ChainPort

SMART CONTRACT AUDIT

zokyo

January 3rd 2024 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that this smart contract passed a security audit.

ZOKYO AUDIT SCORING DEFACTOR

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

HYPOTHETICAL SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: -1 point

Starting with a perfect score of 100:
- 0 Critical issues: 0 points deducted
- 1 High issues: 1 resolved = 0 points deducted
- 0 Medium issue: 0 points deducted
- 1 Low issues: 1 resolved = 0 points deducted
- 17 Informational issues: 4 resolved, 9 verified, 4 unresolved or partially resolved = -3 points deducted
- -3 points were deducted because of the raised concerns regarding centralization, dependency on the backend and upgradeability

Thus, 100 - 3 - 3 = 94

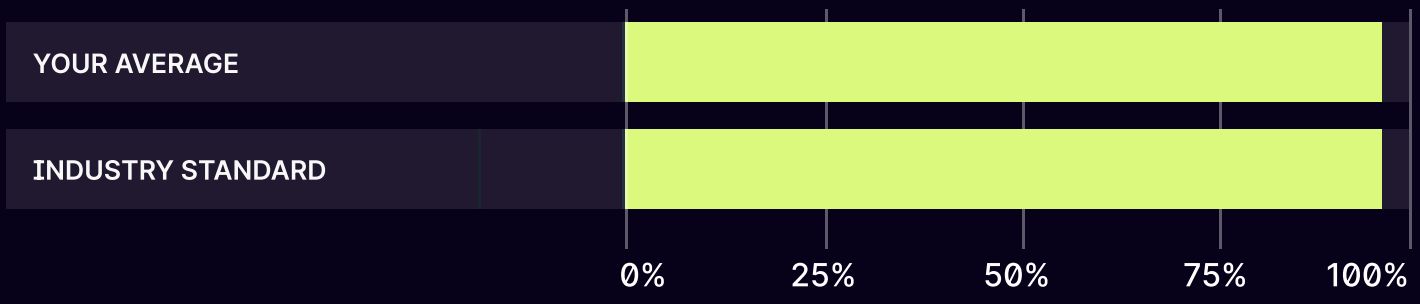
TECHNICAL SUMMARY

This document outlines the overall security of the Chainport smart contracts evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Chainport smart contract codebase for quality, security, and correctness.

Contract Status

Testable Code



Corresponds to industry standards.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the Chainport team put in place a bug bounty program to encourage further active analysis of the smart contract.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Chainport repository:
https://github.com/chainport/smart-contracts

Branch: preprod
Initial commit: 9eef1f421a803a2d8ce30629c0bc48f7f2e39eb6
Final commit: 7593a01b1ed2803ef5e582dbaad551bd8ad5c0b3

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of CHainport smart contracts. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. Part of this work includes writing a test suite using the Hardhat testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.	03	Testing contract logic against common and uncommon attack vectors.
02	Cross-comparison with other, similar smart contracts by industry leaders.	04	Thorough manual review of the codebase line by line.

Executive Summary

The Zokyo Security team has thoroughly audited the Chainport protocol, which serves as a bridge connecting various blockchains. The protocol allows the transfer of tokens between EVM and non-EVM chains and operates with cross-chain swaps. Refer to the Protocol overview section's protocol architecture, roles, and main components. The audit aimed not only to evaluate the overall security of the protocol but also to confirm the reliability of fund storage in smart contracts and verify the accuracy of the logic applied in calculations and token distribution.

The auditor's team conducted a meticulous analysis of all protocol components in several phases:
- system analysis with the decomposition of the protocol to smaller components and actors, resulting in a protocol scheme and understanding of all internal flows;
- manual line-by-line review followed by hypothesis building for detection of edge cases and potential attack vectors;
- review against a list of potential threats and vulnerable areas, including dust attacks, flashloand attacks, fake token attacks, slippage exploiting, etc;
- performing modular tests to verify each function execution and correspondence to the expected logic
- scenario tests for edge cases and complex attacks reliability
- End-to-end test of the system;
- integration tests;
and a list of other checks.

After all audit stages, no critical issues were found. The only high-level issue was connected with the protocol block during the approval of sub-standard tokens (like USDT). All other issues referred to style, logic optimization, and clarification of the implemented business case design choices. From all points of view, the protocol acts as expected and securely performs operations. Though, auditors need to note that:

- The protocol is vulnerable to a single point of failure because of the centralization risk and dependency on the backend services;
- The protocol fails the check against the backdoor because of the upgradeable nature;
- Some best practices are not implemented.

See more explanations in the analysis section. However, the protocol corresponds to a High level of security, implements an impressive set of native tests and the intermediate stage environment.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

HIGH-1 | RESOLVED

Fail because of incompatibility of Standard `approve()` Function with USDT Token.

ArbitrageManager: releaseOrMintAndSwap, swap, swapAndDepositOrBurn, _depositOrBurn.
It is known that the USDT implements an ERC20 version that does not return a `bool` value from the `approve()` function, whereas the expected behavior for ERC20 tokens is to return `true` upon successful execution of `approve()`. This discrepancy leads to the possibility that standard `approve()` calls may revert transactions when interacting with USDT.
The issue is marked as High, as it blocks the work of the contract, and auditors confirmed the case with the appropriate test.

Recommendation:
Implement the `forceApprove()` function from OpenZeppelin's SafeERC20 library, which accommodates such exceptions and ensures correct `approve()` behavior for tokens with non-standard implementations. OR confirm that USDT (or any other sub-standard token implementations) will not be used within the protocol.

Post audit.
The usual approve has been replaced with safeApprove since contracts currently use a solidity version lower than 0.8.0.

LOW-1 | VERIFIED

Unused variables and event.

ChainportSideBridge.sol.
The contract has unused variables maxLpDrainagePercent, mintingProtectionTime, erc20ToBep20Address, functionNameToNonce, maintainerWorkInProgress, lastLpDrainageInitTimestamp, totalMintedFromLastDrainageInit, isSignatureWhitelisted, tokenToThreshold and event SignatureWhitelist, which may lead to confusion and potential issues in the future.

Recommendation:
Remove the unused variables to avoid confusion and potential issues OR add logic in which these variables will be used **OR** verify that they are needed for future functionality.

Post audit.
The Chainport team verified that variables cannot be removed due to the proxy pattern. Though variables are marked as deprecated.

Lack of validation

ChainportMainBridge:
1) initialize() → address _maintainersRegistryAddress, address _chainportCongress, address _signatureValidator;
2) releaseFeesByMaintainer() → address token, address user;
3) releaseTokensByMaintainer() → address token;
4) releaseTokens() → address token, address payable beneficiary;
5) nonEVMDepositTokens() → address token, string calldata receiverAddress;
6) depositTokens() → address token;
7) setPathPauseState() → address token;
8) setFeeCollector() → address feeCollector_;
9) setArbitrageManagerAndLimit() → address _arbManager;

ArbitrageManager:
1) initialize() → addresses _congress, _maintainersRegistry;
2) swapAndDepositOrBurn() → address[] memory path;

ChainportSideBridge:
1) initialize() → address _chainportCongress, address _maintainersRegistry, address _signatureValidator;
2) mintNewToken() → address originalTokenAddress;
3) mintNewTokenFromNonEVM() → string memory originalTokenAddress;
4) burnTokens() → address bridgeToken;
5) nonEVMCrossChainTransfer() → address bridgeToken, string calldata receiverAddress;
6) setSignatureValidator() → address _signatureValidator;
7) setFeeCollector() → feeCollector_;
8) setArbitrageManagerAndLimit() → address _arbManager.

The zero address check is a standard validation to prevent initializing contracts without valid addresses. Add necessary checks to ensure that none of the addresses are equal to the zero address before setting them.

Recommendation:
Add the necessary validation.
Post-audit.
ChainportMainBridge:
1. initialize(): Client has added necessary checks for _maintainersRegistryAddress and _chainportCongress. However, the client stated that the `_signatureValidator` can be set to a zero address on initialization.
2-3. releaseFeesByMaintainer() and releaseTokensByMaintainer(): The client's response indicates that the use of `safeTransfer` precludes the need for an additional zero address check for the token. However, it is noted that the client's response did not address the absence of validation for the `user` parameter within the `releaseFeesByMaintainer()` function. While the function is expected to be called by a maintainer, the lack of validation introduces the risk of inadvertently passing a zero address. This omission could result in incorrect event logging, where the `user` is used exclusively for events and should ideally be validated to prevent any zero address from being recorded
4-7. releaseTokens(), nonEVMDepositTokens(), depositTokens(), and setPathPauseState(): The client trusts the `safeTransfer` function to handle checks against the token being a zero address. Also, beneficiary is validated by the signer and the receiver address is validated by its length, which might be sufficient given the design.
8. setFeeCollector(): The client prefers the flexibility to disable a role by setting it to zero, which is countered by checks in other functions against the feeCollector being a zero address.
9. setArbitrageManagerAndLimit(): As with the fee collector, the client wants the ability to set `_arbManager` to zero for disabling purposes.

ArbitrageManager:
1. initialize(): Checks for `_congress` and `_maintainersRegistry` are accounted for in middleware initialization according to the client. This is acceptable provided that the middleware effectively enforces these checks.
2. `swapAndDepositOrBurn()`: The client relies on Uniswap contract reverts for path validation and does not find additional checks necessary.

ChainportSideBridge:
1. initialize(): The client has ensured checks for _chainportCongress and _maintainersRegistry in middleware but not for _signatureValidator. It remains optional for the same reasons mentioned in the ChainportMainBridge.
2-3. mintNewToken() and mintNewTokenFromNonEVM(): The client sees no issue with minting a zero address, stating it would not harm the project. While this might not have immediate detrimental effects, preventive measures or validations can be beneficial.
4. burnTokens(): The client thinks that burning with a zero address results in a revert, but due to the absence of explicit error messaging, documentation and caution are advised.
5. nonEVMCrossChainTransfer(): The same response is given as in nonEVMDepositTokens on the main bridge, relying on other checks.
6. setSignatureValidator(): Necessary checks have been added.

INFORMATIONAL-2 | ACKNOWLEDGED

Inaccurate version pragma.

`pragma solidity` ^0.6.12 is used in contracts. It is recommended to use the latest version of Solidity and specify the exact pragma. Older solidity versions may contain bugs and vulnerabilities, and be less optimized in terms of gas.

Recommendation:
Specify the latest version of Solidity in the pragma statement.
Post-audit.
The latest version of Solidity in the pragma statement wasn't specified. Still, the team verified that they plan a full architecture upgrade in the future, including custom errors and the latest Solidity version.

INFORMATIONAL-3 | ACKNOWLEDGED

Custom errors should be used.

Starting from the 0.8.4 version of Solidity, it is recommended to use custom errors instead of storing error message strings in storage and using “require” statements. Using custom errors is more efficient regarding gas spending and increases code readability.

Recommendation:
Use custom errors.
Post-audit.
Custom errors aren’t in use now, but the team verified that they plan a full architecture upgrade in the future, including custom errors and the latest solidity version.

INFORMATIONAL-4 | VERIFIED

Inconsistent require statements.

The following functions have require statements without error messages:
ChainportMainBridge.sol:
1) setAddressesWhitelistState() → require(addresses[i] != address(0));
2) setFeeCollector() → require(feeCollector != feeCollector_);
3) withdrawGasFee() → require(success);
4) setArbitrageManagerAndLimit() → require(_arbManagerReleaseLimitPercentB <= 10000 && _arbManagerReleaseLimitPercentTS <= 10000);

ChainportSideBridge.sol:
1) setFeeCollector() → require(feeCollector != feeCollector_);
2) withdrawGasFee() → require(success);
3) setArbitrageManagerAndLimit() → require(_arbManagerMintLimitPercentTS <= 10000).

ArbitrageManager.sol:
1) initialize() →
   a) require(_mainBridge != address(0)),
   b) require(_sideBridge != address(0)),
   c) require(_router != address(0) && routerId[_router] == 0),
   d) require(_operator != address(0) && operatorId[_operator] == 0),
   e) require(_whitelist != address(0) && whitelistsId[_whitelist] == 0);
2) getRouterById() → require(id != 0);
3) getOperatorById() → require(id != 0);
4) getWhitelistById() → require(id != 0);
5) withdraw() → require(tokens[i].balanceOf(address(this)) >= amounts[i] && amounts[i] > 0).

Recommendation:
Consistency in error handling improves the comprehensibility and debuggability of the code. It is recommended to either add descriptive error messages to all "require" statements or follow the best practices suggested in the issue ` Custom errors should be used,` adopt the use of custom errors, which provides gas optimization and improves the developers' and users' ability to understand the failure reason. This approach can enhance the overall security and usability of the contracts.
Post-audit.
The Chainport team verified that the mentioned function will be called by authorities and thus does not need additional bytes spent for messages due to the short troubleshooting process.

INFORMATIONAL-5 | VERIFIED

Inheritance version is different.

ChainportSideBridge and ChainportMainBridge are found to inherit from ChainportMiddleware, while ArbitrageManager inherits from an apparently more recent version, ChainportMiddlewareV3.

Recommendation:
Verify that the use of ChainportMiddleware and ChainportMiddlewareV3 aligns with project requirements.
Post-audit.
Verified by the Chainport team.

INFORMATIONAL-6 | RESOLVED

Lack of events

1) ChainportSideBridge.sol: setSignatureValidator, setOfficialNetworkId, setMintingFreezeState, setTokenProxyAdmin, setTokenLogic, setFeeCollector.
2) ChainportMainBridge.sol: setOfficialNetworkId, setFeeCollector.
In order to keep track of historical changes of storage variables, it is recommended to emit events on every change in the functions that modify the storage.

Recommendation:
Emit events in all functions where state changes to reflect important changes in contract.
Post-audit.
Events are emitted now.

Missing Transaction Reverts for Zero Address Checks.

ChainportSideBridge.sol: _setTokenProxyAdmin, _setTokenLogic.
The functions _setTokenProxyAdmin() and _setTokenLogic() contain checks to prevent setting their respective addresses to the zero address. However, these checks are currently silent – that is, if a zero address is provided as an argument, the function will simply not update the state, but the transaction will not revert and will appear successful. This behavior might lead to a misunderstanding, where calling functions believe that they have successfully set a new address when, in fact, no update has occurred due to providing an invalid zero address.
Ensuring that transactions revert with an informative error message when critical checks fail is a fundamental security best practice in smart contract development. If the function's intended behavior is to disallow a zero address, it should be enforced with a revert statement that clearly indicates why the transaction failed. This ensures transparency and helps prevent potential errors during contract interactions.

Recommendation:
Update the functions to explicitly revert if the input argument is the zero address.

Post audit.
The team verified that this behavior is intentional as the initialization flow is made to cross the roles of maintainer and congress authorities, and there is an initialization function so that initially, the maintainer can do it but may lack an address, so it is possible to set only one address and leave the other one zero.

Unused Role.

The address of fundManager is declared within the ChainportMainBridge contract, with functions provided to set and update its value. However, no active functions or codes within the contract demonstrate any interaction by the fund manager. This suggests that the fundManager currently does not have a defined purpose or functionality.
Leaving the fundManager without explicit use in the contract can lead to confusion regarding the contract's capabilities and purpose.

Recommendation:

Verify the need for the fundManager within the project:
The fundManager is intended for future use **OR** if it is an unnecessary part of the contract with no planned use, consider removing the related functions to simplify contract management and mitigate any associated risks.

Post-audit.

The functionality including fundManager was removed, the variable remained but marked as deprecated.

Inconsistent Error Reporting in Require Statements.

Throughout the provided smart contracts ChainportSideBridge and ChainportMainBridge, there is an observed inconsistency in the format of error messages used within `require` statements. Some contracts use short, code-based error messages like "E33," whereas others use descriptive text such as "Only maintainer can call this function." While both methods serve to halt execution when conditions are unmet, the inconsistency may lead to confusion and could potentially hinder the debugging and maintenance process.

Recommendation:

Establish a consistent error reporting convention throughout the smart contract codebase. Below are steps to improve error messaging consistency:
1. Use of custom error types introduced in Solidity 0.8.4, which can provide gas efficiency alongside descriptive power.
2. If retaining code-based error messaging, maintain a detailed and easily accessible documentation or a legend within the codebase to decode error messages.
3. If descriptive text messages are preferred, ensure they are well-crafted to be informative yet concise, adequately describing the reason for transaction failure.

Post-audit.

The team verified that they will strive for the convention to apply rules of errors. That will be either via custom errors once they increase the solidity version or via short naming like "AB01" which they documented/linked with proper messages.

Violations of Solidity Style Guide Conventions - confusion naming.

1. Variable naming inconsistencies: Presence of variables like `value`, `_value`, and `value_` within the same codebase which could cause confusion.
More specifically:
a) In ChainportSideBridge some functions have parameters in the form of _value, while other functions do not. Functions with parameters in the form of _value:
setArbitrageManagerAndLimit, setBlacklistRegistry, initializeTokenProxyAdminAndLogic, setTokenLogic, setTokenProxyAdmin, setSignatureValidator, setAssetFreezeState → bool _isFrozen, initialize.

b) Same in ArbitrageManager with: initialize.

c) Same in ChainportMainBridge: initialize, setSignatureValidator, setAssetFreezeState → bool _isFrozen, setBlacklistRegistry, setArbitrageManagerAndLimit.

d) There are parameters of the form _value, and value, and value_ in ChainportMiddlewareV3:
event CongressSet(address indexed _chainportCongress) and
event MaintainersRegistrySet(address indexed _maintainersRegistry) while event BlacklistRegistrySet(address indexed blacklistRegistry);
__Chainport_Middleware_V3_init() → chainportCongress_, maintainersRegistry_;
setChainportCongress, setMaintainersRegistry, setBlacklistRegistry, _setChainportCongress, _setMaintainersRegistry, _setBlacklistRegistry → all have parameters of the form value_.

Recommendation:

Align variable naming with the style guide, ensuring that parameters are named clearly without prefix or postfix underscores unless indicating scope as per the style guide.

Post-audit.

In terms of variable naming inconsistencies, the team is looking forward to sorting this out in the near future with part of the inconsistencies resolved via: https://github.com/chainport/smart-contracts/commit/8fe461c2c8de5982ad9b048dad712b58d3647d0a

INFORMATIONAL-10.2 | RESOLVED

Violations of Solidity Style Guide Conventions - magic numbers.

2. Magic numbers usage. Hard-coded constants like:
a) ChainportSideBridge: setArbitrageManagerAndLimit() → `10000`, preMintThreshold() → `10000`;
b) ChainportMainBridge: setArbitrageManagerAndLimit() → `10000`, `10000`;
preReleaseThreshold() → `10000`, `10000`.
are not self-explanatory or defined as named constants, making it difficult to understand their meaning or how they were derived.

Recommendation:

Replace magic numbers with named constants that provide context and make the code more maintainable and understandable.

INFORMATIONAL-10.3 | RESOLVED

Violations of Solidity Style Guide Conventions - typos.

3. A typo in the case of a reverted transaction
ChainportMainBridge:closePreRelease() has require that checks amount for a specific nonce is correct and in case of falling this statement error says:
"Amount is correct" which is opposite and can cause misunderstanding.

Recommendation:

Make sure that the reverted message is correct.

Post-audit.
In terms of variable naming inconsistencies, the team is looking forward to sorting this out in the near future. In terms of magic numbers usage, magic numbers were replaced with named constants. In terms of typo, the reverted message was fixed.

INFORMATIONAL-11 | VERIFIED

Use of Zero Address in Blacklist Check.

ChainportSideBridge: _burnTokens().
The function _burnTokens() uses the modifier onlyNonBlacklistedAddress(address(0×0)), passing the zero address rather than the transaction initiator msg.sender. This could lead to incorrect behavior of blacklist enforcement.

Recommendation:
Verify the use of onlyNonBlacklistedAddress in _burnTokens() to ensure it correctly enforcing blacklist controls on the correct participant in the transaction.

Post-audit.
The team verified that enforcement is correct as the mentioned modifier by default takes into consideration both tx.origin and msg.sender; the following argument is for an additional address to check, and in case of 'burn,', there is no recipient or other address to check the blacklist status.

INFORMATIONAL-12 | VERIFIED

Dynamic Gas Fees Adjustment.

The contracts reference a set gas fee value, which may become outdated as network gas prices fluctuate. This could lead to the bridge not covering its operational costs if the predefined fees are lower than the actual gas prices needed for transactions.

Recommendation:
Verify if the current method and frequency for updating gas fees (gasFeesPerNetwork[networkId]) is effective.

Post-audit.
The Chainport team verified this is an expected behavior. Still the auditors team raises a warning that such an approach may lead to losses in operational costs.

Backend Dependency and Centralization Concerns.

The overall functionality of the bridge heavily relies on backend processes, even though numerous checks are present within the contract, equating to a centralized bridge operation. It is worth discussing the extent to which centralized components are used and the potential implications for the system's decentralization and security stance.

The issue is marked as info, referring to the business logic decision and core architecture solution. However, it should be listed in the report due to the potential risk it may pose.

Recommendation:
Verify the role of backend services within the bridge infrastructure and evaluate the necessity of each centralized component.

Post-audit.
The Chainport team verified that this is an expected behavior. Still, the auditor's team warns that such an approach may create a risk of a single point of failure for the protocol.

Unclear Commission Parameters Source.

ChainportSideBridge, ChainportMainBridge.
It has been observed that commission fees are applied within various functions. However, the source and calculation method for these commission values is not explicitly defined within the contracts. It appears that the backend system may be inputting these values. This lack of clarity could result in transparency issues and potential inconsistencies in fee applications.

The issue is marked as info, referring to the business logic decision and core architecture solution. However, it should be listed in the report due to the potential risk it may pose.

Recommendation:
Verify how commission values are derived.

Post-audit.
The Chainport team verified that this is an expected behavior. Still, the auditor's team warns that such an approach may create a risk of a single point of failure for the protocol because of dependency on another component.

Potential Overreach of Modifier.

1. ChainportSideBridge: _crossChainTransfer.
The ChainportSideBridge contract's function _crossChainTransfer employs the modifier isPathNotPaused(bridgeToken, "crossChainTransfer") to check whether a transfer path is currently paused. However, this modifier is used as a precondition for both `crossChainTransfer` and `nonEVMCrossChainTransfer` functions. If the `crossChainTransfer` pathway is paused, it inadvertently impacts the `nonEVMCrossChainTransfer` functionality as well, given that the same `bridgeToken` is used in both functions. This could unintentionally restrict token transfers to non-EVM chains.

2. ChainportMainBridge: _depositTokens.
The ChainportMainBridge contract's function _depositTokens employs the modifier isPathNotPaused(token, "depositTokens") to check whether a transfer path is currently paused. However, this modifier is used as a precondition for both `depositTokens` and `nonEVMDepositTokens` functions. If the `depositTokens` pathway is paused, it inadvertently impacts the `nonEVMDepositTokens` functionality as well, given that the same `token` is used in both functions. This could unintentionally restrict token transfers to non-EVM chains.

Recommendation:
Verify the intention behind the shared pause functionality. If these functions should indeed have independent pause controls, consider introducing separate pause states for `depositTokens` and `nonEVMDepositTokens`.

Post-audit.

The team verified that the behavior of these modifiers is intentional.

Potential withdrawal gas fee to zero address.

ChainportMainBridge: withdrawGasFee.
The ChainMainBridge contract's function withdrawGasFee implements the withdrawal of the whole balance of the contract to the fee collector address. The function `setFeeCollector` has no requirement for checking that the new `feeCollector` can not be a zero address. However, there is no check that the `feeCollector` address is set, and it may cause a case when the whole balance could be sent to a zero address.

Recommendation:
Add check that the `feeCollector` address is set to non-zero address to prevent withdrawal to zero address.

Post-audit.
Check was added along with gas optimization.

Unused Functionality.

ChainportMiddlewareV3: modifiers onlyMaintainer(), onlyNotBlacklisted(), onlyNotBlacklistedWithUser().
The ArbitrageManager contract inherits from ChainportMiddlewareV3, which contains certain functionalities and modifiers. It has been observed that not all inherited capabilities of ChainportMiddlewareV3 are utilized within the ArbitrageManager. Given that ChainportMiddlewareV3 is solely inherited by ArbitrageManager and no other contracts, this leads to the presence of redundant code paths and inherited properties that are not applicable to the ArbitrageManager's operations.

Recommendation:
Verify the necessity of these modifiers for ArbitrageManager. If certain parts of ChainportMiddlewareV3 are not needed by ArbitrageManager, consider refactoring to enhance clarity.

Post-audit.
Team verified that described modifiers might be used in the future by ArbitrageManager and will either way be used by the other contracts which will inherit ChainportMiddlewareV3.

	HordETHStakingManager.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Fail

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security

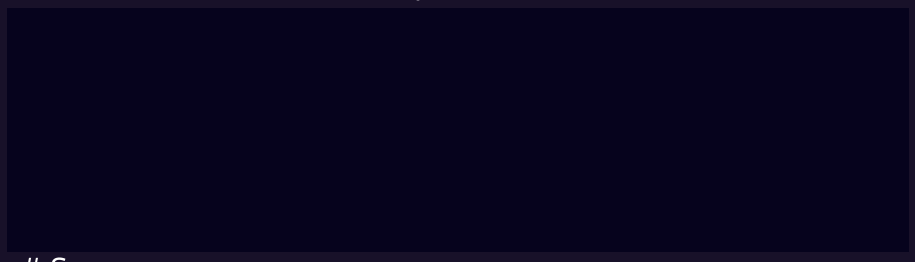
As a part of our work assisting Chainport in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Hardhat testing framework.

The tests were based on the functionality of the code, as well as a review of the Chainport contracts requirements for details about issuance amounts and how the system handles these.

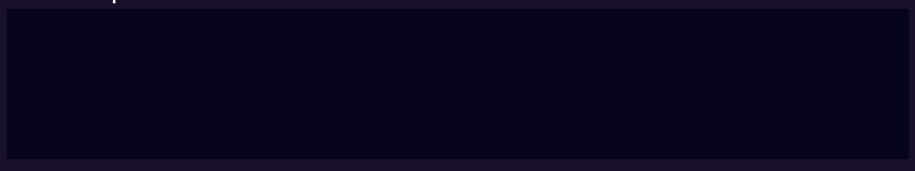
Scope: ArbitrageManager.sol
ArbitrageManager/ChainportMiddlewareV3

Initialize

Release or mint and swap



Swap



Swap and deposit or burn

Deposit or burn

Add routers

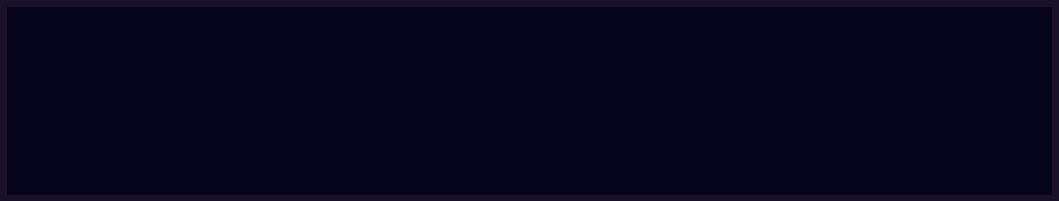
Remove routers

Get routers

Arbitrage balances

Arbitrage balances specification

Add operators



Remove operators

```
# Get operators
```

```
# Add whitelists
```

```
# Remove whitelists
```

```
# Get whitelists
```

```
# Withdraw
```

```
# Chainport middleware V3
```

Scope: ChainportSideBridge.sol
ChainportSideBridge

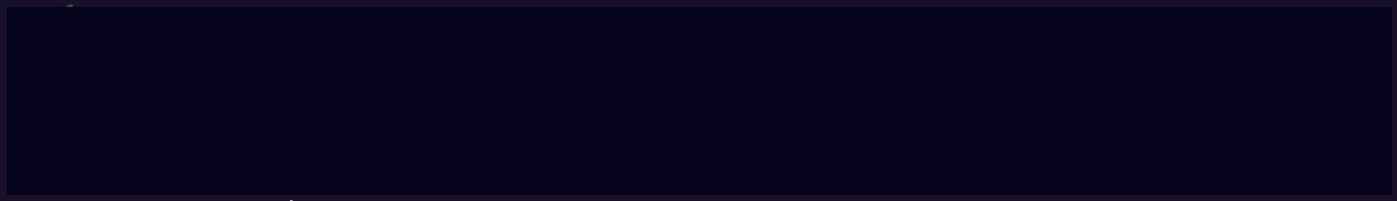
```
# Deployment

# Initialize

# Freeze bridge

# Set bridge freeze state

# Mint new token

# Mint Tokens
```

Mint Fees

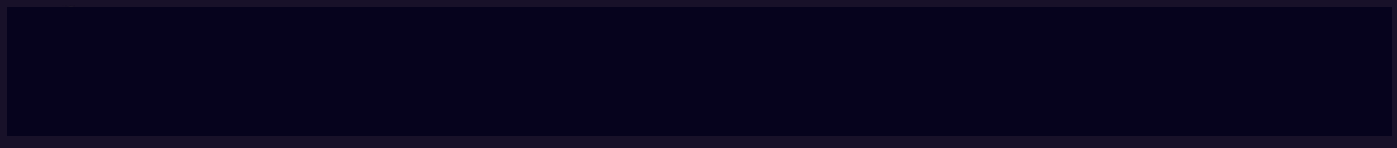
Burn tokens

Non-EVM cross-chain transfer

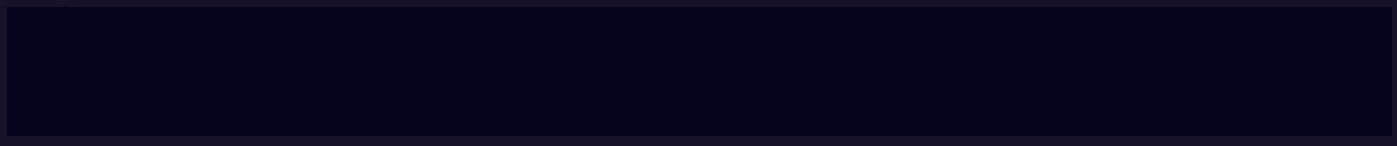
Cross-chain transfer



Activate networks



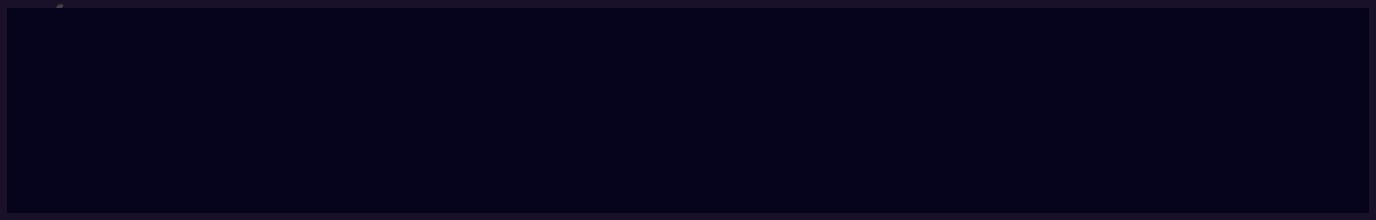
Set network activity state



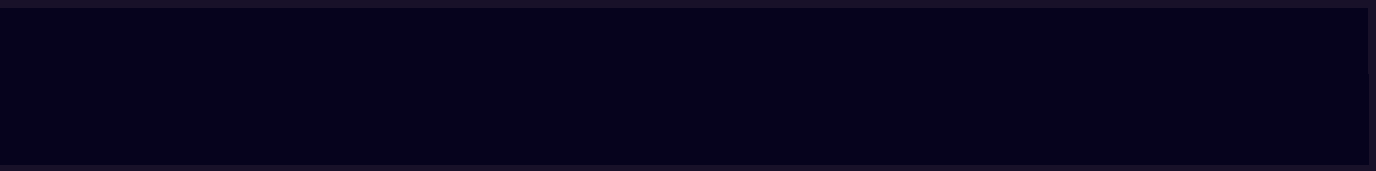
```
# Set asset freeze state

# Freeze assets by Maintainer

# Set path pause state

# Set signature validator

# Set official network Id

# Set minting freeze state

# Minting and asset unfreeze

# Set token proxy admin

# Set token logic

# Initialize token proxy admin and logic

# Pause bridge mintable token
```

```
# Block address on bridge mintable token

# Set fee collector

# Is fee metadata used

# Is msg hash used

# Set gas fees and address length

# Withdraw gas fee

# Set blacklist registry

# Set arbitrage manager and limit

# PreMint

# Burn to close preMint or release
```

Close preMint

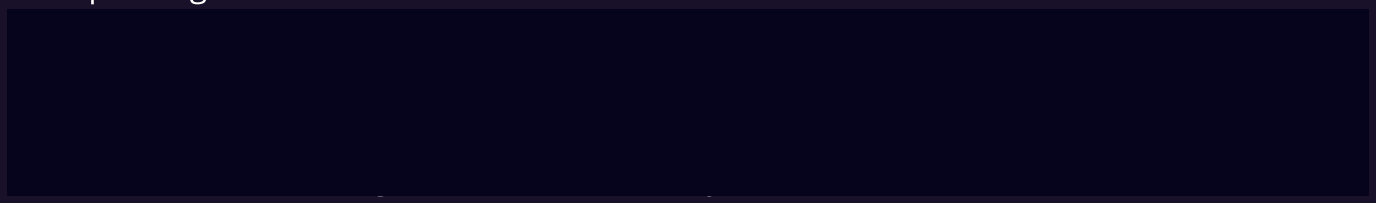


Scope: ChainportMainBridge.sol
ChainportMainBridge



Activate networks

Depositing tokens



Non-EVM depositing tokens

Release tokens

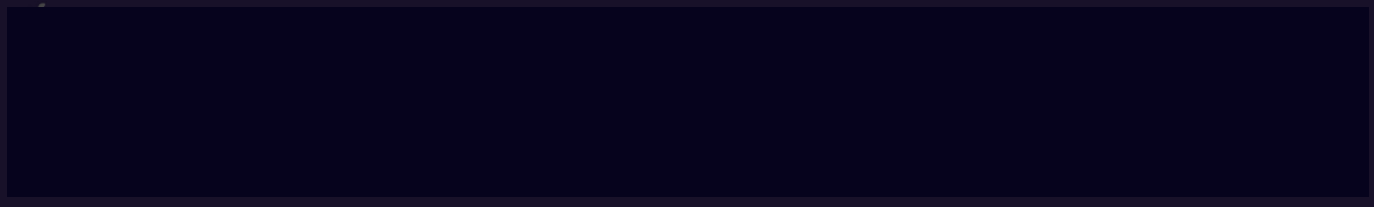
Set new arbitrage manager and limit

PreRelease tokens

Deposit to close premint

Close pre release

Whitelist controller



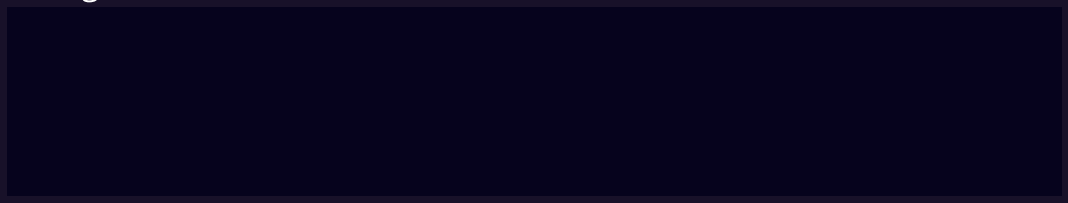
Release tokens by maintainer

Fee collector

Release fees by maintainer

Freeze functions

bridge

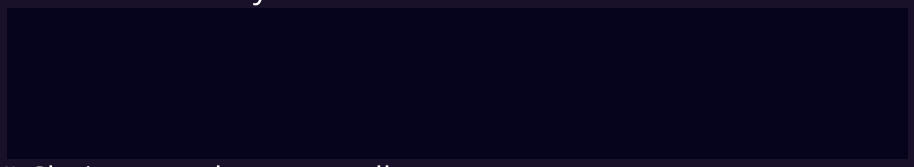


assets

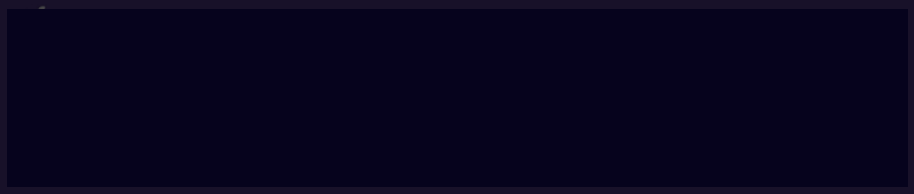
Withdraw gas fee

Official network id setter

Network Activity State setter



Chainport token controller



Signature validator setter

Additional scenarios

Deployment

✓ Should initialize correctly (160ms)

Transfer between bridges

✓ Transfer from main bridge to side bridge + crossChainTransfer and releaseTokens (822ms)

✓ Transfer from main bridge to side bridge + crossChainTransfer and releaseTokens shouldn't work because of invalid signatures (616ms)

Swap

✓ Shouldn't revert approve on USDT during deposit-swap flow (3752ms)

✓ Deposit tokens -> swap -> deposit new token after swap (1921ms)

✓ Deposit tokens -> mint bridge tokens -> preMint -> swap (2436ms)

✓ Transfer from main bridge to side bridge -> nonEvmChainTransfer (from evm chain to non-evm) (362ms)

✓ Transfer from non-evm chain -> releaseTokens (79ms)

FILE	% STMTS	% BRANCH	% FUNCS
ChainportMainBridge.sol	100%	92.47%	100.00%
ChainportSideBridge.sol	100%	95.65%	100.00%
ArbitrageManager.sol	100%	96.43%	100.00%
ChainportMiddlewareV3.sol	71.05%	76.32%	72.73%

We are grateful for the opportunity to work with the Chainport team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the Chainport team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo