000 milliETH: 1 ETH ratio) equivalent to the deposited Ether value. milliETH is an ERC-20 token, where 1 milliETH is equivalent to 1/1000th of one Ether. It requires the deposited ETH to be divisible by 1e12 WEI for correct conversion.

Branches and code coverage

Intended branches

- Mint milliETH equivalent successfully (1,000 milliETH: 1 ETH ratio).

☑ Test coverage

Negative behavior

- Revert if `msg.value` is lower than zero.
  ☑ Negative test
- Revert if `msg.value` % 1e12 wei is not equal to zero.
  ☑ Negative test

Function: `withdraw(uint256 amount)`

This function is used to withdraw milliETH; equivalent Ether value is sent back. This burns the specified amount of milliETH and sends the equivalent amount of ETH back to the sender.

Inputs

- `amount`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of the amount to withdraw.

Branches and code coverage

Intended branches

- Burn the given amount of milliETH.
  ☑ Test coverage
- Send the equivalent amount of ETH to the sender.
  ☑ Test coverage

Negative behavior

- Revert if the amount is lower than zero.
  ☑ Negative test
- Revert if withdrawal has failed.
  ☑ Negative test

4.3. Module: Portal.sol

Function: `initialize(address _coldMultisig, uint256 _messageToll, address[] _signers, uint256 _signatureThreshold, byte[3][] _supportedChains)`

This function is used to initialize the contract with the required parameters. It sets the initial owners, message toll, signers, and signature threshold. This function can only be called once and called in the same transaction as deployment.

Inputs

- `_coldMultisig`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of owner.
- `_messageToll`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of message toll price.
- `_signers`
  - **Control**: Arbitrary.
  - **Constraints**: Nonzero address.
  - **Impact**: Array of addresses authorized to attest messages were sent to this chain.
- `_signatureThreshold`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Number of required signatures to validate a message.
- `_supportedChains`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Array of supported chains.

Branches and code coverage

Intended branches

- Set the contract's settings.
  - ☑ Test coverage

Negative behavior

- Revert if this function is called more than once.

- ☑ Negative test
- Revert if provided signer address is zero.
  - ☑ Negative test

Function: `receiveMessage(byte[32] _nonce, byte[3] _source_chain, byte[32] _source, address _destination, byte[32][] _contents, bytes _sigs)`

This function is used to receive and relay a cross-chain message from another blockchain. It verifies the message signatures, checks for replay, and if valid, forwards the message to the destination contract. Normally called by the user that sent the message.

Inputs

- `_nonce`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Nonce of the message.
- `_source_chain`
  - **Control**: Arbitrary.
  - **Constraints**: Valid mapping of supported chains.
  - **Impact**: Chain ID of the source chain.
- `_source`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the source.
- `_destination`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the destination contract.
- `_contents`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Content of the message being processed.
- `_sigs`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Signatures verifying the message.

Branches and code coverage

Intended branches

- Mark the nonce as used.
  - ☑ Test coverage
- Check the signatures of the message.
  - ☑ Test coverage
- Forward the message to the destination contract by calling the `receiveMessage` function of the destination contract.
  - ☑ Test coverage

Negative behavior

- Revert if the source chain is not supported.
  - ☑ Negative test
- Revert if the length of the signatures is not equal to the signature threshold.
  - ☑ Negative test
- Revert if the nonce has been used before.
  - ☑ Negative test
- Revert if the signatures are not in order.
  - ☑ Negative test
- Revert if the signer is not authorized.
  - ☑ Negative test

Function call analysis

- `IPortalMessageReceiver(_destination).receiveMessage(_nonce, _source_chain, _source, _contents)`
  - Destination contract is expected as the bridge contract address. The bridge contract will execute token transfer using `_contents` params.

Function: `rescueAsset(address _assetContract, address[] _receivers, uint256[] _amounts)`

This function is used to send stuck ERC-20 tokens, or ERC-20 tokens mistakenly sent to the contract, to a list of addresses. This function is only callable by the contract owner.

Inputs

- `_assetContract`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the ERC-20 token to transfer.

- `_receivers`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Array of addresses to receive the tokens.
- `_amounts`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Corresponding amounts of tokens to be sent to the receivers.

Branches and code coverage

Intended branches

- Send tokens to the receivers.
  - ☑ Test coverage

Negative behavior

- Revert if the function is not called by the owner.
  - ☑ Negative test

Function: `rescueEther(address[] _receivers, uint256[] _amounts)`

This function is used to send stuck Ether, or Ether mistakenly sent to the contract, to a list of addresses. This function is only callable by the contract owner.

Inputs

- `_receivers`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Array of addresses to receive the Ether.
- `_amounts`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Corresponding amounts of Ether to be sent to the receivers.

Branches and code coverage

Intended branches

- Send Ether to the receivers.
  - ☑ Test coverage

Negative behavior

- Revert if the function is not called by the owner.
  - ☑ Negative test

Function: `sendMessage(byte[3] _destination_chain, byte[32] _destination, byte[32][] _contents)`

This function is used to send a cross-chain message to another blockchain. It charges a toll and emits a `MessageSent` event. This event will be monitored by the bridge system, which will relay the message to the destination chain.

Inputs

- `_destination_chain`
  - **Control**: Arbitrary.
  - **Constraints**: Valid `supportedChains` mapping.
  - **Impact**: Chain ID of the destination chain.
- `_destination`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Target address of the destination chain.
- `_contents`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Content of the message being sent, including token address, amount, and recipient address.

Branches and code coverage

Intended branches

- Check message toll is paid.
  - ☑ Test coverage
- Check destination chain is supported.
  - ☑ Test coverage
- Increment the nonce.
  - ☑ Test coverage
- Send the toll to the miner.
  - ☑ Test coverage
- Emit a `MessageSent` event.
  - ☑ Test coverage

Negative behavior

- Revert if message toll is not paid.
  - ☑ Negative test
- Revert if destination chain is not supported.
  - ☑ Negative test

Function call analysis

- `block.coinbase.call()`
  - **What happens if it reverts, reenters or does other unusual control flow?** If miner is the contact address, the reentrance risk exists, which can result in a nonsequential `ethNonce` occurrence of `MessageSent`.

Function: `updateMessageToll(uint256 _newValue)`

This function is used to update the message toll fee required to send messages. This function is only callable by the contract owner.

Inputs

- `_newValue`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: New toll fee.

Branches and code coverage

Intended branches

- Update the message toll fee.
  - ☑ Test coverage

Negative behavior

- Revert if the function is not called by the owner.
  - ☑ Negative test
- Revert if the new value is the same as the current value.
  - ☑ Negative test

Function: `updateSignatureThreshold(uint256 _newValue)`

This function is used to update the threshold of required signatures for message verification. This function is only callable by the contract owner.

Inputs

- `_newValue`
  - **Control**: Arbitrary.
  - **Constraints**: Nonzero value.
  - **Impact**: New number of required signatures.

Branches and code coverage

Intended branches

- Update the threshold of required signatures.
  - ☑ Test coverage

Negative behavior

- Revert if the function is not called by the owner.
  - ☑ Negative test
- Revert if the value is zero.
  - ☑ Negative test
- Revert if the new value is the same as the current value.
  - ☑ Negative test

Function: `updateSigner(address _signer, bool _newValue)`

This function is used to update the authorization status of a signer (validator). This function is only callable by the contract owner.

Inputs

- `_signer`
  - **Control**: Arbitrary.
  - **Constraints**: Nonzero address.
  - **Impact**: Address of the signer to update.
- `_newValue`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: New authorization status (true for authorized, false for not authorized).

Branches and code coverage

Intended branches

- Update the authorization status of the signer.
  - ☑ Test coverage

Negative behavior

- Revert if the function is not called by the owner.
  - ☑ Negative test
- Revert if the signer address is zero.
  - ☑ Negative test
- Revert if the status is not different.
  - ☑ Negative test

Function: `updateSupportedChain(byte[3] _chainId, bool _supported)`

This function is used to update the supported status of a chain with a given ID. This function is only callable by the contract owner.

Inputs

- `_chainId`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of the chain ID.
- `_supported`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of the supported status.

Branches and code coverage

Intended branches

- Update the supported status of the chain.
  - ☑ Test coverage

Negative behavior

- Revert if the function is not called by the owner.
  - ☑ Negative test
- Revert if the new value is the same as the current value.
  - ☑ Negative test

4.4. Module: WrappedCAT.sol

Function: `bridgeBack(byte[32] _receiver, uint256 _mojoAmount)`

This function is used to burn wrapped CAT ERC-20 tokens and send a message to unlock the original CAT on the Chia network.

Inputs

- `_receiver`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Receiver puzzle hash for tokens.
- `_mojoAmount`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Amount of CAT tokens (in Mojos) to unlock on Chia.

Branches and code coverage

Intended branches

- Set minimum transfer tip.
  - ☑ Test coverage
- Burn CAT ERC-20 tokens correctly.
  - ☑ Test coverage
- Invoke `sendMessage` to the portal to unlock CAT tokens.
  - ☑ Test coverage

Negative behavior

- Revert if msg.value is not equal to the value of the message toll.
  - ☑ Negative test
- Revert if Mojo amount is less than the transfer tip, which is 1 Mojo at least.
  - ☑ Negative test
- Revert if tip is equal to the amount, causing the bridged amount to be zero.
  - ☑ Negative test

Function: `initializePuzzleHashes(byte[32] _lockerPuzzleHash, byte[32] _unlockerPuzzleHash)`

This function is used to initialize puzzle hashes for locking and unlocking tokens. It can only be called once. This function is expected to be called during deployment.

Inputs

- `_lockerPuzzleHash`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Puzzle hash for locking CATs on Chia.
- `_unlockerPuzzleHash`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Puzzle hash for unlocking CATs on Chia.

Branches and code coverage

Intended branches

- Initializes puzzle hashes for locking and unlocking tokens.
  - ☑ Test coverage

Negative behavior

- Revert if attempting to set puzzle hashes again.
  - ☑ Negative test

Function: `receiveMessage(byte[32], byte[3] _source_chain, byte[32] _source, byte[32][] _contents)`

This function is used to receive messages from the portal contract and process them. This function transfers tokens to the receiver and the transfer tip to the portal. This function is called by the portal contract.

Inputs

- `nonce`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Not used in the function.
- `_source_chain`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Source chain ID.
- `_source`
  - **Control**: Arbitrary.
  - **Constraints**: None.

    - **Impact**: Source puzzle hash (address of sender).
- `_contents`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: The data being relayed; can be (receiver, amount) or (asset contract, receiver and amount)

Branches and code coverage

Intended branches

- Calculate the correct transfer tip.
    - ☑ Test coverage
- Mint the amount of tokens minus the `transferTip`.
    - ☑ Test coverage
- Mint `transferTip` amount of tokens to the portal.
    - ☑ Test coverage

Negative behavior

- Revert if `msg.sender` is not the portal.
    - ☑ Negative test
- Revert if source is not the `lockerPuzzleHash`.
    - ☑ Negative test
- Revert if source chain is not the `otherChain`.
    - ☑ Negative test
- Revert if `transferTip` is less than zero.
    - ☑ Negative test
- Revert if the amount is less than the transfer tip.
    - ☑ Negative test

5. Assessment Results

At the time of our assessment, the reviewed code was deployed to the Ethereum Mainnet, Base Mainnet, and Chia Mainnet.

During our assessment on the scoped warpdotgreen-cli contracts, there were no security vulnerabilities discovered.

5.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.