QuillAudits

AUDIT REPORT

March 2025

For

SUPRA

Table of Content

Table of Content 02

Executive Summary 04

Number of Issues per Severity 09

Checked Vulnerabilities 10

Techniques & Methods 12

Types of Severity 14

Types of Issues 15

Medium Severity Issues 16

1. Removing the registered token is not possible 16

2. Incorrect token argument passed in getMinLimits() and getMaxLimits() 17

Low Severity Issues 18

1. Withdraw function lacks isNotPaused modifier 18

2. Failed to check return value. 19

3. Only transferFrom can be used 20

4. Redundant allowance check 21

5. Store and reuse the fee 22

6. The unsupported chain ID can be used by user 23

7. Add validation for toChainId in sendNative() 24

8. Centralization concern 25

Informational Severity Issues 26

1. Floating pragma 26

2. Block reorg can cause state inconsistencies 27

3. An attacker can DOS the system if the source chain has considerably less gas fee 28

4. Remove the else statements for returning the "false" value 29

5. Check if the indexed parameters are required. 30

6. updateTokenFee() allows 0 address as token address 31

7. Unused tokenBridgeContract address 32

8. The contract lacks the functionality to transfer the ownership 33

9. Check the expected functionality 34

10. postMessage() external function can be maliciously used 35

Functional Tests 36

Closing Summary & Disclaimer 38

Executive Summary

Project Name	Supra
Overview	Hypernova: This is the core contract of the protocol. This contract's duty is to emit necessary data that is required for bridging the tokens. This contract contains only 2 external (non-view) functions postMessage(), upgradeImplementation(). None of the above mentioned functions can be called by end-users. postMessage() can only be called by tokenBridge contract and upgadeImplementation() can only be called by an Admin address which is stored in the contract at the time of initialization. Bridge base fee, admin address and token bridge contract address are stored in the contract at the time of initialization. RelayOperator: This contract is used by relayer to update the gas price of the destination chain (Supra) and token bridge contract fetches the latest gas price from here. Only 2 external (non-view) functions updateGasPrice(). upgradeImplementation(). None of the functions can be called by end-users. UpdateGasPrice() can only be called by relayer, whose address is stored at the time of initialization. (upgradeImplementation() same as in Hypernova) admin address, token bridge contract address and relayer address gets stored in the contract at the time of initialization.

Executive Summary

TokenBridge: This contract is used by the end users to bridge their tokens (only registered tokens) and native currency (eg. ETH) to Supra. Users' tokens get locked in the token bridge contract and postMessage function of hypernova is called which emits an event with all the necessary data required by the destination contracts to complete the transaction. Before bridging, the token needs to be registered. Only admin / owner has the powers to register new token, pause/unpause the bridge and upgrade the implementation.
For end-users, there are only 2 functions sendTokens() and sendNative(). First one for tokens and second one for native currency. Other external functions are for admin / owner.

Token Vault & Vault Implementation: The contract implements a token vault system using an upgradeable proxy pattern. The TokenVault.sol serves as a proxy contract that inherits from OpenZeppelin's ERC1967Proxy, allowing for upgradeability of the vault's implementation. The main business logic resides in VaultImplementation.sol, which is a bridge-controlled token vault that can handle both ERC20 and native tokens (ETH).

Audit Scope

The Scope of the audit is to analyse the code security, Quality and Correctness of Supra Hyper Nova Contracts.

Blockchains

Ethereum , Supra

Executive Summary

Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Source Code	https://github.com/Entropy-Foundation/supra-interoperability-solutions/tree/hn-milestone1/hypernova/eth-supra/eth_contracts
Branch	Vault
Commit hash	35497dcb27620707cf1a977f4d235070d2b05789
Contracts In-Scope	hypernova/eth-supra/eth_contracts/contracts/hypernova-core/Hypernova.sol hypernova/eth-supra/eth_contracts/contracts/hypernova-core/implementations/HypernovaImplementation.sol hypernova/eth-supra/eth_contracts/contracts/hypernova-core/implementations/State.sol hypernova/eth-supra/eth_contracts/contracts/hypernova-core/implementations/Helpers.sol hypernova/eth-supra/eth_contracts/contracts/hypernova-core/implementations/Errors.sol hypernova/eth-supra/eth_contracts/contracts/relay-operator/RelayOperator.sol hypernova/eth-supra/eth_contracts/contracts/relay-operator/implementations/RelayOperatorImpl.sol

Executive Summary

hypernova/eth-supra/eth_contracts/contracts/relay-operator/implementations/State.sol

hypernova/eth-supra/eth_contracts/contracts/relay-operator/implementations/Helpers.sol

hypernova/eth-supra/eth_contracts/contracts/relay-operator/implementations/Errors.sol
hypernova/eth-supra/eth_contracts/contracts/tokenBridge-service/TokenBridge.sol

hypernova/eth-supra/eth_contracts/contracts/tokenBridge-service/implementations/TokenBridgeImplementation.sol

hypernova/eth-supra/eth_contracts/contracts/tokenBridge-service/implementations/State.sol

hypernova/eth-supra/eth_contracts/contracts/tokenBridge-service/implementations/Helpers.sol

hypernova/eth-supra/eth_contracts/contracts/tokenBridge-service/implementations/Errors.sol

hypernova/eth-supra/eth_contracts/contracts/token-vault/TokenVault.sol

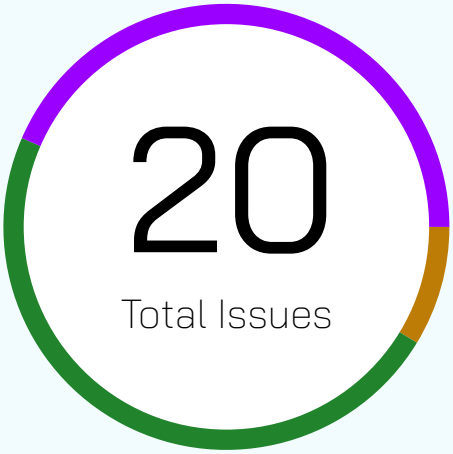
hypernova/eth-supra/eth_contracts/contracts/token-vault/implementations/VaultImplementation.sol

hypernova/eth-supra/eth_contracts/contracts/token-vault/implementations/State.sol

Executive Summary

	hypernova/eth-supra/eth_contracts/contracts/token-vault/implementations/Helpers.sol hypernova/eth-supra/eth_contracts/contracts/token-vault/implementations/Errors.sol hypernova/eth-supra/eth_contracts/contracts/token-vault/implementations/State.sol
Updated Code Received	24th February 2025
Review 2	3rd March 2025 - 4th March 2025
Fixed In	Branch: eth_contractsV2 Commit hash: ad7e6431515d0ea5b3c6cfca61bf7b7fbd90eecd
Note	Supra team made a different change in updateTokenFee function Post Audit,which was out of scope.

Number of Issues per Severity



High	0 (0.00%)
Medium	2 (10.00%)
Low	8(40.00%)
Informational	10 (50.00%)

Severity

Issues	High	Medium	Low	Informational
Open	0	0	0	0
Resolved	0	**2**	**8**	**7**
Acknowledged	0	0	0	**3**
Partially Resolved	0	0	0	0

Checked Vulnerabilities

- ☑ Access Management
- ☑ Arbitrary write to storage
- ☑ Centralization of control
- ☑ Ether theft
- ☑ Improper or missing events
- ☑ Logical issues and flaws
- ☑ Arithmetic Computations Correctness
- ☑ Race conditions/front running
- ☑ SWC Registry
- ☑ Re-entrancy
- ☑ Timestamp Dependence
- ☑ Gas Limit and Loops
- ☑ Exception Disorder
- ☑ Gasless Send
- ☑ Use of tx.origin
- ☑ Malicious libraries
- ☑ Compiler version not fixed
- ☑ Address hardcoded
- ☑ Divide before multiply
- ☑ Integer overflow/underflow
- ☑ ERC's conformance
- ☑ Dangerous strict equalities
- ☑ Tautology or contradiction
- ☑ Return values of low-level calls
- ☑ Missing Zero Address Validation
- ☑ Private modifier
- ☑ Revert/require functions
- ☑ Multiple Sends
- ☑ Using suicide
- ☑ Using delegatecall
- ☑ Upgradeable safety
- ☑ Using throw

- [x] Using inline assembly
- [x] Unsafe type inference
- [x] Style guide violation
- [x] Implicit visibility level.

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments, match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools And Platforms Used For Audit

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity statistical analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational Issues

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

Security vulnerabilities identified that must be resolved and are currently unresolved.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/ impact of the security issue, but are not completely resolved.

Medium Severity Issues

Removing the registered token is not possible

Resolved

Path

TokenBridgeImplementation.sol

Function

registerToken()

Description

The owner can register tokens with registerToken() function. However, removing the registered token is not possible.
There may be a case where the project owner considers removing a registered token, or a token might be registered by mistake, requiring removal later. In such cases, having a function to unregister the token address would be helpful.

Recommendation

Verify the logic and add the function to unregister the token.

Incorrect token argument passed in getMinLimits() and getMaxLimits()

Resolved

Path

VaultImplementation.sol

Description

The deposit() calls getMinLimits() and getMaxLimits(), The token address getting passed in both function calls is nativeToken address. Because the limits are getting fetched in the deposit function which is called when the ERC20 tokens are staked, the the token address passed to getMinLimits() and getMaxLimits() should be tokenAddr param received by the deposit().

Recommendation

Verify the logic and change the nativeToken to tokenAddr.

Low Severity Issues

Withdraw function lacks isNotPaused modifier

Resolved

Path

TokenBridgeImplementation.sol

Description

Unlike functions like sendNative and sendTokens, the withdraw function does not have an isNot-Paused modifier. This means that withdraw is accessible even then the contract is paused which is an antipattern.

Recommendation

Add isNotPaused modifier to this function

Failed to check return value.

Resolved

Description

In upgradeImplementation(), checkZeroAddr(newImplementation) is called but the return value is not checked.
But because ERC1967Utils.upgradeToAndCall() calls _setImplementation() which reverts if the newImplementation.code.length == 0, the transaction will revert anyway when the newImplementation would be zero address.

Recommendation

Fix the required functionality.

Only transferFrom can be used

Resolved

Path

TokenBridgeImplementation.sol

Function

sendTokens()

Description

Only transferFrom() can be used instead of using both transferFrom() on L63 and transfer() on L69. The transfer from () on L63 will transfer the amount directly from the msg.sender to vault. This will also minimize the gas cost.

Recommendation

Consider using transferFrom() to transfer the amount directly from the msg.sender to vault.

Teams Comment

Supra Team's Comment : we said vault is something which was not exposed to user for the reason of in future token bridge may have multiple vaults. So, User only interacts with token bridge.

Fixed in

ad7e6431515d0ea5b3c6cfca61bf7b7fbd90eecd

Redundant allowance check

Resolved

Description

Allowance check on L55 is redundant as it will revert anyway while transferring (let's say using transferFrom()) if there isn't enough allowance (assuming the token reverts when there's less allowance remaining.)

Recommendation

Any specific case for which an allowance check was added can be discussed if it's added for an already known case. Otherwise, it can be removed.

Store and reuse the fee

Resolved

Path

TokenBridgeImplementation.sol

Function

sendTokens(), sendNative()

Description

In both sendTokens() and sendNative() the fee (getCurrentFee()) is fetched two times, one for comparison with the amount and the other while creating the MessageData struct. It can be stored in a local variable and reused instead of calling two times.

Recommendation

Consider storing the getCurrentFee() value and reusing it.

The unsupported chain ID can be used by user

Resolved

Path

TokenBridgeImplementation.sol

Function

sendTokens(), sendNative()

Description

sendTokens() and sendNative() are taking toChainId as parameter. There is no validation for the toChainId apart from reverting when checkZeroValue(toChainId) is true.
This means any toChainId can be entered.

Recommendation

Verify the logic and add the validation for adding only supported chain IDs where the supported chain IDs can be added through other dedicated function.

Add validation for toChainId in sendNative()

Resolved

Path

TokenBridgeImplementation.sol

Function

sendNative()

Description

The sendTokens() validates the entered toChainId with checkZeroValue() and reverts if it's true. A similar check can be added to the sendNative().

Recommendation

Consider adding code to check the toChainId value.

Centralization concern

Resolved

Description

The admin can withdraw users' staked tokens. A malicious or compromised admin account can cause risk if it tries to steal the amount.

Recommendation

Use a multi-sig wallet as an admin.

Teams Comment

Supra Team's Comment: Right now its only a one way bridge, so there is no way to move funds from vault. Thats why we added an admin withdrawal but this doesn't make it centralized, to gain users trust we made it as a delayed withdrawal.

Audit Team Comment: adding delayed withdrawal doesn't matter here because user has nothing to stop that.

Informational Severity Issues

Floating pragma

Resolved

Description

Multiple contracts are using version (^0.8.20) with floating pragma instead of locking to a specific version. floating pragmas allow the contract to be compiled with any version greater than or equal to the specified version. If the contract wasn't thoroughly tested with that version, this can introduce possible bugs.

Recommendation

Consider using a fixed solidity version with which the contracts are getting thoroughly tested.

Block reorg can cause state inconsistencies

Acknowledged

Path

TokenBridgeImplementation.sol

Description

Consider a scenario where source chain underwent a block reorganization after event emission.
Alice wants to send some tokens from Polygon to destination chain
She initiates a transaction that gets confirmed as event is emitted that is picked up by relayers
Shortly after this, block reorg happened
On polygon, the state will be as if alice never initiated a transaction and on destination chain there will be a transaction representing the same.
This effectively duplicated alice's asssets

Recommendation

Implement mechanisms to detect reorg in those connected chains where reorg can happen (like Polygon), and implement methods to reverse the transaction in if it detects the reorg and invalidation of the original Polygon transaction.

An attacker can DOS the system if the source chain has considerably less gas fee

Resolved

Path

TokenBridgeImplementation.sol

Description

There is no minimum amount enforced to send tokens from source to destination. We know that there is a rate limit set up to ensure only a certain number of transactions can go through via relayers.
An attacker can do many transactions of tiny deposits and can DOS the system since no other transaction will be executed until these tiny ones are processed.
The attack is more cost-effective in chains like Polygon where gas fee is very low.

Recommendation

Set a minimum threshold for deposits to ensure only economically meaningful transactions are processed.
Introduce rate limiting to prevent any single user from flooding the system with transactions in a short period.

Remove the else statements for returning the "false" value

Resolved

Path

tokenBridge-service\implementations\Helpers.sol

Function

checkZeroAddr(), checkZeroBytes32(), checkZeroValue()

Description

Functions are explicitly returning false values if the value/param is not zero in these functions: checkZeroAddr(), checkZeroBytes32(), checkZeroValue()

But because the default value returned (if no value is returned) is false, Hence the else return false; statement is redundant and can be removed.

Recommendation

remove the else return false statements for returning the false value if the if{} fails.

Check if the indexed parameters are required.

Resolved

Path

VaultImplementation.sol

Function

withdraw(), adminWithdraw(), execAdminWithdraw()

Description

Since some of the project logic can be dependent on the emitted events, Check if the functions like withdraw() (not getting used currently), adminWithdraw(), execAdminWithdraw(), and others need to have events that will be monitored with any services.

Recommendation

Verify the logic and add the required events. If events are present then check if the parameters should be indexed.

updateTokenFee() allows 0 address as token address

Acknowledged

Path

relay-operator\implementations\RelayOperatorImpl.sol

Function

updateTokenFee(), _setTokenFee()

Description

updateTokenFee() and _setTokenFee() allows adding fees for 0 address. In this case resetting the fee for zero address is not possible as the _setTokenFee() reverts if _tokenFee is 0.

Recommendation

Add a check and revert if the address is zero address.

Unused tokenBridgeContract address

Resolved

Path

relay-operator\implementations\Helpers.sol

Function

_setTokenBridgeService()

Description

tokenBridgeContract address is not used anywhere and additionally, it’s not even a public variable (in case of showing the bridge address to increase transparency) and that's why the _setTokenBridge-Service() is redundant.

Recommendation

Remove the tokenBridgeContract and _setTokenBridgeService() functionality.

The contract lacks the functionality to transfer the ownership

Resolved

Path

VaultImplementation.sol, RelayOperatorImpl.sol, HypernovaImplementation.sol

Description

Check if the ownership transfer functionality is required to be added in helpers of the following contracts:
VaultImplementation.sol, RelayOperatorImpl.sol, HypernovaImplementation.sol

Recommendation

Consider adding ownership transfer functionality if required.

Check the expected functionality

Resolved

Path

TokenBridgeImplementation.sol

Function

sendTokens()

Description

On L71 if the bool value for the transfer is false then it never calls vault.deposit() but still calls the postMessage() on Hypernova.
It should be checked if it should revert the transaction when the bool success value would be false.

Additionally, It should be noted that since the function call already reverts on L65 if the return value from the transferFrom() is false. The token transfer to the vault contract should never fail given that the token is ERC20 compliant and the tokens are already transferred to this contract.

Recommendation

Consider checking the required logic.

postMessage() external function can be maliciously used

Acknowledged

Path

HypernovaImplementation.sol

Function

postMessage()

Description

Since anyone can call postMessage() function, anyone can create the messageData e.g. for sending tokens on another chain without actually sending the tokens to the vault.

Since the relayer would be listening to these events, It will process this transaction if no additional verification occurs because, on the smart contract level, the valid event is emitted.

Even though msg.sender i.e. caller parameter would be different in the emitted event here, it's worth taking note if it creates any problem for the logic.

Recommendation

Verify the logic and restrict the postMessage() access only to the bridge contract.

Teams Comment

Supra Team's Comment: postMessage is just an event emitting function, anybody can call it. We do informed that token bridge is service contract which are going to use this hypernove-core to emit an event, in future we will be having other service contracts which use the same postMessage. And also, in the event signature we have the caller param, when any other user calls it we can find it and simply ignore those events.

Functional Tests

Some of the tests performed are mentioned below:

- test_updateGasPrice(uint256)
- test_pause()
- test_postMessage(bytes32)
- test_registerToken()
- test_sendToken(address
- uint256)
- test_sendNative(address
- uint256)
- test_sendNativeWithZeroValueAndOnlyFees(address)
- testFail_sendToken4WithZeroValue(bytes32
- address)
- testAdminWithdraw(address
- uint256)
- testExecAdminWithdraw(address
- uint256)
- testExecAdminWithdraw_revert_with_Locked(address
- uint256)
- testFail_registerToken3()
- test_deposit_limit_breached()
- test_disableInitializers()

- ✔ test_execAdminWithdraw_deletes_adminTransfers_data_on_withdrawl(address
- ✔ uint256)
- ✔ test_initialize_again_reverts()
- ✔ test_multipleUsersSendNative()
- ✔ test_multipleUsersSendToken()
- ✔ test_multiple_adminWithdraw_for_deposited_amount()
- ✔ test_ownershipTransfer()
- ✔ test_postMessage_directly_callable()
- ✔ test_reset_deposit_limit()
- ✔ test_sendToken_revert_because_no_allowance(address
- ✔ uint256)
- ✔ test_unsupported_chain(address
- ✔ uint256)
- ✔ test_upgradeImplementation()
- ✔ test_upgradeImplementationToZeroAddress()

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the Supra Hypernova contracts. We performed our audit according to the procedure described above.

Some issues of Medium,low and informational severity were found, In the End, Supra team resolved almost all issues.

Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the received smart contract source code

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem

7+ Years of Expertise	**1M+** Lines of Code Audited
$30B+ Secured in Digital Assets	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

March 2025

For

SUPRA

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com