Blaize.Security

January 30th 2023 / V. 1.0

BLUELIGHT

SMART CONTRACT AUDIT

TABLE OF CONTENTS

AUDIT RATING

Kale Bridge contracts` source code was taken from the repository provided by the Bluelight team.

SCORE **9.8**/10

The scope of the project includes the **Kale Bridge** set of contracts**:**

kale-bridge-v2\contracts
BridgeBase.sol
BridgeBsc.sol
BridgeEmergencyStop.sol
BridgeEth.sol
BridgeSigTransfer.sol
BridgeTransfer.sol
BridgeUserRegistry.sol
Signer.sol
BridgeRefundRequest.sol
BridgeRoles.sol

user-token-registry\contracts\core

BlockTokens.sol
BlockUsers.sol
RegistryRoles.sol
RegistryStorage.sol

user-tokenregistry\contracts\extensions
ERC20UserRegistry.sol
ERC721UserRegistry.sol
ERC1155UserRegistry.sol
UserLock.sol

user-token-registry\contracts\extensions-upgradeable

ERC20UserRegistryUpgradeable.sol
ERC721UserRegistryUpgradeable.sol
ERC1155UserRegistryUpgradeable.sol
UserLockUpgradeable.sol

user-token-registry\contracts\libraries\VerboseReverts.sol
user-token-registry\contracts\Registry.sol
kale-bnb\contracts\BNBKale.sol

Repository:
https://github.com/viewpoint-labs/smart-contracts/tree/kale-bridge

Initial commit:

- e1dc11f756724322c14b88b8c3486620d9dcba17

Final commit:

- 78cc05b5a49c550cf92af2cba146e375cf569939

TECHNICAL SUMMARY

During the audit, we inspected the security of the smart contracts of the Kale Bridge protocol. Our task was to find and describe any security issues in the smart contracts of the platform. This report presents the findings of the security audit of the **Kale Bridge** smart contracts conducted during **January 12th, 2023 - January 30th, 2023.**

Testable code

The testable code has 99% coverage, which is above the industry standard of 95%.

The scope of the audit includes the unit test coverage, which is based on the smart contracts code, documentation, and requirements presented by the Kale Bridge team. The coverage is calculated based on the set of the Hardhat framework tests and scripts from additional testing strategies. However, in order to ensure full security of the contract, the Blaize.Security team suggests the Kale Bridge team launch a bug bounty program to encourage further active analysis of the smart contracts.

THE GRAPH OF VULNERABILITIES DISTRIBUTION:

The table below shows the number of the detected issues and their severity. A total of 11 problems were found. Most of the issues were successfully fixed by the Kale Bridge team.

	FOUND	FIXED/VERIFIED
Critical	0	0
High	2	2
Medium	0	0
Low	2	2
Lowest	7	6

SEVERITY DEFINITION

Critical

The system contains several issues ranked as very serious and dangerous for users and the secure work of the system. Requires immediate fixes and a further check.

High

The system contains a couple of serious issues, which lead to unreliable work of the system and migh cause a huge data or financial leak. Requires immediate fixes and a further check.

Medium

The system contains issues that may lead to medium financial loss or users' private information leak. Requires immediate fixes and a further check.

Low

The system contains several risks ranked as relatively small with the low impact on the users' information and financial security. Requires fixes.

Lowest

The system does not contain any issues critical to the secure work of the system, but best practices should be implemented.

AUDITING STRATEGY AND TECHNIQUES APPLIED/PROCEDURE

We have scanned this smart contract for commonly known and more specific vulnerabilities:

- Unsafe type inference;
- Timestamp Dependence;
- Reentrancy;
- Implicit visibility level;
- Gas Limit and Loops;
- Transaction-Ordering Dependence;
- Unchecked external call - Unchecked math;
- DoS with Block Gas Limit;
- DoS with (unexpected) Throw;
- Byte array vulnerabilities;
- Malicious libraries;
- Style guide violation;
- ERC20 API violation;
- Uninitialized state/storage/ local variables;
- Compile version not fixed.

Procedure

We checked the contract for the following parameters:

- Whether the contract is secure;
- Whether the contract corresponds to the documentation;
- Whether the contract meets the best practices in the efficient use of gas, code readability.

Automated analysis:

Scanning contract by several publicly available automated analysis tools such as Mythril, Solhint, Slither, and Smartdec. Manual verification of all the issues found with these tools.

Manual audit:

Manual analysis of smart contracts for security vulnerabilities. We checked smart contract logic and compared it with the one described in the documentation.

EXECUTIVE SUMMARY

During the audit, Blaize Security has audited the whole set of smart contracts within 3 folders. The protocol consists of Kale BNB ERC20 token, a set of the Bridge smart contracts and a set of smart contracts necessary for the user and token registry.

The goal of the audit was to analyze the security level of the smart contracts against the list of common vulnerabilities and auditors' own checklist, ensure that Solidity best practices in terms of code quality and gas optimization are applied, verify the security of users' funds and the security of Bridge implementation.

There were 2 high and several low and lowest issues detected during the manual part of the audit. High-severity issues were connected to the ability of the owner to withdraw tokens from the Bridge smart contract and the ability of the signer to claim tokens in any quantity and to any receiver. Both issues were successfully resolved by the Bluelight team. To solve the first issue, the team restrained the owner from withdrawing Bridge tokens, and in order to refund users, a refund system was implemented. For the second issue, a system of multiple signers was implemented so that a certain number of signatures must be provided by the signers to process claiming. Though the protocol still can work with a single signer, which is why it is the Bluelight team's responsibility to keep a sufficient number of signers in the protocol. Also, a single admin can still claim tokens to any address using the claim function. Other issues were connected to the lack of variables' validation, clarification of the business logic of the protocol, and other minor things. Most of them were successfully fixed by the Bluelight team as well. It should also be mentioned that the BNBKale.sol is an upgradable smart contract, which means that the owner of the protocol can update its logic at any time. All the issues can be seen in the Complete analysis section.

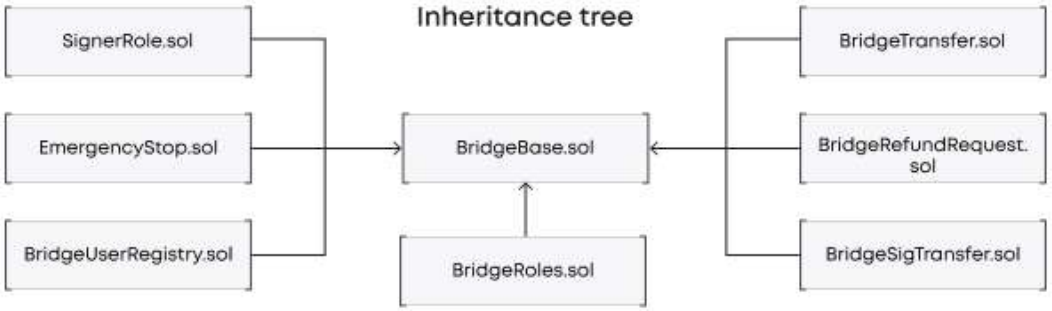
The overall security of the protocol is high enough. Smart contracts are well-written and contain a detailed natspec documentation. The Bluelight team has also prepared a set of unit tests, which have a sufficient coverage. Nevertheless, the Blaize.Security team has prepared their own set of unit-tests, including a set of additional scenarios, to verify the security and correctness of the implementation of the Bridge. Once the fixes were applied, the protocol has passed all the security tests.

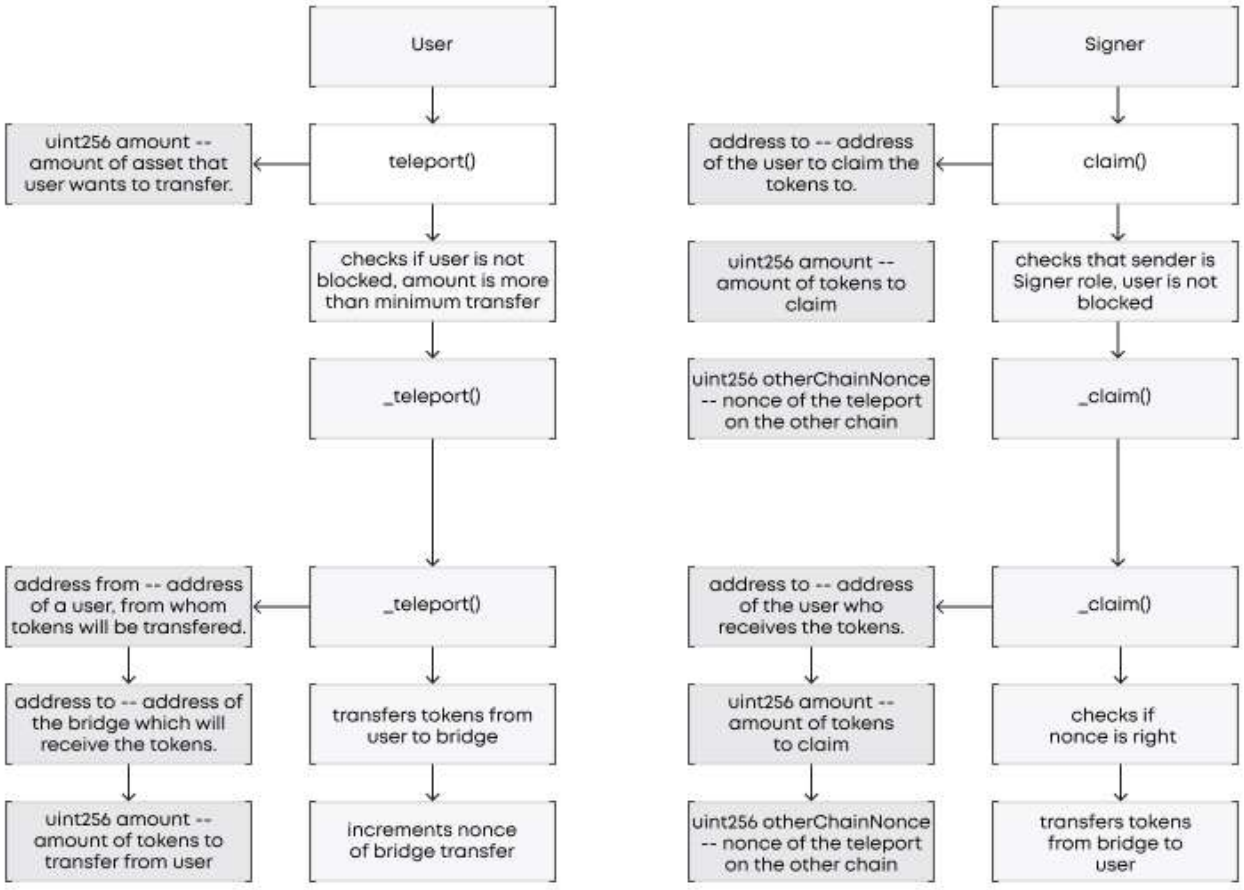
	RATING
Security	9.9
Gas usage and logic optimization	9.8
Code quality	9.6
Test coverage	9.9
Total	9.8

KALE BRIDGE SCHEME

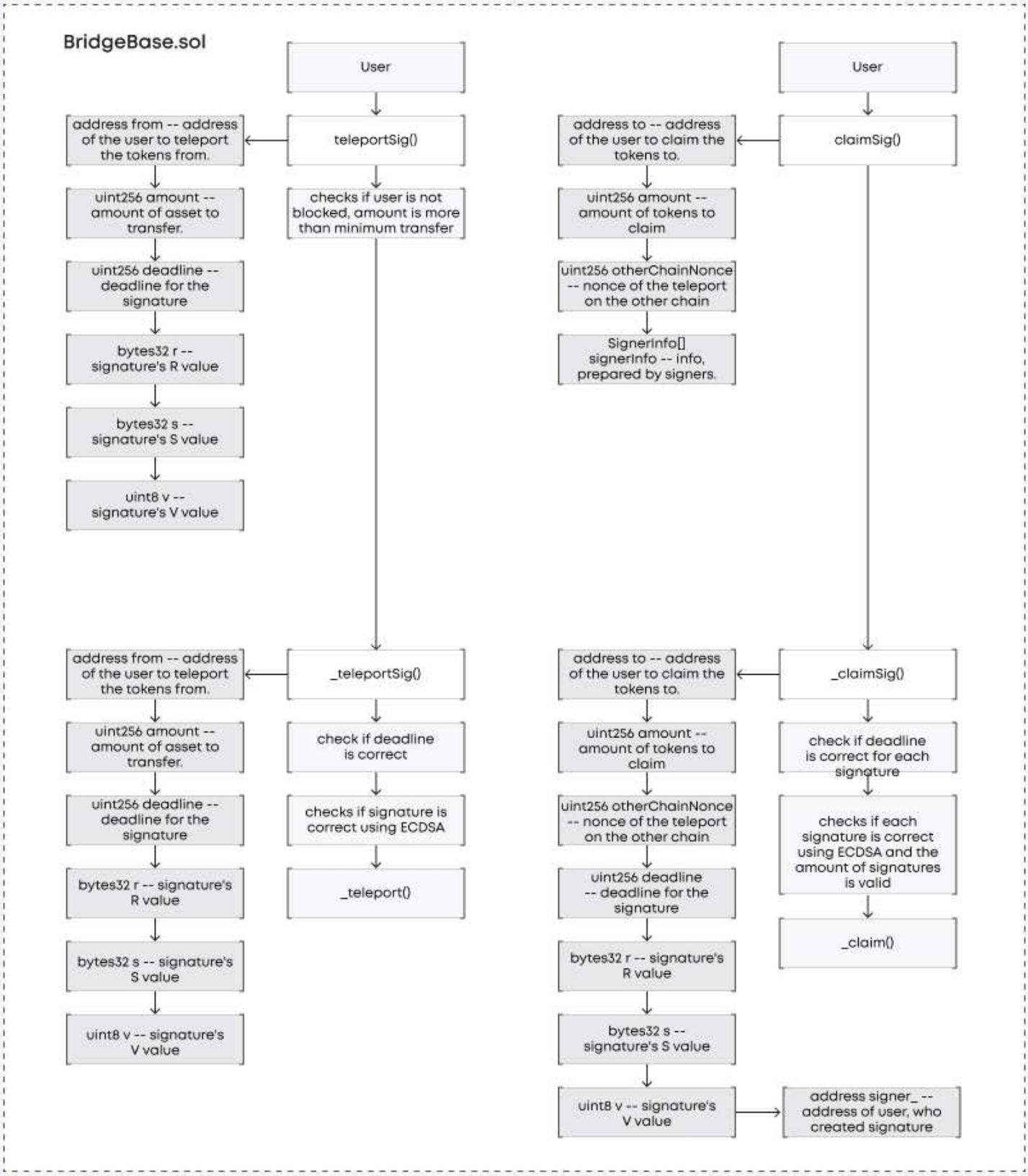
BridgeBase.sol

BridgeBase.sol is a smart contract that inherits SignerRole, BridgeUserRegistry, EmergencyStop, BridgeTransfer, and BridgeSigTransfer, BridgeRefundRequest, BridgeRoles. The smart contract implements the main functionality for the bridge like transferring assets from one chain to another.



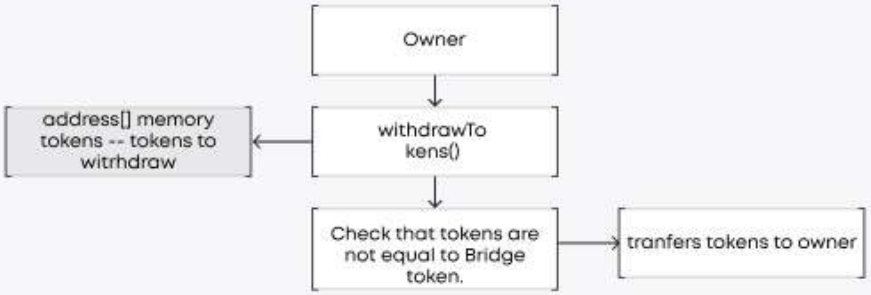


KALE BRIDGE SCHEME



KALE BRIDGE SCHEME

BridgeBase.sol



BridgeRefundRequest.sol

BridgeRefundRequest.sol is a contract that contains the main functionality for refunding tokens to users. Users can request a refund and the owner can decide whether to approve or decline it.

- requestRefund()
- _approveRefund()
- _declineRefund()
- _reopenRefund()

BridgeUserRegistry.sol

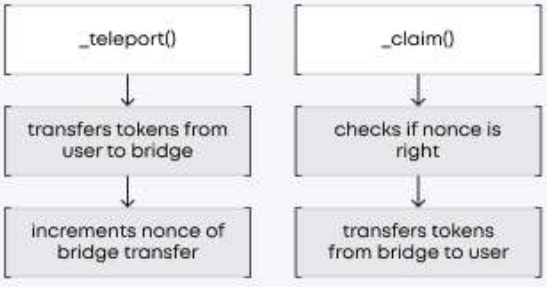
BridgeUserRegistry.sol is a contract that indicates the user status on the bridge. Its main function is a modifier that checks if user is not blocked.

Signer.sol

Signer.sol is an abstract contract that determines the Signer Role on the bridge. It functions as a modifier to check if the sender is a Signer and transfer the role to another address.

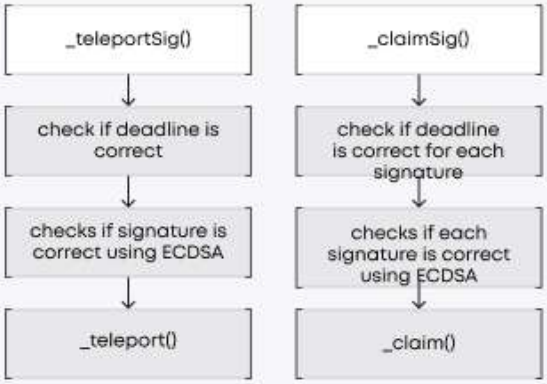
BridgeTransfer.sol

BridgeTransfer.sol is a contract that contains the main functionality of transferring and claiming tokens on the bridge.



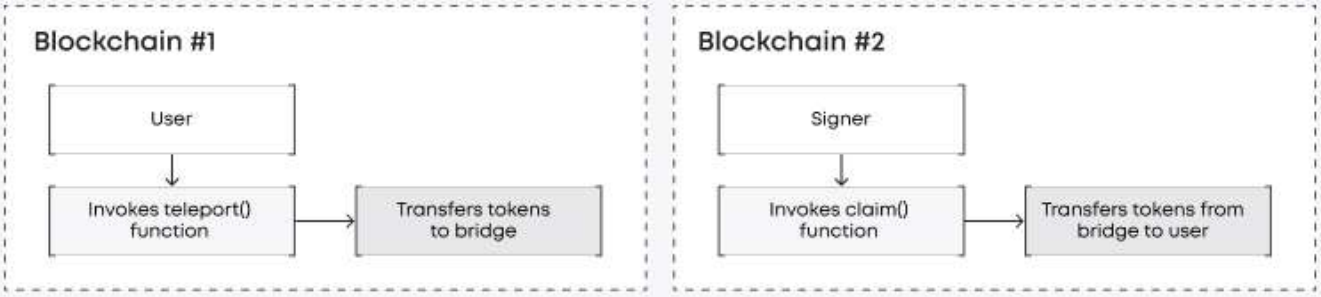
BridgeSigTransfer.sol

BridgeSigTransfer.sol is a contract that contains the main functionality of the BridgeTransfer contract with a signature.



KALE BRIDGE SCHEME

Transfer tokens from one chain to another



RegistryRoles.sol

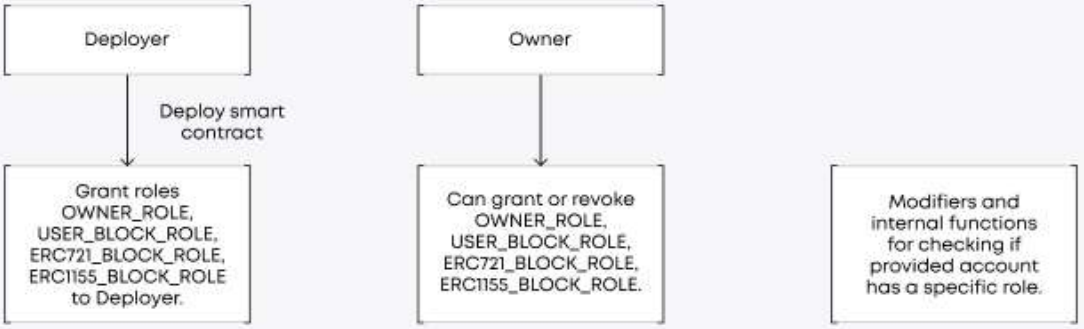
RegistryRoles is an upgradable smart contract that extends the functionality of a standard AcessControl by OpenZeppelin by adding a new custom admin role and several additional roles, an external interface for managing roles, and internal modifiers and functions for checking the roles.

New admin role:

- OWNER_ROLE

Roles within the smart contract:

- USER_BLOCK_ROLE
- ERC721_BLOCK_ROLE
- ERC1155_BLOCK_ROLE



RegistryStorage.sol

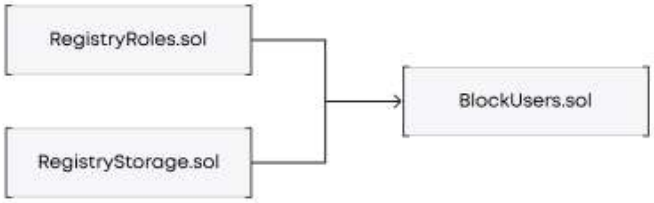
RegistryStorage is an abstract upgradable smart contract designed to store the address of blocked users and IDs of ERC721 and ERC1155 smart contracts. The contract contains internal setters and getters for block statuses. By default, these functions aren't restricted and don't emit an event on any storage change, thus this functionality is to be implemented in the ancestor smart contract.

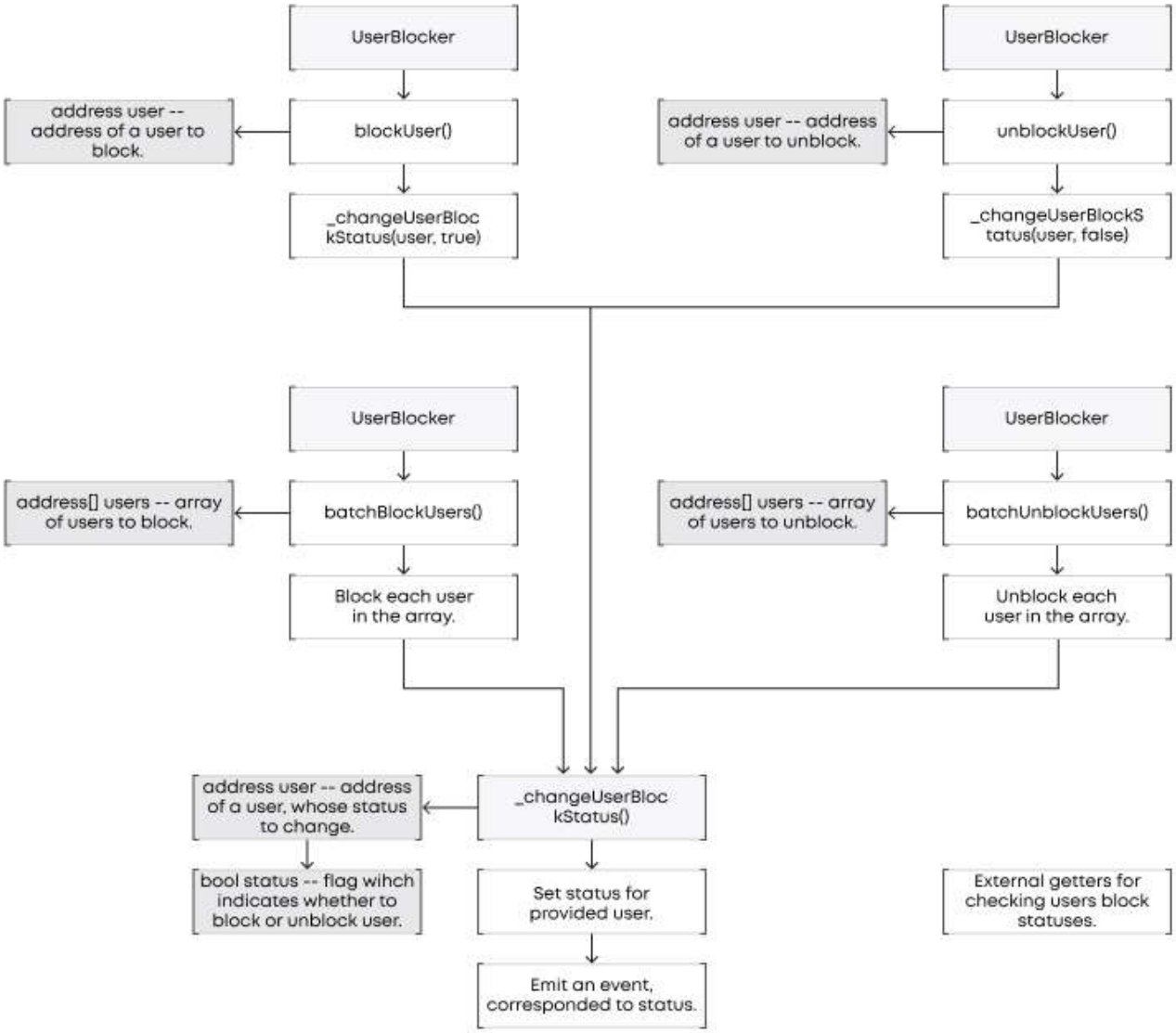
KALE BRIDGE SCHEME

BlockUsers.sol

BlockUsers.sol is an upgradable smart contract that inherits RegistryRoles and RegistryStorage. The smart contract implements external functionality for (un)blocking users individually and in batch and also check the block status of any user.

Inheritance tree



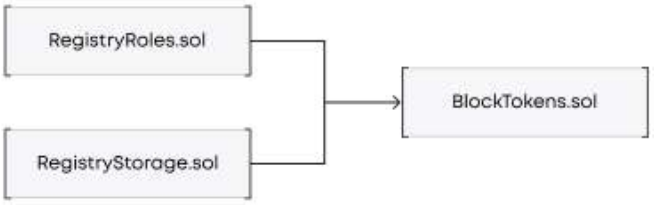


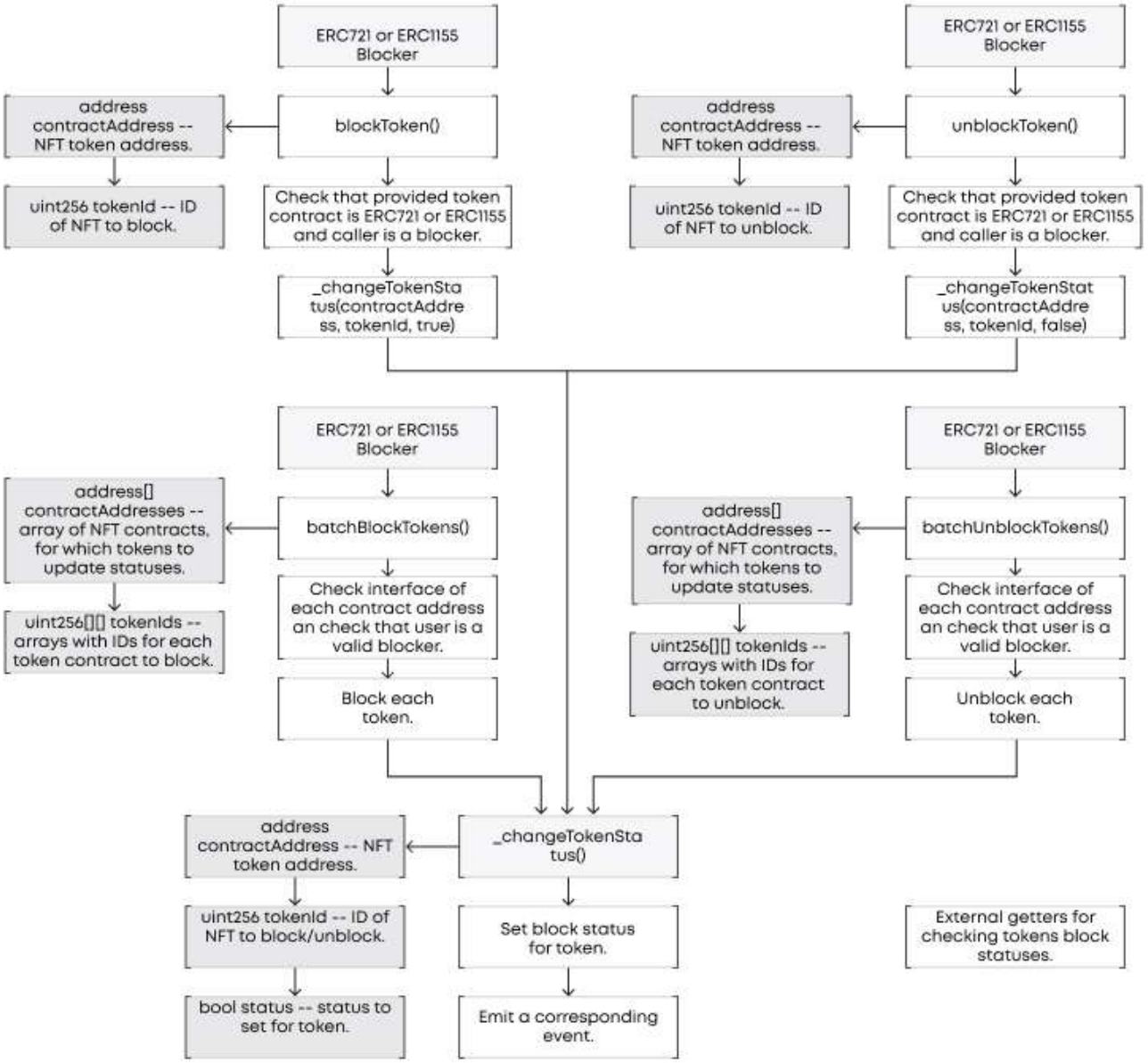
KALE BRIDGE SCHEME

BlockTokens.sol

BlockTokens.sol is an upgradable smart contract that inherits RegistryRoles and RegistryStorage. The smart contract implements external functionality for (un)blocking tokens individually and in batch and check the block status of any token.

Inheritance tree



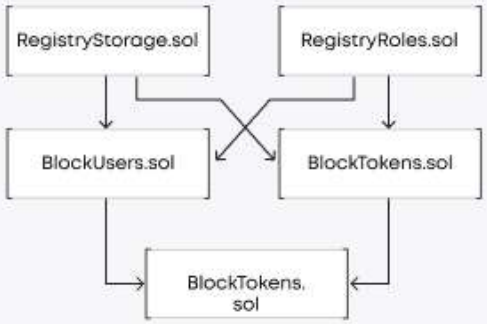


KALE BRIDGE SCHEME

Registry.sol

Registry.sol is an upgradable smart contract that inherits all contracts connected to the Block functionality, thus implementing all the logic. The contract also contains several getter functions to return users and tokens block statuses in different combinations (e.g. the status of a single user and a token, the status of a single user and multiple tokens, etc.)

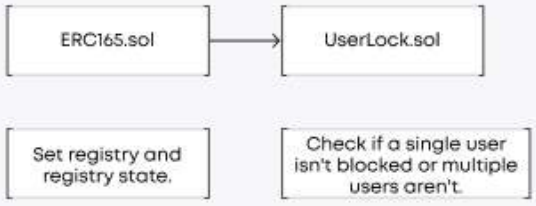
Inheritance tree



UserLock.sol

UserLock.sol is an abstract smart contract designed to interact with Registry.sol and validate if a specific user is blocked in the Registry.sol smart contract. Thus, the contract can be inherited by other smart contracts that need such validations.

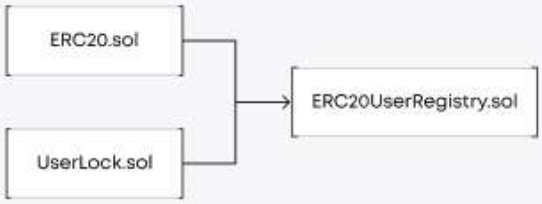
Inheritance tree

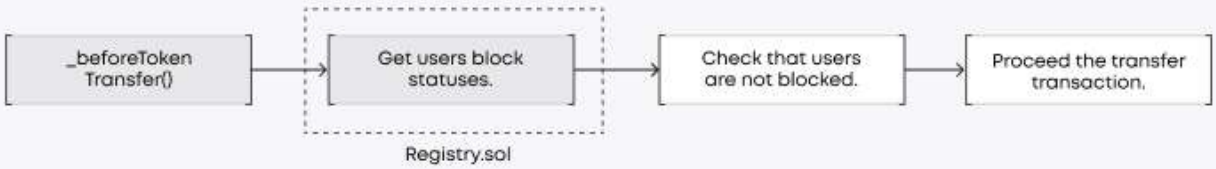


ERC20UserRegistry.sol

ERC20UserRegistry.sol is an abstract ERC20 contract where the _beforeTokenTransfer() hook is overriden in order to validate that both the sender and the recipient are not blocked in the Registry.sol. There is also an upgradable version of this contract, ERC20UserRegistryUpgradeable.sol.

Inheritance tree



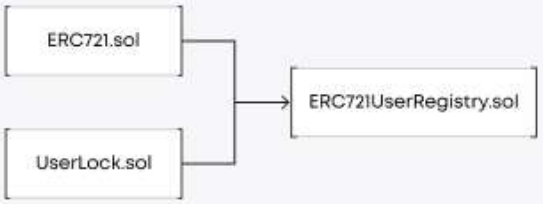


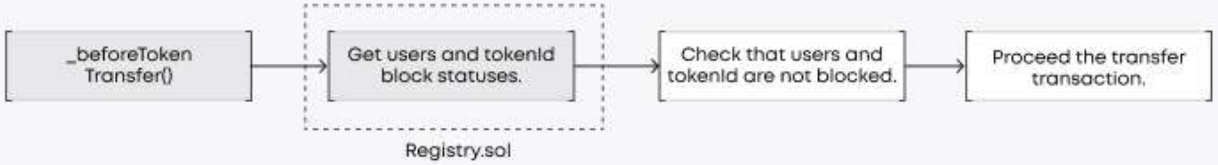
KALE BRIDGE SCHEME

ERC721UserRegistry.sol

ERC721UserRegistry.sol is an abstract ERC721 contract where the _beforeTokenTransfer() hook is overriden in order to validate that the sender, the recipient, and the operator, as well as the transferred tokenId are unblocked in the Registry.sol. There is also an upgradable version of this contract, ERC721UserRegistryUpgradeable.sol.

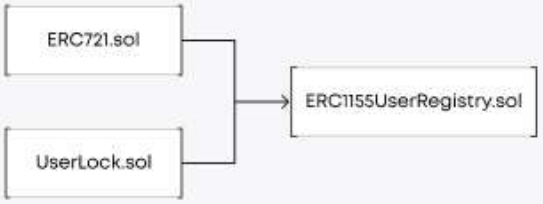
Inheritance tree

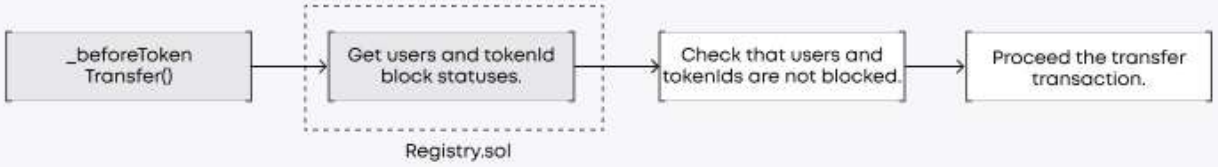




ERC1155UserRegistry.sol

ERC1155UserRegistry.sol is an abstract ERC1155 contract where the _beforeTokenTransfer() hook is overriden to validate that the sender, the recipient and the operator, as well as the transferred tokenIds are not blocked in the Registry.sol. There is also an upgradable version of this smart contract, ERC1155UserRegistryUpgradeable.sol.





COMPLETE ANALYSIS

HIGH-1	✔ Resolved

The owner can withdraw tokens at any time.

BridgeBase.sol: emergencyWithdraw().
The owner account can withdraw any ERC20 token from the contract's balance, including the token that is transferred across chains. As a result, in case the private key of the owner account is exploited, users' funds can be withdrawn directly from the contract. The issue is marked as high as such functionality creates a dangerous backdoor where the owner of the contract has direct access to users' funds. Thus, it is usually recommended to remove such functionality from the contract. In case the withdrawal could only happen in case of emergency, the owner address should be a multisig wallet or some sort of a DAO with the Timelock contract applied and additional restrictions implemented in the contract (to prevent any rug-pull like activity).

Recommendation:

Confirm the necessity of an emergency withdrawal function and that the owner account will be a multisig or a DAO.

From the client:

The system to withdraw all tokens was created to mitigate refunds for users, but it showed as a malicious one. To prevent the owner from withdrawing all tokens at any time and still be able to perform refunds for users, a system for performing refunds was implemented.

Post-audit

The team added new functionality to withdraw tokens that are not the bridge token. For the bridge token, new refund functionality was created so that users can request their tokens back.

HIGH-2 ✔ Resolved

The Signer has direct access to the funds.

BridgeBase.sol: claim(), claimSig().

As for now, when the user wants to transfer tokens using the bridge, the Signer set by the owner should invoke the claim function to transfer tokens to the user on another chain. Although this is a common approach in the work of a cross-chain bridge, where the backend invokes a function on the other bridge to fulfill the cross-chain operation, the backend part of the protocol is out of scope of the current audit.

The current principle of work of the bridge is as follows:

1) The user calls the teleport function on chain 1 and transfers N tokens to the bridge.
2) Backend listens to teleport events, parses the data from them and prepares calldata with the receiver and the amount of funds for the claim transaction on chain 2.
3) Backend calls the claim function on chain 2 with prepared calldata, and the funds are transferred to the receiver.

Currently, it is unknown how step 2 is processed and if a valid user and the amount of funds are specified in the calldata.

Thus, the level of decentralization of backend is unknown, and there is a risk that the signer account can transfer funds from the bridge to any account without a corresponding event on the other chain. That also creates a centralization risk. Therefore, the best approach is to have on-chain validation of the proofs generated on both sides, but that is a general recommendation for bridges rather than this particular case.

Recommendation:

Validate the safety of the Signer's private key and that it will only be used for cross-chain transfers. Also, validate the safety of backend and its processing of events and cross-chain transfers.

Post-audit:

A system of multiple signers was implemented in the protocol. Yet, the protocol can still work with one signer, which means that the Bluelight team has to keep a sufficient number of signers.

LOW-1	✔ Resolved

Parameters lack validation.

1) BlockTokens.sol: blockToken(), unblockToken(), batchBlockTokens(), batchUnblockTokens().

Parameters that represent tokens addresses should be validated in order not to be zero addresses.

2) BlockUsers.sol: blockUser(), unblockUser(), batchBlockUsers(), batchUnblockUsers().

It is recommended to validate that parameters representing users addresses are not zero addresses, especially since the block status of the user is checked in _beforeTokenTransfer() for ERC20, ERC721, and ERC1155 smart contracts. This hook is also invoked in the _mint() and _burn() functions, where either the 'from' or 'to' parameter is passed as a zero address. In case it is designed so that mint() and burn() functionality can be blocked in such a manner, it should be validated by the team.

3) Signer.sol: constructor().

4) BridgeTransfer.sol: constructor().

Validate the address parameter during deployment.

Recommendation:

Validate functions parameters.

Post-audit

Necessary validations were added.

LOW-2	✔ Resolved

Wrong minimum amount might be set during deployment.

BridgeBase.sol: constructor().
When the bridge is deployed, the minimum amount that could be sent is used with a 18-decimal token. In the case of a token that has other number of decimals, there should be additional calculations to get the minimum amount right. Otherwise, it should be verified if the bridge takes only 18 decimals tokens.

Recommendation:

Validate that only tokens with 18 decimals will be used OR multiply the amount by 10**decimals for tokens to cover tokens with any possible decimals.

Post-audit:

The minimum amount of token decimals is set now.

LOWEST-1	✔ Resolved

Mappings can be accessed directly in ancestor smart contracts.

RegistryStorage.sol: mappings `_userBlockStatus`, `_contractTokenIdStatus`. Internal setter and getter functions designed to be used in ancestor smart contracts are implemented in the smart contract for these mappings. However, since both mappings are internal, they can be accessed in ancestor smart contracts both for write and read actions. This makes the usage of any additional setters and getters redundant.

Recommendation:

Consider making mappings private so that they can't be accessed directly in ancestor smart contracts OR leave them internal and remove setters and getters.

Post-audit:

Mappings are now private.

LOWEST-2	✔ Resolved

Confusing mechanism of roles management.

RegistryRoles.sol

1) During the deployment of the contract, OWNER_ROLE is set as a default role for every role except for OWNER_ROLE. Thus, all the roles except for the OWNER_ROLE can be granted/revoked via the standard AccessControl external functionality (grantRole(), revokeRole()).
2) The contract contains a duplicate of the AcessControl external functionality for granting/revoking roles (e.g. addOwner(), addUserBlocker(), removeUserBlocker()). However, roles can still be managed via a standard AccessControl external functionality (grantRole(), revokeRole()) by users with the OWNER_ROLE. Thus, it should be verified if it is an intended functionality and roles should be managed both via the AcessControl and RegistryRoles interfaces.

Recommendation:

1) Set the admin role for OWNER_ROLE as well during the initialization.
2) Either remove the duplication of the AcessControl functionality OR forbid the standard external functionality of AcessControl OR verify that roles should be managed both via the AcessControl and RegistryRoles interfaces.

Post-audit:

1) OWNER_ROLE is now set during the initialization for OWNER_ROLE.
2) It was verified that roles could be managed using both AccessControl and RegistryRoles interfaces.

LOWEST-3	Unresolved

Custom errors should be used.

BlockToken.sol: _batchSetTokenStatus(), line 104;
_checkInterfaceAndBlocker(), line 178.
BridgeBase.sol: moreThanMinTeleportAmount(), line 30;
BridgeTransfer.sol: _claim(), line 57;
BridgeSigTransfer.sol: _teleportSig(), line 33; _claimSig(), line 63;
checkDeadline(), line 69;
BridgeEmergencyStop.sol: notInEmergencyStop(), line 12;
inEmergencyStop(),
line 17;
Signed.sol: onlySigner(), line 35; transferSignerRole(), line 51;
Starting from the 0.8.4 version of Solidity, it is recommended to use custom errors instead of storing error message strings in the storage and using the "require" statements. Using custom errors is more efficient in terms of gas expenditure and increases code readability.

Recommendation:

Use custom errors.

LOWEST-4	✔ Verified

Centralization risk in roles management.

RegistryRoles.sol
Since the OWNER_ROLE is an essential role for the protocol, it is recommended to grant it only to multisig wallet accounts. Thus, the risk of private keys being compromised decreases and the decentralization level increases as it requires multiple users to perform any owner's action. The issue is marked as the lowest and doesn't need any changes on the smart contracts.

Post-audit:

According to the team, a multi-sig wallet will be used for OWNER_ROLE.

LOWEST-5	✔ Resolved

The Spender is not checked.

ERC20UserRegistry.sol: _beforeTokenTransfer().
In case the transferFrom() transaction is called and _beforeTokenTransfer() is invoked during it, the spender of the transaction (msg.sender) is not checked. Thus, in case the spender is in the block list, they can still invoke transferFrom() transactions. The issue is marked as the lowest as it regards the business logic of the contract and should be verified by the team if it might cause an issue for the protocol. Consider the case where any malicious user or smart contract that gets approval from valid users can't be banned and can execute malicious transfers.

Recommendation.
Verify that the spender should not be checked OR check the spender in the block list as well.

Post-audit:
A Spender check was added.

LOWEST-6	✔ Resolved

Unnecessary boolean check.

BridgeEmergencyStop.sol: notInEmergencyStop(), inEmergencyStop().
Boolean variables can be used directly. There is no need to compare them with true or false.

Recommendation.
Remove the equality to the boolean constant.
Post-audit:

The functions were removed, a pausable contract functionality was added.

LOWEST-7	✔ Verified

Availability of tokens on the next side of the bridge.

BridgeBase.sol: claim().
When the user wants to transfer tokens on one chain, it is unclear how tokens appear on the bridge on the other chain. For example, the user wants to transfer 100 tokens from Ethereum to BSC. The user calls the teleport function, and 100 tokens are transferred from the user to the bridge on Ethereum. After that, a signer calls the claim function, and 100 tokens have to be transferred from the bridge on BSC to the user. However, it is unclear how the first tokens will appear on the chain as there are no functions necessary for funding the bridges. Thus, it should be verified if the tokens would already be transferred by the owner, or if there is a balance that is stored on the bridge for such transfers, or only own tokens will be used with the ability to mint tokens to the bridge.

Recommendation.
Verify how tokens will appear on the balance of the bridge on the other chain.

Post-audit:

According to the team, all token supplies are minted and distributed across supply wallets on the Ethereum chain. On the BNB Chain, the same amount of tokens is minted and locked on the bridge contract.

kale-bnb\contracts\BNBKale.sol
user-token-registry\contracts\libraries\VerboseReverts.sol
user-token-registry\contracts\Registry.sol

✔ Re-entrancy	Pass
✔ Access Management Hierarchy	Pass
✔ Arithmetic Over/Under Flows	Pass
✔ Delegatecall Unexpected Ether	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)	Pass

kale-bridge-v2\contracts

BridgeBase.sol
BridgeBsc.sol
BridgeEmergencyStop.sol
BridgeEth.sol
BridgeSigTransfer.sol
BridgeTransfer.sol
BridgeUserRegistry.sol
Signer.sol
BridgeRefundRequest.sol
BridgeRoles.sol

✔ Re-entrancy	Pass
✔ Access Management Hierarchy	Pass
✔ Arithmetic Over/Under Flows	Pass
✔ Delegatecall Unexpected Ether	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)	Pass

user-token-registry\contracts\core

BlockTokens.sol
BlockUsers.sol
RegistryRoles.sol
RegistryStorage.sol

✔ Re-entrancy	Pass
✔ Access Management Hierarchy	Pass
✔ Arithmetic Over/Under Flows	Pass
✔ Delegatecall Unexpected Ether	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)	Pass

user-tokenregistry\contracts\extensions

ERC20UserRegistry.sol
ERC721UserRegistry.sol
ERC1155UserRegistry.sol
UserLock.sol

✔ Re-entrancy	Pass
✔ Access Management Hierarchy	Pass
✔ Arithmetic Over/Under Flows	Pass
✔ Delegatecall Unexpected Ether	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)	Pass

user-token-registry\contracts
extensions-upgradeable

ERC20UserRegistryUpgradeable.sol
ERC721UserRegistryUpgradeable.sol
ERC1155UserRegistryUpgradeable.sol
UserLockUpgradeable.sol

✔ Re-entrancy	Pass
✔ Access Management Hierarchy	Pass
✔ Arithmetic Over/Under Flows	Pass
✔ Delegatecall Unexpected Ether	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)	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES BY THE BLAIZE.SECURITY TEAM

Registry IGetStatus interface

- ✓ Set new registry
- ✓ Set new registry state (70ms)

BridgeBase

Main functionality

- ✓ Teleport tokens
- ✓ Teleport tokens with signature
- ✓ Claim tokens
- ✓ Claim tokens with signature
- ✓ Set minimum teleport amount
- ✓ Check processed nonce
- ✓ Start/End emergency
- ✓ Emergency withdraw
- ✓ Transfer Signature role
- ✓ Change registry state
- ✓ Set registry

Revert

- ✓ When try to teleport less that minimum amount
- ✓ When try to teleport when emergency stop
- ✓ When try to claim when emergency stop
- ✓ When not signer try to claim
- ✓ When try to claim with processed nonce
- ✓ When signature is invalid
- ✓ When signature expired
- ✓ When not owner try to set minimum teleport amount
- ✓ When try to set minimum teleport amount when emergency stop
- ✓ When try to emergency withdraw when it is not emergency
- ✓ When not owner try to emergency withdraw
- ✓ When try to set emergency when it is already in emergency
- ✓ When try to end emergency when it is not in emergency
- ✓ When not owner try to set emergency
- ✓ When not owner try to transfer signer role
- ✓ When not owner try to set zero address to signer role

Bridge Scenarios

Signer

- ✓ Can transfer tokens any time to anyone

Owner

- ✓ Can withdraw tokens from bridge (41ms)

Revert

- ✓ When not enough tokens on other bridge
- ✓ When try to transfer native token to contract

Transfer tokens from one chain to other

- ✓ Teleport -> claim
- ✓ TeleportSig -> claimSig (43ms)
- ✓ TeleportSig -> claim
- ✓ Teleport -> claimSig (39ms)

Direct transfer

- ✓ Before Teleport
- ✓ After Teleport
- ✓ Before Claim (53ms)
- ✓ After Claim (56ms)

ERC1155UserRegistry

- ✓ Should implement the LockUsers interface
- ✓ Should implement the ERC1155 interface
- ✓ Should allow safeTransferFrom without lock
- ✓ Should revert safeTransferFrom with sender block
- ✓ Should revert safeTransferFrom with receiver block
- ✓ Should revert safeTransferFrom with token block
- ✓ Should allow safeBatchTransferFrom without lock
- ✓ Should revert safeBatchTransferFrom with sender block
- ✓ Should revert safeBatchTransferFrom with receiver block
- ✓ Should revert safeBatchTransferFrom with token block (44ms)
- ✓ Should revert safeBatchTransferFrom with sender and receiver block (40ms)
- ✓ Should revert safeBatchTransferFrom with sender and token block (48ms)
- ✓ Should revert safeBatchTransferFrom with receiver and token block (47ms)
- ✓ Should revert safeBatchTransferFrom with sender, receiver and token block (52ms)
- ✓ Should revert safeBatchTransferFrom with operator block (39ms)
- ✓ Should allow safeBatchTransferFrom with sender, receiver, token and operator block and disabled registry (58ms)

ERC1155UserRegistry

- ✓ Should implement the LockUsers interface
- ✓ Should implement the ERC1155 interface
- ✓ Should allow safeTransferFrom without lock
- ✓ Should revert safeTransferFrom with sender block
- ✓ Should revert safeTransferFrom with receiver block
- ✓ Should revert safeTransferFrom with token block
- ✓ Should allow safeBatchTransferFrom without lock
- ✓ Should revert safeBatchTransferFrom with sender block (39ms)
- ✓ Should revert safeBatchTransferFrom with receiver block
- ✓ Should revert safeBatchTransferFrom with token block (48ms)
- ✓ Should revert safeBatchTransferFrom with sender and receiver block (44ms)
- ✓ Should revert safeBatchTransferFrom with sender and token block (52ms)
- ✓ Should revert safeBatchTransferFrom with receiver and token block (49ms)
- ✓ Should revert safeBatchTransferFrom with sender, receiver and token block (54ms)
- ✓ Should revert safeBatchTransferFrom with operator block (43ms)
- ✓ Should allow safeBatchTransferFrom with sender, receiver, token and operator block and disabled registry (59ms)

ERC20UserRegistry

- ✓ Should return true for supported interface of USER_LOCK_HASH
- ✓ Should transfer tokens without block
- ✓ Should transfer tokens with block of sender
- ✓ Should transfer tokens with block of receiver
- ✓ Should transfer tokens with block of both sender and receiver
- ✓ Should transfer tokens with blocked user and disabled registry
- ✓ Should transfer tokens with blocked user and disabled registry
- ✓ Should transfer tokens with blocked user and disabled registry

1) Blocked user try send tokens
2) User try send tokens to blocked user
3) Burn when address 0 is blocked
4) Mint when address 0 is blocked

ERC20UserRegistry

- ✓ Should return true for supported interface of USER_LOCK_HASH
- ✓ Should transfer tokens without block
- ✓ Should transfer tokens with block of sender

- ✓ Should transfer tokens with block of receiver
- ✓ Should transfer tokens with block of both sender and receiver
- ✓ Should transfer tokens with blocked user and disabled registry
- ✓ Should transfer tokens with blocked user and disabled registry
- ✓ Should transfer tokens with blocked user and disabled registry

ERC721UserRegistry

- ✓ Should implement the LockUsers interface
- ✓ Should implement the IERC721 interface
- ✓ Should implement the IERC721Metadata interface
- ✓ Should transferFrom tokens without block
- ✓ Should revert token transferFrom with sender block
- ✓ Should revert token transferFrom with receiver block
- ✓ Should revert token transferFrom with token block
- ✓ Should revert token transferFrom with sender and token block
- ✓ Should revert token transferFrom with receiver and token block
- ✓ Should revert token transferFrom with sender, receiver and token block
- ✓ Should revert token transferFrom with operator block
- ✓ Should revert token transferFrom with sender, operator, receiver and token block and disabled registry (43ms)

ERC721UserRegistry

- ✓ Should implement the LockUsers interface
- ✓ Should implement the IERC721 interface
- ✓ Should implement the IERC721Metadata interface
- ✓ Should transferFrom tokens without block
- ✓ Should revert token transferFrom with sender block
- ✓ Should revert token transferFrom with receiver block
- ✓ Should revert token transferFrom with token block
- ✓ Should revert token transferFrom with sender and token block
- ✓ Should revert token transferFrom with receiver and token block
- ✓ Should revert token transferFrom with sender, receiver and token block (38ms)
- ✓ Should revert token transferFrom with operator block
- ✓ Should revert token transferFrom with sender, operator, receiver and token block and disabled registry (44ms)

Registry IGetStatus interface

- ✓ Should return true for supported interface of IGetStatus
- ✓ Should return correct userTokenIdBlockStatus

- ✓ Should return correct userTokenIdsBlockStatus
- ✓ Should return correct usersTokenIdBlockStatus
- ✓ Should return correct usersTokenIdsBlockStatus
- ✓ Should return correct usersTokensIdsBlockStatus

BlockTokens

- ✓ Should implement IBlockTokens interface
- ✓ Should block ERC721 token
- ✓ Should unblock ERC721 token
- ✓ Should block ERC1155 token
- ✓ Should unblock ERC1155 token
- ✓ Should batch block ERC721 tokens
- ✓ Should batch unblock ERC721 tokens
- ✓ Should batch block ERC1155 tokens
- ✓ Should batch unblock ERC1155 tokens
- ✓ Should batch block ERC721 and ERC1155 tokens
- ✓ Should batch unblock ERC721 and ERC1155 tokens (53ms)
- ✓ Should revert if batch blocking ERC721 and ERC1155 tokens with different lengths
- ✓ Should revert if batch unblocking ERC721 and ERC1155 tokens with different lengths
- ✓ Should batch get token status
- ✓ Should revert if blocking ERC721 token with invalid role
- ✓ Should revert if unblocking ERC721 token with invalid role
- ✓ Should revert if blocking ERC1155 token with invalid role
- ✓ Should revert if unblocking ERC1155 token with invalid role
- ✓ Should revert if blocking ERC721 token with ERC1155_BLOCK_ROLE
- ✓ Should revert if blocking ERC1155 token with ERC721_BLOCK_ROLE
- ✓ Should revert if unblocking ERC721 token with ERC1155_BLOCK_ROLE
- ✓ Should revert if unblocking ERC1155 token with ERC721_BLOCK_ROLE
- ✓ Should revert if blocking token that does not implement ERC721 or ERC1155
- ✓ Should revert if unblocking token that does not implement ERC721 or ERC1155
- ✓ Should revert if batch blocking token that does not implement ERC721 or ERC1155
- ✓ Should revert if batch unblocking token that does not implement ERC721 or ERC1155
- ✓ Should revert if batch blocking ERC721 and ERC1155 without both roles (68ms)
- ✓ Should revert if trying to block tokens with USER_BLOCK_ROLE
- ✓ Should check batch status of ERC721 and ERC1155 tokens

- ✓ Should not be able to block tokens with ERC721_BLOCK_ROLE removed
- ✓ Should not be able to block tokens with ERC1155_BLOCK_ROLE removed

BlockUsers

- ✓ Should implement the IERC165 interface
- ✓ Should implement the IBlockUsers interface
- ✓ Should add user to owner role (50ms)
- ✓ Should block a user
- ✓ Should block multiple users
- ✓ Should unblock a user
- ✓ Should unblock multiple users
- ✓ Should revert if a non-USER_BLOCK_ROLE tries to block a user
- ✓ Should revert if a non-USER_BLOCK_ROLE tries to unblock a user
- ✓ Should revert if a non-USER_BLOCK_ROLE tries to batch block users
- ✓ Should revert if a non-USER_BLOCK_ROLE tries to batch unblock users
- ✓ Should return the correct user block status
- ✓ Should return the correct batch user block status
- ✓ Should allow a user with the USER_BLOCK_ROLE to block a user
- ✓ Should not allow a user without the USER_BLOCK_ROLE to block a user

UserLock

- ✓ Should return true for supported interface of USER_LOCK_HASH
- ✓ Should get registry
- ✓ Should get registry
- ✓ Should change registry
- ✓ Should change registry state
- ✓ Should allow unblocked user to perform action
- ✓ Should revert if blocked user tries to perform action
- ✓ Should succeed if blocked user tries to perform action and registry is disabled
- ✓ Should allow to perform batch checks
- ✓ Should revert if blocked user tries to perform batch checks
- ✓ Should revert if blocked users try to perform batch checks
- ✓ Should succeed if blocked users try to perform batch checks and registr is disabled

UserLockUpgradeable

- ✓ Should return true for supported interface of USER_LOCK_HASH
- ✓ Should get registry
- ✓ Should get registry
- ✓ Should change registry

- ✓ Should change registry state
- ✓ Should allow unblocked user to perform action
- ✓ Should revert if blocked user tries to perform action
- ✓ Should succeed if blocked user tries to perform action and registry is disabled
- ✓ Should allow to perform batch checks
- ✓ Should revert if blocked user tries to perform batch checks
- ✓ Should revert if blocked users try to perform batch checks
- ✓ Should succeed if blocked users try to perform batch checks and registry is disabled

BridgeBase

- ✓ Claim tokens with 9/12 correct signatures (84ms)
- ✓ Support interface

Refund

- ✓ Request refund
- ✓ Approve refund (39ms)
- ✓ Decline refund
- ✓ Reopen refund (39ms)
- ✓ Process multiple refunds (69ms)

Pausable

- ✓ Pause/unpause contract (41ms)

Withdrawals

- ✓ Withdraw random tokens (57ms)
- ✓ Withdraw ether

Revert

- ✓ When not enough signatures
- ✓ When too many signatures
- ✓ When signature not sorted
- ✓ When try to process refunds with wrong data
- ✓ When try to withdraw bridge token
- ✓ When try to request refund for non-existent transfer
- ✓ When try to request refund again at the same transfer
- ✓ When try to approve non-existent refund request
- ✓ When try to approve refund when delay is not passed
- ✓ When try to reopen refund when it is not declined
- ✓ When owner try to revoke own role
- ✓ When try to renounce/grand/revoke specific role

Bridge Scenarios

Admin

✓ Can transfer tokens any time to anyone

Bridge

✓ Claim is possible if only one signer assigned

218 passing (6s)

TEST COVERAGE RESULTS

FILE	% STMTS	% BRANCH	% FUNCS
Registry.sol	100	100	100
BlockTokens.sol	100	100	100
BlockUsers.sol	100	100	100
RegistryRoles.sol	92	100	88,89
RegistryStorage.sol	100	100	100
ERC1155UserRegistryUpgradeable.sol	100	100	100
ERC20UserRegistryUpgradeable.sol	100	100	100
ERC721UserRegistryUpgradeable.sol	100	100	100

FILE	% STMTS	% BRANCH	% FUNCS
UserLockUpgradeable.sol	100	100	100
ERC1155UserRegistry.sol	100	100	100
ERC20UserRegistry.sol	100	100	100
ERC721UserRegistry.sol	100	100	100
UserLock.sol	100	100	100
BNBKale.sol	100	100	100
BridgeBase.sol	100	100	100
BridgeBsc.sol	100	100	100
BridgeEmergencyStop.sol	100	100	100
BridgeEth.sol	100	100	100
BridgeSigTransfer.sol	100	100	100
BridgeTransfer.sol	100	100	100
BridgeUserRegistry.sol	100	100	100
Signer.sol	100	100	100
BridgeRefundRequest.sol	100	100	100
BridgeRoles.sol	100	100	100
All files	99,63	100	99,5

CODE COVERAGE AND TEST RESULTS FOR ALL FILES BY THE BLUELIGHT TEAM

BNBKale

- ✓ Should upgrade (62ms)

BridgeEmergencyStop

- ✓ Should only allow the owner to emergency stop the contract
- ✓ Should allow the owner to emergency stop the contract
- ✓ Should allow the owner to end the emergency stop
- ✓ Should not be active by default
- ✓ Should not allow the owner to end the emergency stop if it's not active
- ✓ Should not allow the owner to start the emergency stop if it's already active
- ✓ Should not allow the owner to withdraw tokens if the emergency stop is not active
- ✓ Should allow the owner to withdraw tokens if the emergency stop is active

BridgeTransfer

- ✓ Should allow changing address of signer role
- ✓ Should allow teleporting tokens with a signature
- ✓ Should allow claiming tokens with a signature

BridgeTransfer

- ✓ Should allow teleporting tokens
- ✓ Should allow claiming tokens
- ✓ Should not allow teleporting less than the minTeleportAmount
- ✓ Should allow setting the minTeleportAmount

BridgeUserRegistry

- ✓ Should allow to change registry state
- ✓ Should allow to change registry address
- ✓ Should allow to perform action
- ✓ Should not allow to perform action by blocking in external registry

ERC20Permit

- ✓ Should permit (43ms)

Bridge contract

- ✓ Should check balances
- ✓ Should transfer from deployer to the bridge(teleport)
- ✓ Should transfer from the bridge back to the deployer(claim)
- ✓ Should call claim without admin right and fail

ERC1155UserRegistry

- ✓ MUST implement the LockUsers interface
- ✓ MUST implement the ERC1155 interface
- ✓ MUST allow safeTransferFrom without lock
- ✓ MUST revert safeTransferFrom with sender block
- ✓ MUST revert safeTransferFrom with receiver block
- ✓ MUST revert safeTransferFrom with token block (40ms)
- ✓ MUST allow safeBatchTransferFrom without lock
- ✓ MUST revert safeBatchTransferFrom with sender block (44ms)
- ✓ MUST revert safeBatchTransferFrom with receiver block (38ms)
- ✓ MUST revert safeBatchTransferFrom with token block (44ms)
- ✓ MUST revert safeBatchTransferFrom with sender and receiver block (44ms)
- ✓ MUST revert safeBatchTransferFrom with sender and token block (47ms)
- ✓ MUST revert safeBatchTransferFrom with receiver and token block (47ms)
- ✓ MUST revert safeBatchTransferFrom with sender, receiver and token block (52ms)
- ✓ MUST revert safeBatchTransferFrom with operator block (43ms)
- ✓ MUST allow safeBatchTransferFrom with sender, receiver, token and operator block and disabled registry (56ms)

ERC1155UserRegistry

- ✓ MUST implement the LockUsers interface
- ✓ MUST implement the ERC1155 interface
- ✓ MUST allow safeTransferFrom without lock
- ✓ MUST revert safeTransferFrom with sender block
- ✓ MUST revert safeTransferFrom with receiver block
- ✓ MUST revert safeTransferFrom with token block
- ✓ MUST allow safeBatchTransferFrom without lock
- ✓ MUST revert safeBatchTransferFrom with sender block (38ms)
- ✓ MUST revert safeBatchTransferFrom with receiver block
- ✓ MUST revert safeBatchTransferFrom with token block (46ms)
- ✓ MUST revert safeBatchTransferFrom with sender and receiver block (42ms)
- ✓ MUST revert safeBatchTransferFrom with sender and token block (49ms)
- ✓ MUST revert safeBatchTransferFrom with receiver and token block (49ms)
- ✓ MUST revert safeBatchTransferFrom with sender, receiver and token block (53ms)
- ✓ MUST revert safeBatchTransferFrom with operator block (42ms)
- ✓ MUST allow safeBatchTransferFrom with sender, receiver, token and operator block and disabled registry (59ms)

ERC20UserRegistry

- ✓ MUST return true for supported interface of USER_LOCK_HASH
- ✓ MUST transfer tokens without block
- ✓ MUST transfer tokens with block of sender
- ✓ MUST transfer tokens with block of receiver
- ✓ MUST transfer tokens with block of both sender and receiver
- ✓ MUST transfer tokens with blocked user and disabled registry
- ✓ MUST transfer tokens with blocked user and disabled registry
- ✓ MUST transfer tokens with blocked user and disabled registry

ERC20UserRegistry

- ✓ MUST return true for supported interface of USER_LOCK_HASH
- ✓ MUST transfer tokens without block
- ✓ MUST transfer tokens with block of sender
- ✓ MUST transfer tokens with block of receiver
- ✓ MUST transfer tokens with block of both sender and receiver
- ✓ MUST transfer tokens with blocked user and disabled registry
- ✓ MUST transfer tokens with blocked user and disabled registry
- ✓ MUST transfer tokens with blocked user and disabled registry

ERC721UserRegistry

- ✓ MUST implement the LockUsers interface
- ✓ MUST implement the IERC721 interface
- ✓ MUST implement the IERC721Metadata interface
- ✓ MUST transferFrom tokens without block
- ✓ MUST revert token transferFrom with sender block
- ✓ MUST revert token transferFrom with receiver block
- ✓ MUST revert token transferFrom with token block
- ✓ MUST revert token transferFrom with sender and token block
- ✓ MUST revert token transferFrom with receiver and token block
- ✓ MUST revert token transferFrom with sender, receiver and token block
- ✓ MUST revert token transferFrom with operator block
- ✓ MUST revert token transferFrom with sender, operator, receiver and token block and disabled registry (42ms)

ERC721UserRegistry

- ✓ MUST implement the LockUsers interface
- ✓ MUST implement the IERC721 interface
- ✓ MUST implement the IERC721Metadata interface
- ✓ MUST transferFrom tokens without block

- ✓ MUST revert token transferFrom with receiver block
- ✓ MUST revert token transferFrom with token block
- ✓ MUST revert token transferFrom with sender and token block
- ✓ MUST revert token transferFrom with receiver and token block
- ✓ MUST revert token transferFrom with sender, receiver and token block (41ms)
- ✓ MUST revert token transferFrom with operator block
- ✓ MUST revert token transferFrom with sender, operator, receiver and token block and disabled registry (43ms)

Registry IGetStatus interface

- ✓ MUST return true for supported interface of IGetStatus
- ✓ MUST return correct userTokenIdBlockStatus
- ✓ MUST return correct userTokenIdsBlockStatus
- ✓ MUST return correct usersTokenIdBlockStatus
- ✓ MUST return correct usersTokenIdsBlockStatus
- ✓ MUST return correct usersTokensIdsBlockStatus

BlockTokens

- ✓ MUST implement IBlockTokens interface
- ✓ MUST block ERC721 token
- ✓ MUST block ERC721 token
- ✓ MUST block ERC1155 token
- ✓ MUST unblock ERC1155 token
- ✓ MUST batch block ERC721 tokens
- ✓ MUST batch unblock ERC721 tokens
- ✓ MUST batch block ERC1155 tokens
- ✓ MUST batch unblock ERC1155 tokens
- ✓ MUST batch block ERC721 and ERC1155 tokens
- ✓ MUST batch unblock ERC721 and ERC1155 tokens (53ms)
- ✓ MUST revert if batch blocking ERC721 and ERC1155 tokens with different lengths
- ✓ MUST revert if batch unblocking ERC721 and ERC1155 tokens with different lengths
- ✓ MUST batch get token status
- ✓ MUST revert if blocking ERC721 token with invalid role
- ✓ MUST revert if unblocking ERC721 token with invalid role
- ✓ MUST revert if blocking ERC1155 token with invalid role
- ✓ MUST revert if unblocking ERC1155 token with invalid role
- ✓ MUST revert if blocking ERC721 token with ERC1155_BLOCK_ROLE

- ✓ MUST revert if blocking ERC1155 token with ERC721_BLOCK_ROLE
- ✓ MUST revert if unblocking ERC721 token with ERC1155_BLOCK_ROLE
- ✓ MUST revert if unblocking ERC1155 token with ERC721_BLOCK_ROLE
- ✓ MUST revert if blocking token that does not implement ERC721 or ERC1155
- ✓ MUST revert if unblocking token that does not implement ERC721 or ERC1155
- ✓ MUST revert if batch blocking token that does not implement ERC721 or ERC1155
- ✓ MUST revert if batch unblocking token that does not implement ERC721 or ERC1155
- ✓ MUST revert if batch blocking ERC721 and ERC1155 without both roles (70ms)
- ✓ MUST revert if trying to block tokens with USER_BLOCK_ROLE
- ✓ MUST check batch status of ERC721 and ERC1155 tokens
- ✓ MUST not be able to block tokens with ERC721_BLOCK_ROLE removed
- ✓ MUST not be able to block tokens with ERC1155_BLOCK_ROLE removed

BlockUsers

- ✓ MUST implement the IERC165 interface
- ✓ MUST implement the IBlockUsers interface
- ✓ MUST add user to owner role (48ms)
- ✓ MUST block a user
- ✓ MUST block multiple users
- ✓ MUST unblock a user
- ✓ MUST unblock multiple users
- ✓ MUST revert if a non-USER_BLOCK_ROLE tries to block a user
- ✓ MUST revert if a non-USER_BLOCK_ROLE tries to unblock a user
- ✓ MUST revert if a non-USER_BLOCK_ROLE tries to batch block users
- ✓ MUST revert if a non-USER_BLOCK_ROLE tries to batch unblock users
- ✓ MUST return the correct user block status
- ✓ MUST return the correct batch user block status
- ✓ MUST allow a user with the USER_BLOCK_ROLE to block a user
- ✓ MUST not allow a user without the USER_BLOCK_ROLE to block a user

UserLock

- ✓ MUST return true for supported interface of USER_LOCK_HASH
- ✓ MUST get registry
- ✓ MUST get registry
- ✓ MUST change registry
- ✓ MUST change registry state

- ✓ MUST allow unblocked user to perform action
- ✓ MUST revert if blocked user tries to perform action
- ✓ MUST succeed if blocked user tries to perform action and registry is disabled
- ✓ MUST allow to perform batch checks
- ✓ MUST revert if blocked user tries to perform batch checks
- ✓ MUST revert if blocked users try to perform batch checks
- ✓ MUST succeed if blocked users try to perform batch checks and registry is disabled

UserLock

- ✓ MUST return true for supported interface of USER_LOCK_HASH
- ✓ MUST get registry
- ✓ MUST get registry
- ✓ MUST change registry
- ✓ MUST change registry state
- ✓ MUST allow unblocked user to perform action
- ✓ MUST revert if blocked user tries to perform action
- ✓ MUST succeed if blocked user tries to perform action and registry is disabled
- ✓ MUST allow to perform batch checks
- ✓ MUST revert if blocked user tries to perform batch checks
- ✓ MUST revert if blocked users try to perform batch checks
- ✓ MUST succeed if blocked users try to perform batch checks and registry is disabled

173 passing (6s)

TEST COVERAGE RESULTS

FILE	% STMTS	% BRANCH	% FUNCS
Registry.sol	100	100	100
BlockTokens.sol	100	100	100
BlockUsers.sol	100	100	100
RegistryRoles.sol	92	100	88,89

FILE	% STMTS	% BRANCH	% FUNCS
RegistryStorage.sol	100	100	100
ERC1155UserRegistryUpgradeable.sol	100	100	100
ERC20UserRegistryUpgradeable.sol	100	100	100
ERC721UserRegistryUpgradeable.sol	100	100	100
UserLockUpgradeable.sol	100	100	100
ERC1155UserRegistry.sol	100	100	100
ERC20UserRegistry.sol	100	100	100
ERC721UserRegistry.sol	100	100	100
UserLock.sol	100	100	100
BNBKale.sol	77,78	100	60
BridgeBase.sol	100	100	100
BridgeBsc.sol	100	100	0
BridgeEmergencyStop.sol	100	100	100
BridgeEth.sol	100	100	100
BridgeSigTransfer.sol	100	50	100
BridgeTransfer.sol	100	50	100
BridgeUserRegistry.sol	100	100	100
Signer.sol	100	75	100
All files	98,63	94,32	93,13

DISCLAIMER

The information presented in this report is an intellectual property of the customer, including all the presented documentation, code databases, labels, titles, ways of usage, as well as the information about potential vulnerabilities and methods of their exploitation. This audit report does not give any warranties on the absolute security of the code. Blaize.Security is not responsible for how you use this product and does not constitute any investment advice.

Blaize.Security does not provide any warranty that the working product will be compatible with any software, system, protocol, or service and operate without interruption. We do not claim the investigated product is able to meet your or anyone else's requirements and be fully secure, complete, accurate, and free of any errors and code inconsistency.

We are not responsible for all subsequent changes, deletions, and relocations of the code within the contracts that are the subjects of this report.

You should perceive Blaize.Security as a tool that helps investigate and detect any weaknesses and vulnerable parts that may accelerate the technology improvements and faster error elimination.