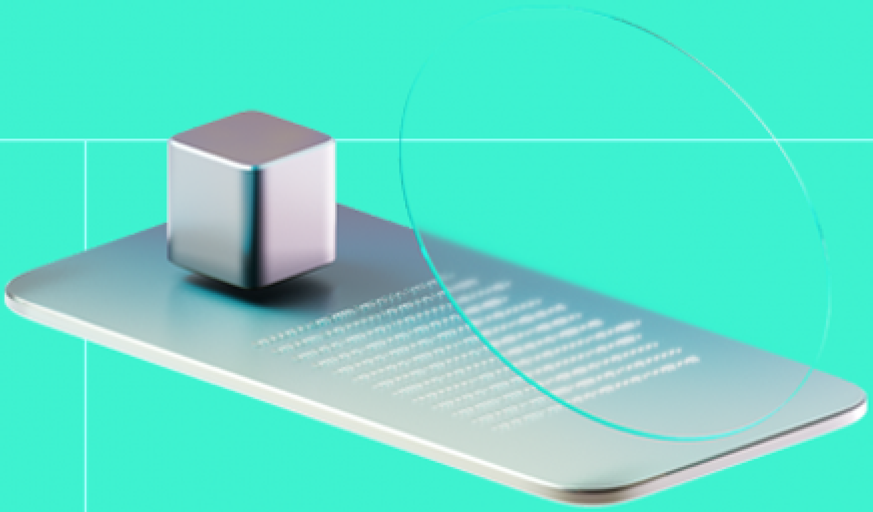
Smart Contract Code Review And Security Analysis Report



Customer: warp.green

Date: 21/05/2024

We express our gratitude to The warp.green Team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

warp.green is a protocol that facilitates the communication of messages across supported blockchains (Chia, Ethereum, and Base) through a trusted set of validators.

Platform: EVM

Language: Solidity

Tags: Bridge; Fungible Token; Permit Token; Signatures; Centralization; Upgradable

Timeline: 14/05/2024 - 21/05/2024

Methodology: https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/warpdotgreen/cli/tree/master/contracts
Commit	dddc8a5

Audit Summary

10/10	10/10	100%	10/10
Security score	Code quality score	Test coverage	Documentation quality score

Total 10/10

The system users should acknowledge all the risks summed up in the risks section of the report

7	4	1	2
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	0
Medium	2
Low	5

Vulnerability	Status
F-2024-2948 - Front-Running Risk Due to Lack of Access Control in initializePuzzleHashes	Mitigated
F-2024-2988 - Potential Fee Bypass in bridgeToChia and bridgeEtherToChia Functions	Mitigated
F-2024-2987 - Incompatibility with Fee-On-Transfer and Rebasing Tokens in ERC20Bridge.sol	Accepted
F-2024-2859 - Inadequate Signature Validation and Nonce Management in receiveMessage	Fixed
F-2024-2950 - Unrestricted messageToll Updates Pose Risk of Unexpected Fee Changes	Fixed
F-2024-2996 - Potential Failures in Ether Transfer Using transfer Function	Fixed
F-2024-3062 - Lack of Minimum Amount Leads to Zero Transfer	Fixed

Document

Name	Smart Contract Code Review and Security Analysis Report for warp.green
Audited By	Ivan Bondar
Approved By	Ataberk Yavuzer
Website	https://warp.green/ & https://docs.warp.green/
Changelog	17/05/2024 - Preliminary Report
	21/05/2024 - Final Report

Table of Contents

System Overview

warp.green is a protocol that facilitates the communication of messages across supported blockchains (Chia, Ethereum, and Base) through a trusted set of validators. Applications can integrate with the protocol to enable cross-chain communication. At launch, two primary applications were introduced: an ERC-20 bridge and a CAT bridge.

The warp.green protocol uses a set of smart contracts deployed on the blockchain to manage the wrapping and unwrapping of assets and the sending and receiving of cross-chain messages.

The files in the scope:

- **ERC20Bridge.sol** - Manages the bridging of ERC-20 tokens between Ethereum and Chia. It handles the wrapping of ERC-20 tokens on Chia and unwrapping them back to the original network.
  - Key Functions:
    - initializePuzzleHashes: Sets the puzzle hashes for burning and minting CATs on Chia.
    - receiveMessage: Receives and processes messages from the warp.green portal.
    - bridgeToChia: Bridges ERC-20 tokens to Chia.
    - bridgeEtherToChia: Bridges native Ether to Chia by wrapping it into milliETH.
    - bridgeToChiaWithPermit: Bridges ERC-20 tokens to Chia using a permit for gas-efficient token approval and transfer.
- **MilliETH.sol** - Implements an ERC-20 token (milliETH) which is equivalent to 1/1000th of one ether. It allows the conversion between ether and milliETH.
  - Key Functions:
    - deposit: Mints milliETH tokens equivalent to the deposited ether value.
    - withdraw: Burns milliETH tokens and returns the equivalent amount of ether.
    - receive: Allows the contract to accept direct ether transfers and mints corresponding milliETH tokens.
- **WrappedCAT.sol** - Manages the wrapping of Chia Asset Tokens (CATs) into ERC-20 tokens on Ethereum, allowing these CATs to be used within the Ethereum ecosystem.
  - Key Functions:
    - initializePuzzleHashes: Sets the puzzle hashes for locking and unlocking CATs on Chia.
    - receiveMessage: Processes messages from the warp.green portal to handle unwrapping of CATs.
    - bridgeBack: Burns Wrapped CAT tokens and sends a message to unlock the original CAT tokens on Chia.
- **Portal.sol** - Handles the sending and receiving of cross-chain messages via a trusted set of validators.
  - Key Functions:
    - sendMessage: Sends a cross-chain message to another blockchain.
    - receiveMessage: Receives and relays a cross-chain message from another blockchain.
    - rescueEther: Allows the owner to transfer ether owned by the contract to specified addresses.
    - rescueAsset: Allows the owner to transfer ERC-20 tokens owned by the contract to specified addresses.
    - updateSigner: Updates the authorization status of a signer (validator).

    - updateSignatureThreshold: Updates the threshold of required signatures for message verification.
    - updateMessageToll: Updates the toll fee required to send messages.
- **IPortal.sol** - Interface defining the functions and events for the warp.green portal contract.
- **IPortalMessageReceiver.sol** - Interface defining the function for receiving messages from the warp.green portal.
- **IWETH.sol** - Interface defining the deposit and withdraw functions for WETH (Wrapped Ether).

Privileged roles

- **Portal.sol**:
  - **Owner** (validator cold key multisig): can manage signers, signature thresholds, message tolls, and assets held by the contract.

Executive Summary

This report presents an in-depth analysis and scoring of the customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

Documentation quality

The total Documentation quality score is **10** out of **10**.

- Functional requirements have some gaps.
  - Basic system description is provided.
  - All roles in the system are described.
  - Use cases are described.
- Technical description is detailed.
  - Run instructions are provided.
  - Technical specification is provided.
  - The NatSpec documentation is sufficient.

Code quality

The total Code quality score is **10** out of **10**.

- The development environment is configured.

Test coverage

Code coverage of the project is **100%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Negative cases coverage is present.
- Interactions by several users are tested.

Security score

Upon auditing, the code was found to contain **0** critical, **0** high, **2** medium, and **5** low severity issues. Out of these, **4** issues have been addressed and resolved, leading to a security score of **10** out of **10**.

All identified issues are detailed in the “Findings” section of this report.

Summary

The comprehensive audit of the customer's smart contract yields an overall score of **10**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- **Dependency on Trusted Validators:** The warp.green protocol relies on a trusted set of validators to verify and relay cross-chain messages. This dependency introduces risks associated with validator behavior and consensus. If validators fail to perform their duties correctly or act maliciously, it could compromise the integrity of the bridge, leading to delays, message loss, or incorrect message relays, ultimately affecting user trust and the protocol's reliability.
- **Administrative Control Over Protocol Parameters:** The protocol's administrative roles, such as the owner of the Portal contract, have the ability to modify key parameters, including message toll fees, signer authorization, and signature thresholds. While these controls are necessary for maintaining and updating the protocol, they also introduce risks of misuse or arbitrary changes that could affect the protocol's stability and user trust. Users should remain vigilant about potential adjustments and the impact on their interactions with the protocol.
- **Risk of Message Toll Adjustments:** The message toll fee required to send messages via the Portal can be updated by the owner. Sudden or significant increases in the toll fee could make it more expensive for users to send cross-chain messages, potentially reducing the protocol's usability and accessibility.
- **Absence of Time-lock Mechanisms for Critical Operations:** The protocol currently does not implement time-lock mechanisms for critical administrative operations, such as updating signers or modifying toll fees. The absence of time-locks increases the risk of rapid and potentially harmful changes being executed without sufficient review or the ability to revert actions.
- **Flexibility and Risk in Contract Upgrades:** The use of upgradable contracts allows the protocol to evolve and address issues promptly. However, this flexibility also introduces risks if the upgrade processes are not properly secured. Unauthorized or malicious upgrades could compromise the protocol's integrity.
- **Dependency on Off-chain Components:** The sendMessage function in Portal.sol is responsible for integrating other Dapps and sending messages from the EVM to the Chia network. The warp.green protocol relies heavily on off-chain components and logic to process these messages. This reliance introduces several risks:
  - Compromise of Off-chain Components: If the off-chain components are compromised, they could lead to unauthorized message relays or message tampering.
  - Malfunction or Failures: Any malfunction or operational failure in the off-chain components could result in message delivery delays, failures, or incorrect message processing.
  - Uncovered Vulnerabilities: Off-chain logic is not part of this audit, which means potential vulnerabilities in the off-chain components remain unidentified and unaddressed.
  - Impact on Integrity and Security: Any issues with the off-chain logic can compromise the overall integrity and security of the warp.green protocol. This could lead to security breaches or loss of user assets.
- **Dynamic Chain Support:** The protocol introduces the ability to dynamically update the list of supported chains. While this provides flexibility in managing and maintaining cross-chain interactions, it also introduces several risks:
  - Temporary Removal of Supported Chains: Chains that were previously supported can be removed from the list of supported chains. This can result in bridged funds being stuck on the bridged chain until support for that chain is reinstated.
  - Impact on User Assets: Users with assets bridged to or from a chain that is temporarily unsupported may face delays in accessing or recovering their funds.

    - Operational Flexibility: This mechanism allows for the suspension of support for chains undergoing maintenance or experiencing issues, providing operational flexibility. However, users should be aware of this risk and stay informed about the status of chain support.
    - User Awareness: Users should be vigilant and regularly check the list of supported chains to ensure their transactions are not affected by dynamic changes in chain support.
- **Solidity Version Compatibility and Cross-Chain Deployment:** The project utilizes Solidity version 0.8.20 or higher, which includes the introduction of the PUSH0 (0×5f) opcode. This opcode is currently supported on the Ethereum mainnet but may not be universally supported across other blockchain networks. Consequently, deploying the contract on chains other than the Ethereum mainnet, such as certain Layer 2 (L2) chains or alternative networks, might lead to compatibility issues or execution errors due to the lack of support for the PUSH0 opcode. In scenarios where deployment on various chains is anticipated, selecting an appropriate Ethereum Virtual Machine (EVM) version that is widely supported across these networks is crucial to avoid potential operational disruptions or deployment failures.

Findings

Vulnerability Details

F-2024-2859 - Inadequate Signature Validation and Nonce Management in receiveMessage - Medium

Description:

The `receiveMessage` function processes cross-chain messages, verifying signatures and ensuring the uniqueness of nonces. However, the function does not verify the `s` value of ECDSA signatures, allowing for two valid `s` values for each signature. This can lead to replay attacks or acceptance of invalid signatures. Additionally, the nonce is checked only after signature verification, leading to potential wasted computational resources if the nonce is already used.

Affected Code:

```
function receiveMessage(
bytes32 _nonce,
bytes3 _source_chain,
bytes32 _source,
address _destination,
bytes32[] calldata _contents,
bytes memory _sigs
) external {
require(_sigs.length == signatureThreshold * 65, "!len");

//..hash generation
address lastSigner = address(0);

for (uint256 i = 0; i < signatureThreshold; i++) {
uint8 v;
bytes32 r;
bytes32 s;

assembly {
let ib := add(mul(65, i), 32)
v := byte(0, mload(add(_sigs, ib)))
r := mload(add(_sigs, add(1, ib)))
s := mload(add(_sigs, add(33, ib)))
}

address signer = ecrecover(messageHash, v, r, s);
require(isSigner[signer], "!signer");
require(signer > lastSigner, "!order");
lastSigner = signer;
}

bytes32 key = keccak256(abi.encodePacked(_source_chain, _nonce));
require(!usedNonces[key], "!nonce");
usedNonces[key] = true;


//..other code
}
```

Exploitation of these vulnerabilities can result in:

- Signature Malleability: The s value of the ECDSA signature is not checked to ensure it is in the lower half of the elliptic curve order, leading to potential replay attacks or acceptance of invalid signatures.

- Inefficient Nonce Checking: The nonce check is performed after signature verification, which can lead to unnecessary computations if the nonce has already been used.
- Lack of EIP-712 Usage: The function does not utilize EIP-712 for structured data hashing and signing, which is a more secure and standardized method for signing messages on Ethereum.

Assets:

- Portal.sol

Status: Fixed

Classification

Impact:	5/5
Likelihood:	3/5
Exploitability:	Semi-Dependent
Complexity:	Medium
Severity:	Medium

Recommendations

Remediation:

- Add Early Nonce Check:
  - Perform the nonce check before any signature verification to prevent unnecessary computations.

```
bytes32 key = keccak256(abi.encodePacked(_source_chain, _nonce));
require(!usedNonces[key], "!nonce");
usedNonces[key] = true;
```

- Implement Signature Malleability Check:
  - Add a check to ensure the `s` value is in the lower half of the curve to mitigate signature malleability attacks. Use OpenZeppelin's ECDSA.sol for handling ECDSA signature operations.
- Adopt EIP-712 Standard:
  - Utilize EIP-712 for structured data hashing and signing to enhance security and standardize message signatures. By adopting EIP-712, the contract ensures that signatures are unique to the specific chain and contract, preventing replay attacks across different chains.

Resolution: (Revised commit: 1af33e4): The issue with the receiveMessage function was fixed by implementing the following changes:

- Early Nonce Check: The nonce is now checked before any signature verification, preventing unnecessary computations if the nonce has already been used.
- Signature Malleability Check: The function now uses OpenZeppelin's ECDSA.recover, which ensures that the s value is in the lower half of the elliptic curve order, mitigating signature malleability attacks.
- EIP-712 Adoption: The function now utilizes EIP-712 for structured data hashing and signing, enhancing security and standardizing message signatures.

F-2024-2988 - Potential Fee Bypass in bridgeToChia and bridgeEtherToChia Functions - Medium

Description:

The `bridgeToChia` function allows users to bridge ERC-20 tokens to the Chia network by specifying the amount of mojos to receive on the Chia network. The `transferTip` is calculated in the `_handleBridging` function as:

```
uint256 transferTip = (_amount * tip) / 10000;
```

If `_amount * tip` is less than `10000`, `transferTip` will be zero, allowing users to bridge tokens without paying transfer tip. This can be more problematic for tokens with high decimals.

Example from `bridgeToChia` function:

```
function bridgeToChia(
address _assetContract,
bytes32 _receiver,
uint256 _mojoAmount // on Chia
) external payable {
require(msg.value == IPortal(portal).messageToll(), "!toll");

_handleBridging(
_assetContract,
true,
_receiver,
_mojoAmount,
msg.value,
10 ** (ERC20Decimals(_assetContract).decimals() - 3)
);
}
```

In the `bridgeEtherToChia` function, the transfer tip can also be zero. For WETH having `18` decimals, the `wethToMojosFactor` is `10 ** 15`. If `amountAfterToll` is 0.333 ETH, the calculated mojos amount is `333`, resulting in a zero tip for a standard tip rate of 30 basis points.

Affected Code:

```
function bridgeEtherToChia(bytes32 _receiver) external payable {
uint256 messageToll = IPortal(portal).messageToll();

uint256 amountAfterToll = msg.value - messageToll;
require(
amountAfterToll >= wethToEthRatio &&
amountAfterToll % wethToEthRatio == 0,
"!amnt"
);

IWETH(iweth).deposit{value: amountAfterToll}();

uint256 wethToMojosFactor = 10 ** (ERC20Decimals(iweth).decimals() -
3);

_handleBridging(
iweth,
false,
_receiver,
amountAfterToll / wethToEthRatio / wethToMojosFactor,
messageToll,
wethToMojosFactor
```

```
    );
}
```

Assets:

- ERC20Bridge.sol

Status: Mitigated

Classification

Impact:	3/5
Likelihood:	4/5
Exploitability:	Independent
Complexity:	Simple
Severity:	Medium

Recommendations

Remediation: To address the issue of zero transfer tips for small amounts, calculate the `transferTip` based on the actual ERC-20 or WETH amount instead of mojos. This ensures that the tip is appropriately scaled and prevents the fee from being bypassed.

Resolution: (Revised commit: e1d41c4): The updated implementation enforces a minimum tip of 1 mojo for bridging to the Chia network, ensuring that even for small amounts, a minimum fee is applied. This approach prevents transactions with zero fees and addresses the issue while maintaining the integrity of the fee structure. As a result, the fee can vary for low mojo amounts, which is considered acceptable by the client.

Evidences

Fee Bypass in bridgeEtherToChia Function

Reproduce:

PoC Steps:

- Setup the Environment:
  - Deploy the Portal and ERC20Bridge contracts.
  - Initialize the Portal contract with appropriate parameters.
- Deploy the Contracts:
  - Deploy the Portal contract and call the initialize function.
  - Deploy the ERC20Bridge contract and initialize puzzle hashes using initializePuzzleHashes.

- Prepare for Testing:
  - Deploy WETH/MilliETH contracts.
  - Ensure the Portal contract's messageToll is set.
- Execute the Test:
  - Bridge a small amount of ETH (e.g., 0.333 ETH) using bridgeEtherToChia.
  - Check the balance of the portal contract before and after the transaction to verify the tip deduction.
  - Confirm that the tip received is zero.
- Verify the Results:
  - Ensure the transaction emits the MessageSent event.
  - Verify that the tip received by the portal contract is zero for small WETH amounts.

PoC Code:

```
describe("bridgeEtherToChia", function () {
it("Should correctly bridge ETH and deduct tips", async function ()
{
const receiverOnChia = ethers.encodeBytes32String("receiverOnChia");

// WETH amount below which the transfer tip is zero for 18 decimals
const ethToSend = ethers.parseEther("0.333");
let balanceBefore = await weth.balanceOf(portal.target);

// Perform the bridge operation
const tx = erc20Bridge.connect(user).bridgeEtherToChia(receiverOnChi
a, { value: ethToSend + messageToll });

// Expect the message to be sent
await expect(tx).to.emit(portal, "MessageSent");

let balanceAfter = await weth.balanceOf(portal.target);
let tipReceived = balanceAfter - balanceBefore;

console.log(`tipReceived for ${wethToken.name} with decimals ${wethT
oken.decimals}: `, tipReceived);

// Assert that the tip received
```

[See more]

Results:

```
ERC20Bridge test cases
ERC20Bridge (WETH=MilliETH;)
bridgeEtherToChia
tipReceived for MilliETH with decimals 3: 999n
✔ Should correctly bridge ETH and deduct tips
ERC20Bridge (WETH=WETHMock;)
bridgeEtherToChia
tipReceived for WETHMock with decimals 18: 0n
1) Should correctly bridge ETH and deduct tips


1 passing (2s)
1 failing

1) ERC20Bridge test cases
ERC20Bridge (WETH=WETHMock;)
bridgeEtherToChia
Should correctly bridge ETH and deduct tips:

Tip should be greater than zero
+ expected - actual
```

Files:	ERC20BridgeNoFee.ts

F-2024-2948 - Front-Running Risk Due to Lack of Access Control in initializePuzzleHashes - Low

Description: The `initializePuzzleHashes` functions in both ERC20Bridge.sol and WrappedCAT.sol lack access control. This vulnerability can result in front-running attacks, where an attacker sets incorrect puzzle hashes, potentially necessitating contract redeployment.

The `initializePuzzleHashes` function initializes puzzle hashes used for locking and unlocking tokens. This function is critical and should be called only once during the contract's deployment transaction. However, the current implementation does not restrict access to this function, allowing anyone to call it and set the puzzle hashes.

Affected Code:

```
function initializePuzzleHashes(
bytes32 _burnPuzzleHash,
bytes32 _mintPuzzleHash
) external {
require(
burnPuzzleHash == bytes32(0) && mintPuzzleHash == bytes32(0),
"nope"
);
. . .
}
```

Without access control, these functions can be front-run by attackers. This would result in setting incorrect puzzle hashes, rendering the contracts ineffective and possibly requiring a complete redeployment to correct the issue.

Assets:

- WrappedCAT.sol
- ERC20Bridge.sol

Status: Mitigated

Classification

Impact: 2/5

Likelihood: 2/5

Exploitability: Independent

Complexity: Simple

Severity: Low

Recommendations

Remediation:	Implement access control to restrict who can call the `initializePuzzleHashes` function. Ensure only authorized addresses, such as the contract owner or a designated admin, can execute this function.
Resolution:	(Revised commit: 1e2695e): The deployment process uses the CreateCall contract and Safe's batch transaction capability to deploy the contract and call `initializePuzzleHashes` in the same transaction. This ensures that the functions cannot be front-run and the puzzle hashes are set correctly. The customer is comfortable with this approach as the risks are mitigated by adhering to the established deployment procedure.

F-2024-2950 - Unrestricted messageToll Updates Pose Risk of Unexpected Fee Changes - Low

Description:

The `messageToll` variable defines the fee required to send a message via the portal and is set by the contract owner. The `updateMessageToll` function allows the toll to be updated without any restrictions on frequency or maximum value, posing a risk if the toll is changed unexpectedly or to an unreasonable amount.

```
uint256 public messageToll;

function updateMessageToll(uint256 _newValue) external onlyOwner {
require(messageToll != _newValue, "!diff");
messageToll = _newValue;
emit MessageTollUpdated(_newValue);
}
```

In ERC20Bridge.sol, the `bridgeEtherToChia` function calculates `amountAfterToll` by subtracting `messageToll` from `msg.value`. If the toll changes unexpectedly, users may receive less than expected, leading to potential loss of funds.

```
function bridgeEtherToChia(bytes32 _receiver) external payable {
uint256 messageToll = IPortal(portal).messageToll();
uint256 amountAfterToll = msg.value - messageToll;
require(
amountAfterToll >= wethToEthRatio &&
amountAfterToll % wethToEthRatio == 0,
"!amnt"
);

IWETH(iweth).deposit{value: amountAfterToll}();

. . .
}
```

While the `bridgeToChia` and `bridgeToChiaWithPermit` functions handle changes to `messageToll` safely by checking if `msg.value` matches `messageToll`, the `bridgeEtherToChia` function does not.

This vulnerability can lead to the following issues:

- Loss of Funds: Users may receive less than expected if the `messageToll` is changed after they send their transaction.
- Unexpected Behavior: Sudden changes to `messageToll` can lead to failed transactions and a poor user experience.

Assets:

- Portal.sol
- ERC20Bridge.sol

Status: Fixed

Classification

Impact:	3/5
Likelihood:	2/5
Exploitability:	Dependent
Complexity:	Simple
Severity:	Low

Recommendations

Remediation: Modify the `bridgeEtherToChia` function to include an additional parameter `_expectedMessageToll`. This parameter will be used to compare with the actual `messageToll`, ensuring users are protected from rapid changes and always know the exact toll they will pay:

```
function bridgeEtherToChia(bytes32 _receiver, uint256 _expectedMessa
geToll) external payable {
uint256 messageToll = IPortal(portal).messageToll();
require(messageToll == _expectedMessageToll, "Unexpected message tol
l");

uint256 amountAfterToll = msg.value - messageToll;
require(
amountAfterToll >= wethToEthRatio &&
amountAfterToll % wethToEthRatio == 0,
"!amnt"
);


. . .
}
```

Resolution: (Revised commit: dd7dcae): The issue with the `bridgeEtherToChia` function was fixed. The new implementation includes a `_maxMessageToll` parameter that ensures the `messageToll` does not exceed the expected amount. This protects users from unexpected changes in the `messageToll`, ensuring accurate and predictable transaction costs.

F-2024-2987 - Incompatibility with Fee-On-Transfer and Rebasing Tokens in ERC20Bridge.sol - Low

Description: The contract assumes that transferring a certain number of tokens results in the exact same number of tokens being received. This assumption fails with fee-on-transfer tokens, which deduct a fee during each transfer, and rebasing tokens, which automatically adjust the token balance. Additionally, tokens without decimals may also cause issues.

Example function from ERC20Bridge.sol:

```
function bridgeToChia(
address _assetContract,
bytes32 _receiver,
uint256 _mojoAmount // on Chia
) external payable {
require(msg.value == IPortal(portal).messageToll(), "!toll");

_handleBridging(
_assetContract,
true,
_receiver,
_mojoAmount,
msg.value,
10 ** (ERC20Decimals(_assetContract).decimals() - 3)
);
}
```

The incompatibility can lead to:

- Reverted Transactions: If the token's behavior does not match the contract's assumptions, transactions may revert.
- Stuck Tokens: For rebasing tokens, the balance may change unexpectedly, leading to tokens being stuck in the contract.
- Imbalances: Fee-on-transfer tokens can cause an imbalance between the amount of tokens locked on the Chia chain and those available on the EVM chain, resulting in inaccurate bridging.

Assets:

- ERC20Bridge.sol

Status: Accepted

Classification

Impact: 2/5

Likelihood: 3/5

Exploitability: Independent

Complexity: Simple

Severity: Low

Recommendations

Remediation:	Implement an allowlist of supported tokens to avoid the issues with unsupported tokens. New tokens can be added to the allowlist after validating that they meet the bridging requirements.
Resolution:	(Revised commit: dd7dcae): The issue regarding the incompatibility with fee-on-transfer and rebasing tokens has been acknowledged and accepted by the client. The ERC20Bridge contract is designed to be immutable, with no owner or whitelist functionality. The client has decided to limit the tokens available on their frontend to non-fee-on-transfer and non-rebasing tokens. Users who bridge unsupported tokens do so at their own risk, typically by bypassing the frontend. Should support for such tokens be required in the future, a separate contract will be deployed to handle those specific cases. Therefore, no changes will be made to the current implementation.

F-2024-2996 - Potential Failures in Ether Transfer Using transfer Function - Low

Description:

The `transfer` function in ERC20Bridge.sol and Portal.sol may fail under certain conditions, particularly when the receiver address is a smart contract with specific Gas requirements. This can lead to failed transactions and unexpected outcomes when transferring Ether.

In ERC20Bridge.sol, the `receiveMessage` function uses `transfer` to send Ether to the receiver:

```
if (assetContract != iweth) {
SafeERC20.safeTransfer(
IERC20(assetContract),
receiver,
amount - transferTip
);
SafeERC20.safeTransfer(IERC20(assetContract), portal, transferTip);
} else {
IWETH(iweth).withdraw(amount);

payable(receiver).transfer((amount - transferTip) * wethToEthRatio);
payable(portal).transfer(transferTip * wethToEthRatio);
}
```

In Portal.sol, the `rescueEther` function also uses `transfer` to send Ether to a list of addresses:

```
function rescueEther(
address[] calldata _receivers,
uint256[] calldata _amounts
) external onlyOwner {
for (uint256 i = 0; i < _receivers.length; i++) {
payable(_receivers[i]).transfer(_amounts[i]);
}
}
```

The use of `transfer` may fail in the following scenarios:

- The receiver address is a smart contract without a `payable` function.
- The receiver address is a smart contract with a `payable fallback` function that uses more than `2300` Gas units.
- The receiver address is a smart contract with a `payable fallback` function that is called through a proxy, raising the call's Gas usage above `2300` units.
- Some multi-signature wallets may require a higher Gas limit than `2300`.

The impact of using `transfer` includes:

- Failed Transactions: Ether transfers to contracts with specific Gas requirements may fail, causing the transaction to revert.
- Inconsistent Behavior: Transactions may work in some cases but fail in others depending on the Gas usage of the receiver's `fallback` function.
- Usability Issues: Users may face difficulties when using the protocol if their wallets or contracts cannot accept Ether via `transfer`.

Assets:	• Portal.sol • ERC20Bridge.sol
Status:	Fixed

Classification

Impact:	2/5
Likelihood:	3/5
Exploitability:	Independent
Complexity:	Simple
Severity:	Low

Recommendations

Remediation: To ensure robust Ether transfers, it is recommended to use one of the following methods:

- Use `.call{value: amount}`:

```
(bool success, ) = receiver.call{value: amount}("");
require(success, "Transfer failed");
```

- Use OpenZeppelin's `Address.sendValue`:

```
import "@openzeppelin/contracts/utils/Address.sol";

Address.sendValue(payable(receiver), amount);
```

Care must be taken to avoid reentrancy vulnerabilities. Consider using ReentrancyGuard or the Checks-Effects-Interactions pattern to mitigate such risks.

Resolution: (Revised commit: 129a017): The issue of potential failures in Ether transfers due to gas limitations was fixed. The implementation now uses OpenZeppelin's Address `sendValue` method to handle Ether transfers. This approach ensures compatibility with smart contracts that require more than 2300 Gas units and prevents transaction failures.

F-2024-3062 - Lack of Minimum Amount Leads to Zero Transfer - Low

Description:

Several functions in WrappedCAT.sol and ERC20Bridge.sol lack minimum amount checks. This can result in transactions with zero amounts, causing inefficiencies, potential errors.

Affected Functions:

- WrappedCAT.sol:
  - `receiveMessage`
  - `bridgeBack`
- ERC20Bridge.sol:
  - `receiveMessage`
  - `bridgeToChia`
  - `bridgeToChiaWithPermit`
  - `bridgeEtherToChia`

Example of Issue in `bridgeEtherToChia` Function:

```
function bridgeEtherToChia(
bytes32 _receiver,
uint256 _maxMessageToll
) external payable {
uint256 messageToll = IPortal(portal).messageToll();
require(messageToll <= _maxMessageToll, "!toll");

uint256 amountAfterToll = msg.value - messageToll;
require(
amountAfterToll >= wethToEthRatio &&
amountAfterToll % wethToEthRatio == 0,
"!amnt"
);

IWETH(iweth).deposit{value: amountAfterToll}();

uint256 wethToMojosFactor = 10 ** (ERC20Decimals(iweth).decimals() -
3);

_handleBridging(
iweth,
false,
_receiver,
amountAfterToll / wethToEthRatio / wethToMojosFactor,
messageToll,
wethToMojosFactor
);
}
```

For WETH with 18 decimals:

- The `wethToEthRatio` is `1`.
- The `wethToMojosFactor` is `10**15`.

Any positive `amountAfterToll` divided by `wethToEthRatio` that is less than `wethToMojosFactor` will produce 0 mojos.

This can lead to:

- Zero Transfers: Transactions with zero amounts are inefficient and unnecessary, leading to gas wastage.

- Unexpected Behavior: Functions that process zero amounts can cause unexpected behavior and potential issues within the contract logic.
- Potential Exploits: Attackers can spam the bridge with zero-value transactions if the message toll and transaction gas price are affordable, causing redundant zero transfers
- Funds loss: potential loss of small ETH amounts that produce 0 mojos.

Assets:

- WrappedCAT.sol
- ERC20Bridge.sol

Status: Fixed

Classification

Impact:	2/5
Likelihood:	3/5
Exploitability:	Independent
Complexity:	Simple
Severity:	Low

Recommendations

Remediation: Ensure that all relevant functions check for a minimum amount greater than zero before proceeding with the transfer or mint operations.

Resolution: (Revised commit: 7001f34): The issue of several functions in WrappedCAT.sol and ERC20Bridge.sol lacking minimum amount checks was fixed. The updated code now includes validation to ensure that the transfer amount and calculated transfer tip are greater than zero before proceeding. This prevents inefficient zero transfers, potential errors, and ensures that transactions are processed correctly.

Observation Details

F-2024-2844 - Missing checks for address(0) - Info

Description:

Several constructors and initializer functions within the Warp Green protocol lack checks for zero addresses.

In ERC20Bridge.sol, the constructor does not validate the `_portal` or `_iweth` addresses:

```
constructor(
uint16 _tip,
address _portal,
address _iweth,
uint64 _wethToEthRatio,
bytes3 _otherChain
) {
tip = _tip;
portal = _portal;
iweth = _iweth;
wethToEthRatio = _wethToEthRatio;
otherChain = _otherChain;
}
```

In Portal.sol, the `initialize` and `updateSigner` functions does not validate addresses of the `_signers` :

```
function updateSigner(address _signer, bool _newValue) external only
Owner {
require(isSigner[_signer] != _newValue, "!diff");
isSigner[_signer] = _newValue;

emit SignerUpdated(_signer, _newValue);
}
```

In WrappedCAT.sol, the constructor does not validate the `_portal` address:

```
constructor(
string memory _name,
string memory _symbol,
address _portal,
uint16 _tip,
uint64 _mojoToTokenRatio,
bytes3 _otherChain
) ERC20(_name, _symbol) ERC20Permit(_name) {
portal = _portal;
tip = _tip;
mojoToTokenRatio = _mojoToTokenRatio;
otherChain = _otherChain;
}
```

Absence of checks for zero addresses can lead to the following issues:

- Contract functionalities may fail if the zero address is used in place of valid addresses.
- The zero address can cause unexpected behavior, disrupting protocol operations.

Assets:

- WrappedCAT.sol
- Portal.sol
- ERC20Bridge.sol

Status: Fixed

Recommendations

Remediation: Include checks to ensure that addresses provided as constructor or initializer parameters are not the zero address.

Resolution: (Revised commit: dfab1f1): The issue of missing zero address checks in the constructors and initializer functions was fixed. The updated implementation now includes validation to ensure that zero addresses are not used. This prevents potential disruptions and ensures proper functionality of the protocol.

F-2024-2845 - The public functions not called by the contract should be declared `external` instead - Info

Description: In Solidity, function visibility is an important aspect that determines how and where a function can be called from. Two commonly used visibilities are `public` and `external`. A `public` function can be called both from other functions inside the same contract and from outside transactions, while an `external` function can only be called from outside the contract. A potential pitfall in smart contract development is the misuse of the `public` keyword for functions that are only meant to be accessed externally. When a function is not used internally within a contract and is only intended for external calls, it should be labeled as `external` rather than `public`.

Affected Code:

```
83: function initializePuzzleHashes(

103: function receiveMessage(

129: function bridgeBack(bytes32 _receiver, uint256 _mojoAmount) pub
lic payable {
```

Assets:

- WrappedCAT.sol

Status: Fixed

Recommendations

Remediation: Declare functions that are not called internally within the contract and are intended for external access as `external` rather than `public`.

Resolution: (Revised commit: 03a1c08): The issue of using the `public` visibility for functions intended to be accessed only externally was fixed by changing the visibility of the affected functions to `external`.

F-2024-2847 - Lack of Upper Bound on tip Parameter in WrappedCAT.sol and ERC20Bridge.sol - Info

Description:

The constructors in WrappedCAT.sol and ERC20Bridge.sol do not impose a limit on the `tip` parameter, which represents the tip percentage paid to the portal. This omission can result in miscalculation of `transferTip`, potentially causing incorrect fund distribution.

The `tip` parameter in both WrappedCAT.sol and ERC20Bridge.sol is set during contract deployment and represents the percentage, in basis points, of the amount to be sent as a tip to the portal. The `tip` is used in the `receiveMessage` function to calculate the `transferTip`. However, no upper limit is enforced on this parameter.

Example from ERC20Bridge.sol constructor:

```
constructor(
uint16 _tip,
address _portal,
address _iweth,
uint64 _wethToEthRatio,
bytes3 _otherChain
) {
tip = _tip;
portal = _portal;
iweth = _iweth;
wethToEthRatio = _wethToEthRatio;
otherChain = _otherChain;
}
```

The `receiveMessage` function calculates `transferTip` as follows:

```
uint256 transferTip = (amount * tip) / 10000;
```

Without a limit, if the `tip` is set too high, the calculation of `transferTip` will yield unreasonable values, potentially exceeding the total amount being transferred.

Assets:

- WrappedCAT.sol
- ERC20Bridge.sol

Status: Fixed

Recommendations

Remediation: Implement a check to ensure that the `tip` parameter does not exceed a reasonable maximum value.

Resolution: (Revised commit: acb1f42): The issue of no limit on the tip parameter in WrappedCAT.sol and ERC20Bridge.sol was fixed by implementing a check to ensure that the `tip` parameter is within a reasonable range. The new check enforces that the `tip` must be greater than 0 and not exceed 1000 basis points (10%).

F-2024-2852 - Missing Validation for Supported Chains - Info

Description: The `sendMessage` and `receiveMessage` functions in the Portal.sol contract lack checks for supported blockchain chain IDs. This omission allows messages to be sent and received from unsupported chains, potentially leading to unexpected behavior.

The `sendMessage` function is responsible for sending cross-chain messages, while the `receiveMessage` function handles incoming messages from other blockchains. Both functions accept chain IDs as parameters but do not verify if these chain IDs are supported by the protocol.

Example from `sendMessage` function:

```
function sendMessage(
bytes3 _destination_chain,
bytes32 _destination,
bytes32[] calldata _contents
) external payable {
require(msg.value == messageToll, "!toll");
ethNonce += 1;

(bool success, ) = block.coinbase.call{value: msg.value}(new bytes(0
));
require(success, "!toll");

emit MessageSent(
bytes32(ethNonce),
msg.sender,
_destination_chain,
_destination,
_contents
);
}
```

The lack of validation for `_destination_chain` in `sendMessage` and `_source_chain` in `receiveMessage` allows interactions with unsupported chains. This can result in misrouted messages, unauthorized interactions or increased difficulty in debugging and tracing.

Assets:

- Portal.sol

Status: Fixed

Recommendations

Remediation: Implement validation checks to ensure that only supported chain IDs are accepted in the `sendMessage` and `receiveMessage` functions. Maintain a list of supported chains and verify against this list.

Resolution: (Revised commit: eb94851): The issue with the `sendMessage` and `receiveMessage` functions lacking checks for supported blockchain chain IDs was fixed. The new implementation includes a `supportedChains` mapping and validation logic to ensure that only messages to and from supported chains are processed. This prevents interactions with unsupported chains and ensures reliable cross-chain communication.

F-2024-2947 - Floating Pragma - Info

Description:	A "floating pragma" in Solidity refers to the practice of using a pragma statement that does not specify a fixed compiler version but instead allows the contract to be compiled with any compatible compiler version. This issue arises when pragma statements like `pragma solidity ^0.8.20;` are used without a specific version number, allowing the contract to be compiled with the latest available compiler version. This can lead to various compatibility and stability issues.
Assets:	• WrappedCAT.sol • Portal.sol • ERC20Bridge.sol • MilliETH.sol • IWETH.sol • IPortalMessageReceiver.sol • IPortal.sol
Status:	Fixed

Recommendations

Remediation:	Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs for the compiler version that is chosen.
Resolution:	(Revised commit: 1e2695e): The issue of using a "floating pragma" was fixed by locking the pragma version to 0.8.23.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details

Repository	https://github.com/warpdotgreen/cli/tree/master/contracts
Commit	dddc8a5f4876ce27ad5078616da074a2472bad24
Whitepaper	
Requirements	https://docs.warp.green; NatSpec
Technical Requirements	https://docs.warp.green; README.md

Contracts in Scope

./contracts/WrappedCAT.sol

./contracts/Portal.sol

./contracts/ERC20Bridge.sol

./contracts/MilliETH.sol

./contracts/interfaces/IWETH.sol

./contracts/interfaces/IPortalMessageReceiver.sol

./contracts/interfaces/IPortal.sol