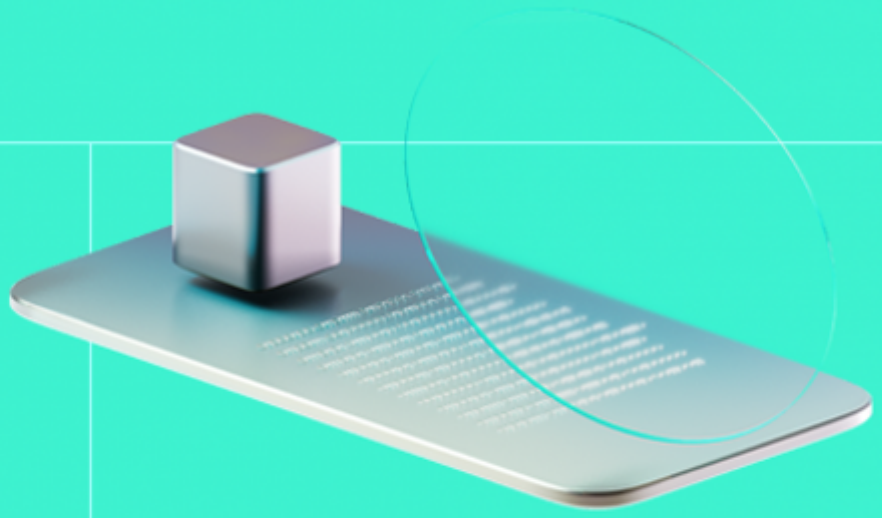
HACKEN

Smart Contract Code Review And Security Analysis Report

Customer: Aviator

Date: 21/08/2024



We express our gratitude to the Aviator team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

SkyBridgeTM by **Aviator Technologies, LLC** is a custom bridge and launchpad solution between the Ethereum Layer-1 and the Base Layer-2 networks that supports any arbitrary ERC-20 or ERC-721 token. **SkyBridge** is built on the existing Optimism bridge, using it wherever possible to ensure the most secure transfer of funds. Most calls mirror the calls in the Optimism bridge, except for the fast bridging functionality. We also augmented the basic transfer functions in the L1 contract to collect fees to fund the liquidity pool used in L1 to enable fast transfers.

Document

Name	Smart Contract Code Review and Security Analysis Report for Aviator
Audited By	David Camps Novi, Viktor Lavrenenko
Approved By	Ataberk Yavuzer
Website	https://aviator.ac/
Changelog	11/07/2024 - Preliminary Report; 21/08/2024 - Final Report
Platform	Base, Ethereum
Language	Solidity
Tags	Bridge, Signatures, EIP712, ERC721, ERC20, MinimalProxy
Methodology	https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/AviatorAC/skybridge
Commit	d140bb2

Audit Summary

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

9	8	1	0
Total Findings	Resolved	Accepted	Mitigated

Findings by Severity

Severity	Count
Critical	0
High	2
Medium	0
Low	6

Vulnerability	Status
F-2024-4237 - onlyEOA Modifier Does Not Prevent Contracts From Calling the Bridge's Functionality	Accepted
F-2024-4213 - Missing Zero Address Validation Can Lead To Loss of Funds	Fixed
F-2024-4214 - Uninitialized flatFeeRecipient Leads to Loss of Fees	Fixed
F-2024-4217 - Public AccessControl Functionality Allows Bypassing of Checks	Fixed
F-2024-4225 - Lack of Refund Functionality Leads To Loss Of Funds	Fixed
F-2024-4250 - Incorrect Signature Validation Will Revert Execution	Fixed
F-2024-4254 - Incorrect EIP712 Constructor Name	Fixed
F-2024-4260 - Users Lose Their Fees If The LiquidityPool is Empty	Fixed
F-2024-4262 - Non-Standard ERC20 Tokens Result in Loss of Funds	Fixed

Documentation quality

- Functional requirements are provided.
- Technical description is partially provided:
  - The purpose of `SkybridgeERC20` and `SkyBridgeERC721` are not described in the documentation.
  - The project's design architecture is missing.
  - Sytem roles roles are partially described in the documentation.

Code quality

- Best practices are not followed: F-2024-4229.
- The development environment is configured.
- Outdated solidity version: F-2024-4229.
- Unused code present: F-2024-4219, F-2024-4226.

Test coverage

Code coverage of the project is **55.24%** (branch coverage).

- Deployment and basic user interactions are not covered with tests.

- SkyBridge factories and tokens have no tests.
- Test coverage is low in L1AviBridge and L2AviBridge, and particularly low in AviERC721Bridge.
- Negative cases coverage is missed.
- Interactions by several users are not tested thoroughly.

Table of Contents

System Overview

SkyBridge is a bridge solution with the following contracts:

- `SkybridgeERC20` — upgradeable ERC-20 token, which is adopted from the `OptimismMintableERC20`. It has the minting and the burning functionality which is controlled by the bridge contract. It is minted or burned during the bridging operation of the IOptimismMintableERC20 tokens.
- `SkyBridgeERC721` - upgradeable ERC721 token, which is Optimism representation of an Ethereum-based token. It has the minting and burning functionality which is controlled. by the bridge contract. It is minted or burned during the bridging operation of the `IOptimismMintableERC721` tokens.
- `SkybridgeERC20Factory` - Factory contract for creating and deploying `SkybridgeERC20` tokens.
- `SkyBridgeERC721` - Factory contract for creating `OptimismMintableERC721` contracts.
- `L1AviBridge` - The L1AviBridge is responsible for transfering ETH and ERC20 tokens between L1 and L2.
- `L1AviERC721Bridge` - is a contract which works together with the L2AviERC721Bridge to make it possible to transfer ERC721 tokens from Ethereum to Optimism. This contract acts as an escrow for ERC721 tokens deposited into L2.
- `LiquidityPool` - this contract is utilized to provide a central place to hold funds for fast bridging and the fees from Layer 1 to Layer 2 transfers.
- `L2AviBridge` - the contract is responsible for transfering ETH and ERC20 tokens between L1 and L2. In the case that an ERC20 token is native to L2, it will be escrowed within this contract. If the ERC20 token is native to L1, it will be burnt.
- `L2AviERC721Bridge` - is a contract which works together with the L1AviERC721Bridge to make it possible to transfer ERC721 tokens from Base to Ethereum. This contract acts as an escrow for ERC721 tokens deposited into L1.
- `AviPredeploys` - a library that contains hardcoded addresses.
- `AviBridge` - a base abstract contract, for the L1 and L2 standard ERC20 bridges. It handles the core bridging logic, including escrowing tokens that are native to the local chain and minting/burning tokens that are native to the remote chain.
- `AviERC721Bridge` - is. a base contract for the L1 and L2 ERC721 bridges.

Privileged roles

- The `SkybridgeERC20` has one privileged role, which is `BRIDGE`:
  - `BRIDGE` can mint and burn `SkybridgeERC20` tokens
- The `SkyBridgeERC721` has one privileged role, which is `BRIDGE`:
  - `BRIDGE` can safeMint and burn `SkyBridgeERC721` tokens
- Abstract `AviBridge` has the following roles:
  - `DEFAULT_ADMIN_ROLE`, which can:
    - change the recipient of the flat fee via `setFlatFeeRecipient()`
    - change the flat fee via `setFlatFee()`
    - add and remove admins via `addAdmin()` and `removeAdmin()`
    - add and remove accounts from the `PAUSER_ROLE` via `addPauser()` and `removePauser()`
    - change the address of the backendUser via `setBackend()`
  - `onlyOtherBridge`, which can:
    - finalize the bridging operations via `finalizeBridgeETH()` and `finalizeBridgeERC20()`
- Abstract `AviERC721Bridge` has the following roles:
  - `DEFAULT_ADMIN_ROLE`, which can:
    - change the flat fee via `setFlatFee()`
    - add and remove admins via `addAdmin()` and `removeAdmin()`
    - add and remove accounts from the `PAUSER_ROLE` via `addPauser()` and `removePauser()`

        - change the address of the backendUser via `setBackend()`
    - onlyEOA addresses, which can:
        - use `bridgeERC721()`
- The `L1AviBridge` contract inherits from the abstract `AviBridge`'s functionality and has the additional privileged roles:
    - `DEFAULT_ADMIN_ROLE`, which can:
        - set the bridging fee via `setBridgingFee()`
        - set the address of the other bridge via `setOtherBridge()`
        - set the address of the AVI token via `setAviTokenAddress()`
    - `onlyPauserOrAdmin`, which can:
        - pause the bridge's functionality via `setPaused()`
    - `onlyEOA`, which can:
        - transfer eth via `receive()` function
        - use the `bridgeETH()` functionality to bridge the ETH tokens
        - use the `bridgeERC20()` functionality to bridge ERC20 tokens
- The `L1AviERC721Bridge` contract inherits from the `AviERC721Bridge`'s functionality and has the additional privileged roles:
    - `DEFAULT_ADMIN_ROLE`, which can:
        - set the address of the other bridge via `setOtherBridge()`
        - finalize the bridging of ERC721 via `finalizeBridgeERC721()`
    - `onlyPauserOrAdmin`, which can:
        - pause the bridge's functionality via `setPaused()`
- The `L2AviBridge` contract inherits from the `AviBridge`'s functionality and has the additional privileged roles:
    - `DEFAULT_ADMIN_ROLE`, which can:
        - add and remove the allowed tokens via `addAllowedToken()` and `removeAllowedToken()` functions
    - `onlyEOA`, which can:
        - initiate the withdrawal from L2 to L1 via `withdraw()` or `receive()`
- The `L2AviERC721Bridge` contract inherits from the `AviERC721Bridge`'s functionality and has the additional privileged roles:
    - `DEFAULT_ADMIN_ROLE`, which can:
        - finalize the bridging of ERC721 via `finalizeBridgeERC721()`
- The `LiquidityPool` has the following roles:
    - `DEFAULT_ADMIN_ROLE`:
        - can add and remove admins via `addAdmin()` and `removeAdmin()` functions
        - can withdraw ERC20 and ETH via withdraw functions
        - add and remove the bridge role via addBridge() and removeBridge() functions.
    - `BRIDGE_ROLE`:
        - can send ETH and ERC20 tokens via `sendETH()` and `sendERC20()` functions.

Risks

- The architecture of some contracts including `L1AviBridge`, `L1AviERC721Bridge`, `L2AviBridge`, `LiquidityPool,` `L2AviERC721Bridge,` `SkyBridgeERC20Factory` and `SkyBridgeERC721Factory` relies on administrative keys for critical operations. Centralized control over these keys presents a significant security risk, as compromise or misuse can lead to unauthorized actions or loss of funds. It is recommended to use a multi-signature wallet with a proper minimal signatures threshold.
- The contracts `L1AviBridge` and `L2AviBridge` provide users with the capability to specify a custom `gasLimit` for the transfer of Ether or ERC20 tokens. However, if the `gasLimit` is set incorrectly, it poses a risk that transactions may become indefinitely stuck in pending status or be reverted.
- The protocol's contracts, including `L1AviERC721Bridge` and `AviBridge`, incorporate functionality that can be paused, such as `finalizeBridgeERC721()`, `finalizeBridgeETH()`, and `finalizeBridgeERC20()`. Pausing these finalization functions creates a risk that deposit or withdrawal transactions may not revert, potentially leading to the loss of users' funds.
- Hardcoded addresses in the `AviPredeploys` library and the `OTHER_BRIDGE` address from the `L2AviBridge` contract, which cannot be changed after its initialization, create a risk of issues if one of the aforementioned addresses will be changed in the future, but the Bridge contracts will not have any setter mechanism to update it.
- The protocol is relying on the out-of-scope and off-chain functionality to handle its operations, which creates a risk of introducing new security concerns, due to the fact that this part of the project is out-of-scope, hence was not deeply reviewed within the current security assessment: `CrossDomainMessenger`, `SafeCall`, `OptimismMintableERC20`, `Predeploys`, `IOptimismMintableERC721`, `ILegacyMintableERC20`.
- The SkyBridge team is planning to add new chains to the current SkyBridge solution in the future, which can potentially introduce new security risks to the system.
- The `SkyBridgeERC20` token contract contains the function `burn()` to burn users' tokens. However, it lacks the `_spendAllowance()` function call, which means that the bridge calling `SkyBridgeERC20::burn()` can burn users' tokens without their approval. It creates a security risk that the `BRIDGE` will contain the address of other entity not related to the bridge, which will be able to burn users' tokens at its will.
- The project's contracts `SkybridgeERC20` and `SkyBridgeERC721` are upgradable, allowing the administrator to update the contract logic at any time. While this provides flexibility in addressing issues and evolving the project, it also introduces risks if upgrade processes are not properly managed or secured, potentially allowing for unauthorized changes that could compromise the project's integrity and security.
- The testing of various contracts and workflows within the project is insufficient, leading to low reliability. It is crucial to emphasize that comprehensive testing is an essential component of any project. Failing to achieve complete test coverage poses significant risks and reduces the project's reliability.
- The address of the `flatFeeRecipient` in the `L1AviBridge` contract should be an independent address according to the project's documentation. However, this address is setup to the same address as the LiquidityPool in the contract `constructor()`. As a consequence, the function `_initiateETHDeposit()` performs two calls to the same address, duplicating the transactions. Furthermore, the malicious address can be set as a `flatFeeRecipient` or `LIQUIDITY_POOL`, which can create additional security risks to the system.
- The protocol has no methods to track users' assets and allow tokens to be retrieved in case of failure. There can be several reasons why the bridge operation may not succeed on the destination chain, and assets become stuck: native tokens being sent to a contract that cannot handle them, transactions not meeting the requirements criteria in the destination chain, lack of liquidity in the liquidity pool.
- Tokens can be retrieved from the `LiquidityPool` contract by the `DEFAULT_ADMIN_ROLE` via `withdrawERC20()` and `withdrawETH()`. Although this is a trusted address, the scenario is still possible and should be noted.
- ERC721 tokens can be bridged into the protocol. Users should be aware that ERC721 are unique assets with specific properties and data, and they rely on the Aviator protocol to bridge the token features correctly.

- The contract `AviPredeploys` contain hardcoded addresses that should be updated during project deployment.
- Although the flat fee and the liquidity fee have a predefined value described in the documentation, the protocol has the capability to change these values.
- The base contracts `AviBridge` and `AviERC721Bridge` should introduce storage gaps. When working with upgradeable contracts, it is necessary to introduce storage gaps to base contracts to allow a storage extension during upgrades. Storage gaps are a convention for reserving storage slots in a base contract, allowing future versions of that contract to use up those slots without affecting the storage layout of child contracts. Without a storage gap, the variable in the child contract might be overwritten by the upgraded base contract if new variables are added to the base contract. This could have unintended and very serious consequences for the child contracts.
- The following contracts lack the invocation of `_disableInitializers` in the `constructor`: `L1AviBridge`, `L2AviBridge`, `L1AviERC721Bridge`, `L2AviERC721Bridge`. Leaving an upgradeable smart contract uninitialized may lead to a takeover of the implementation contract, which could result in several undesired side effects.
- The state variables `REMOTE_TOKEN`, `BRIDGE`, `DECIMALS` and `REMOTE_CHAIN_ID` defined in the contracts `SkyBridgeERC20` and `SkyBridgeERC721` should be set as `immutable` to save gas.
- The Zero Address and Zero Value checks introduced in `SkyBridgeERC721`'s `constructor()` are redundant since those addresses are originated in the corresponding factory, where they are already checked.
- The returned addresses `newSkybridgeERC721` and `newSkyBridgeERC20` should be checked not to be the Zero Address (0x0) to make sure the contract was created correctly.
- The state variables `L1AviERC721Bridge::_isPaused` and `AviBridge::_numAdmins` will not be initialized to the defined values since they belong to an implementation contract. If said variables should have any value, they should be setup in the contract's initialization function.
- Using the `Ownable` instead of `Ownable2Step` in the `SkyBridgeERC20Factory` and `SkyBridgeERC721Factory` creates a risk that the contract ownership will mistakenly be transferred to an address that cannot handle it.
- If the deployment and the initialization of the implementation contracts, which are `L1AviBridge`, `L1AviERC721Bridge`, `L2AviBridge` and `L2AviERC721Bridge`, will not be handled within one transaction, it can cause the frontrunning of the `initialize()` functions by the attacker which will lead to unexpected system behavior.
- The project's contracts `AviBridge` and `AviERC721Bridge` are upgradable, allowing the administrator to update the contract logic at any time. While this provides flexibility in addressing issues and evolving the project, it also introduces risks if upgrade processes are not properly managed or secured, potentially allowing for unauthorized changes that could compromise the project's integrity and security.
- The base upgradeable contracts `AviBridge` and `AviERC721Bridge` lack the upgradeable import `AccessControlUpgradeable`, which creates a risk of problems in the future due to the missing gaps. Moreover, the `initialize()` functions of `AviBridge` and `AviERC721Bridge` should call the `AccessControlUpgradeable::__AccessControl_init()` function of the parent contract `AccessControlUpgradeable`.
- Public salt parameters of the `SkyBridgeERC20Factory::createSkyBridgeERC20()` and `SkyBridgeERC721Factory::createSkyBridgeERC721()` functions create a risk that external actors will use this data and deploy a contract to the generated address in advance.
- The system lacks the whitelisting for both ERC20 and ERC721 bridge operations, which can potentially break the system if malicious tokens are used.

Findings

Vulnerability Details

F-2024-4214 - Uninitialized flatFeeRecipient Leads to Loss of Fees - High

Description:

The `flatFeeRecipient` state variable is the address receiving the `flatFee` from the bridge operations. However, it is never initialized. As a consequence, the fees are sent to the `address(0)` and cannot be recovered.

The parameter `flatFeeRecipient` is used in the internal function `L1AviERC721Bridge::_initiateBridgeERC721()`, which is called in `L1AviERC721Bridge::bridgeERC721()` and `bridgeERC721To()`. The following piece of code shows the usage of the zero address `flatFeeRecipient`.

```
function _initiateBridgeERC721(
    address _localToken,
    address _remoteToken,
    address _from,
    address _to,
    uint256 _tokenId,
    bytes calldata _extraData
)
    internal
    override
{
    require(_remoteToken != address(0), "L1ERC721Bridge: remote token cannot be address(0)")
    require(msg.value == flatFee, "L1ERC721Bridge: bridging ERC721 must include sufficient E
    (bool sent, ) = payable(flatFeeRecipient).call{value: msg.value}("");
    require(sent, "L1ERC721Bridge: failed to send ETH to fee recipient");
    // Lock token into bridge
    deposits[_localToken][_remoteToken][_tokenId] = true;
    IERC721(_localToken).transferFrom(_from, address(this), _tokenId);
    // Send calldata into L2
    emit ERC721BridgeInitiated(_localToken, _remoteToken, _from, _to, _tokenId, _extraData);
}
```

As can be seen from the code snippet above, the function `_initiateBridgeERC721()` contains the transfer of `flatFee` value, which occurs every time the ERC721 token is bridged between chains.

Assets:

- L1AviERC721Bridge.sol [https://github.com/AviatorAC/skybridge]

Status: Fixed

Classification

Impact: 4/5

Likelihood:	5/5
Exploitability:	Independent
Complexity:	Simple
Severity:	High

Recommendations

Remediation: It is recommended to add the necessary functionality to enable the admin to change the address of the flat fee recipient after the contract's deployment is over. In addition, a check should be introduced to avoid sending tokens to the zero address.

It can also be considered to, initialize the fee receiver in the `constructor()` of the contract.

Resolution: Fixed in commit ID `a53e33d`. `flatFeeRecipient` is now set in the `L1AviERC721Bridge::constructor()`

Evidences

POC

Reproduce:

1. Alice wants to move her `ERC721` token to the L2 chain and calls `L1AviERC721Bridge:: bridgeERC721()` function.
2. The `L1AviERC721Bridge:: bridgeERC721()` is not only responsible for transferring ERC721 cross-chain via emitting events, but also for collecting the `flatFee` fees and transferring it to the `flatFeeRecipient`. However, the `flatFeeRecipient` is set to address(0), hence all of the `flatFee` fees are go to the address(0), thus lost.

F-2024-4250 - Incorrect Signature Validation Will Revert Execution - High

Description:

The function `withdraw()` uses an incorrect signature validation mechanism in order to execute supersonic bridging.

The validation function is shown below:

```
function withdraw(
    AviFastWithdrawal calldata _txn,
    bytes calldata _signature
) external {
    require(backendUser != address(0), "invalid backend user");

    (bytes32 r, bytes32 s, uint8 v) = splitSignature(_signature);

    bytes32 structHash = keccak256(abi.encode(_WITHDRAWAL_TYPEHASH,_txn.l1_token,_txn.l2_tok
    address signer = ECDSA.recover(_hashTypedDataV4(structHash), v, r, s);

    require(signer == backendUser, "invalid signature");
    require(fastBridgeNonces[_txn.to] == _txn.nonce, "invalid nonce");

    fastBridgeNonces[_txn.to] += 1;

    // Transfer the tokens, ETH if the l1_token is address(0)
    if (_txn.l1_token == address(0)) {
        LIQUIDITY_POOL.sendETH(_txn.to, _txn.amount);
    } else {
        LIQUIDITY_POOL.sendERC20(_txn.to, _txn.l1_token, _txn.amount);
    }

    bool success = SafeCall.call(optimismPortal, gasleft(), 0, _txn.proof_transaction);
    require(success, "failed to call optimism portal");
}
```

In order to the validation, `withdrawal()` compares the signers of the input `_signature` and the computed `structHash`. However, the provided `_signature` was generated from `AviFastWithdrawal` whilst the `structHash` was obtained from `_WITHDRAWAL_TYPEHASH`, which does not contain the parameter `proof_transaction`.

```
struct AviFastWithdrawal {
    address l1_token;
    address l2_token;
    address from;
    address to;
    uint256 amount;
    uint256 nonce;

    bytes proof_transaction;
}
```

```
bytes32 private constant _WITHDRAWAL_TYPEHASH = keccak256("Message(address l1_token,address
```

Since the signature depends on the content of the hash, the comparison of signers implemented as `require(signer == backendUser, "invalid signature")` cannot be fulfilled, reverting the transaction.

As a consequence, the function `withdraw()` will never succeed to execute, resulting in a revert of the Supersonic bridge feature of the system. In addition, due to the lack of a mechanism to return the funds to the user that triggered the Supersonic bridging, the user funds will be lost.

Assets:

- L1AviBridge.sol [https://github.com/AviatorAC/skybridge]

Status: Fixed

Classification

Impact: 4/5

Likelihood: 5/5

Exploitability: Independent

Complexity: Simple

Severity: High

Recommendations

Remediation: Include the parameter `proof_transaction` within `_WITHDRAWAL_TYPEHASH` and the `structHash` calculation in the method `withdraw()`.

Resolution: Fixed in commit ID `d974780`: `proof_transaction` was added into `_WITHDRAWAL_TYPEHASH` and the `structHash` calculation in `fastWithdraw()`, becoming compliant with the `AviFastWithdrawl` struct.

```
bytes32 private constant _WITHDRAWAL_TYPEHASH = keccak256("Message(address l1_token,address

function fastWithdraw(
    AviFastWithdrawal calldata _txn,
    bytes calldata _signature
)
    external
    payable
{
    ...
    bytes32 structHash = keccak256(abi.encode(_WITHDRAWAL_TYPEHASH, _txn.l1_token, _txn.l2_t
    ...
}
```

F-2024-4213 - Missing Zero Address Validation Can Lead To Loss of Funds - Low

Description:

In Solidity, the Ethereum address `0x0000000000000000000000000000000000000000` is known as the zero address. This address has significance because it is the default value for uninitialized address variables and is often used to represent an invalid or non-existent address.

The following address parameters are used without checking against the possibility of being `0x0`. This can lead to unwanted external calls to `0x0`:

- `SkybridgeERC20::initialize() → _remoteToken, _bridge`
- `L1AviBridge::constructor() → optimismPortal`
- `L1AviERC721Bridge::constructor() → _liquidityPool, setOtherBridge() → _otherBridge`
- `SkyBridgeERC721::initialize() → _bridge, _remoteToken`
- `AviERC721Bridge:: constructor() → _otherBridge`

In the following cases, lack of zero address checks for the `_to` input parameter can result in a loss of funds:

- `L1AviBridge::bridgeETHTo(), bridgeERC20To()`
- `AviBridge.sol::finalizeBridgeETH(), finalizeBridgeERC20()`
- `L1AviERC721Bridge::finalizeBridgeERC721()`
- `L2AviBridge::withdrawTo(), fastWithdrawTo(), finalizeDeposit()`
- `L2AviERC721Bridge::finalizeBridgeERC721()`

Assets:

- L1AviBridge.sol [https://github.com/AviatorAC/skybridge]
- L1AviERC721Bridge.sol [https://github.com/AviatorAC/skybridge]
- LiquidityPool.sol [https://github.com/AviatorAC/skybridge]
- L2AviBridge.sol [https://github.com/AviatorAC/skybridge]
- L2AviERC721Bridge.sol [https://github.com/AviatorAC/skybridge]
- SkyBridgeERC20.sol [https://github.com/AviatorAC/skybridge]
- SkyBridgeERC721.sol [https://github.com/AviatorAC/skybridge]
- AviBridge.sol [https://github.com/AviatorAC/skybridge]

Status: Fixed

Classification

Impact: 3/5

Likelihood: 2/5

Exploitability: Dependent

Complexity: Simple

Severity: Low

Recommendations

Remediation: Consider adding zero address validation for the addresses mentioned above.

Resolution: Fixed in commit ID `d974780`: zero address checks were implemented for the reported addresses.

F-2024-4217 - Public AccessControl Functionality Allows Bypassing of Checks - Low

Description:

The condition `_numAdmins > 1` is implemented in the `removeAdmin()` function of the contracts `LiquidityPool`, `AviBridge` and `AviERC721Bridge` so the `ADMIN_ROLE` of the contracts keeps at least one admin.

It can be seen in the `removeAdmin()` function in the code snippet below:

```
/// @notice Remove an admin from the list of admins.
/// @param _admin Address to remove.
function removeAdmin(address _admin) external onlyRole(DEFAULT_ADMIN_ROLE) {
    require(hasRole(DEFAULT_ADMIN_ROLE, _admin), "Address is not a recognized admin.");
    require (_numAdmins > 1, "Cannot remove the only admin.");
    _revokeRole(DEFAULT_ADMIN_ROLE, _admin);
    _numAdmins--;
    emit AdminRemoved(_admin);
}
```

The above condition can be bypassed due to the inheritance of the `AccessControl` contract, which enables admins to renounce their roles via the `renounceRole()` and `revokeRole()` functionality

```
/**
 * @dev Revokes `role` from the calling account.
 *
 * Roles are often managed via {grantRole} and {revokeRole}: this function's
 * purpose is to provide a mechanism for accounts to lose their privileges
 * if they are compromised (such as when a trusted device is misplaced).
 *
 * If the calling account had been revoked `role`, emits a {RoleRevoked}
 * event.
 *
 * Requirements:
 *
 * - the caller must be `callerConfirmation`.
 *
 * May emit a {RoleRevoked} event.
 */
function renounceRole(bytes32 role, address callerConfirmation) public virtual {
    if (callerConfirmation != _msgSender()) {
        revert AccessControlBadConfirmation();
    }
    _revokeRole(role, callerConfirmation);
}
```

As a consequence, all `ADMIN_ROLE`s of the `LiquidityPool`, `AviBridge`, and `AviERC721Bridge` contracts can be removed, leading to the situation where admin functionality becomes unavailable and unusable. The Proof of Concept of the previously mentioned problem can be found below.

Assets:

- universal/AviERC721Bridge.sol [https://github.com/AviatorAC/skybridge]
- LiquidityPool.sol [https://github.com/AviatorAC/skybridge]
- AviBridge.sol [https://github.com/AviatorAC/skybridge]

Status: Fixed

Classification

Impact: 4/5

Likelihood: 2/5

Exploitability: Dependent

Complexity: Medium

Severity: Low

Recommendations

Remediation: It is recommended to disable the `renounceRole()` and `revokeRole()` functions from the `AccessRole` contract by overriding to ensure that the necessary contract's requirements are not violated.

Resolution: This finding was fixed in commit ID `d974780`: the methods `renounceRole()` and `revokeRole()` were implemented in order to revert all calls:

```
function renounceRole(bytes32, address) public pure override {
    revert("LiquidityPool: renounceRole is disabled, use removeAdmin to remove yourself as a
}

function revokeRole(bytes32 role, address account) public virtual override onlyRole(getRoleA
    revert("AviBridge: revokeRole is disabled, use removeAdmin to remove yourself as an admi
}
```

Evidences

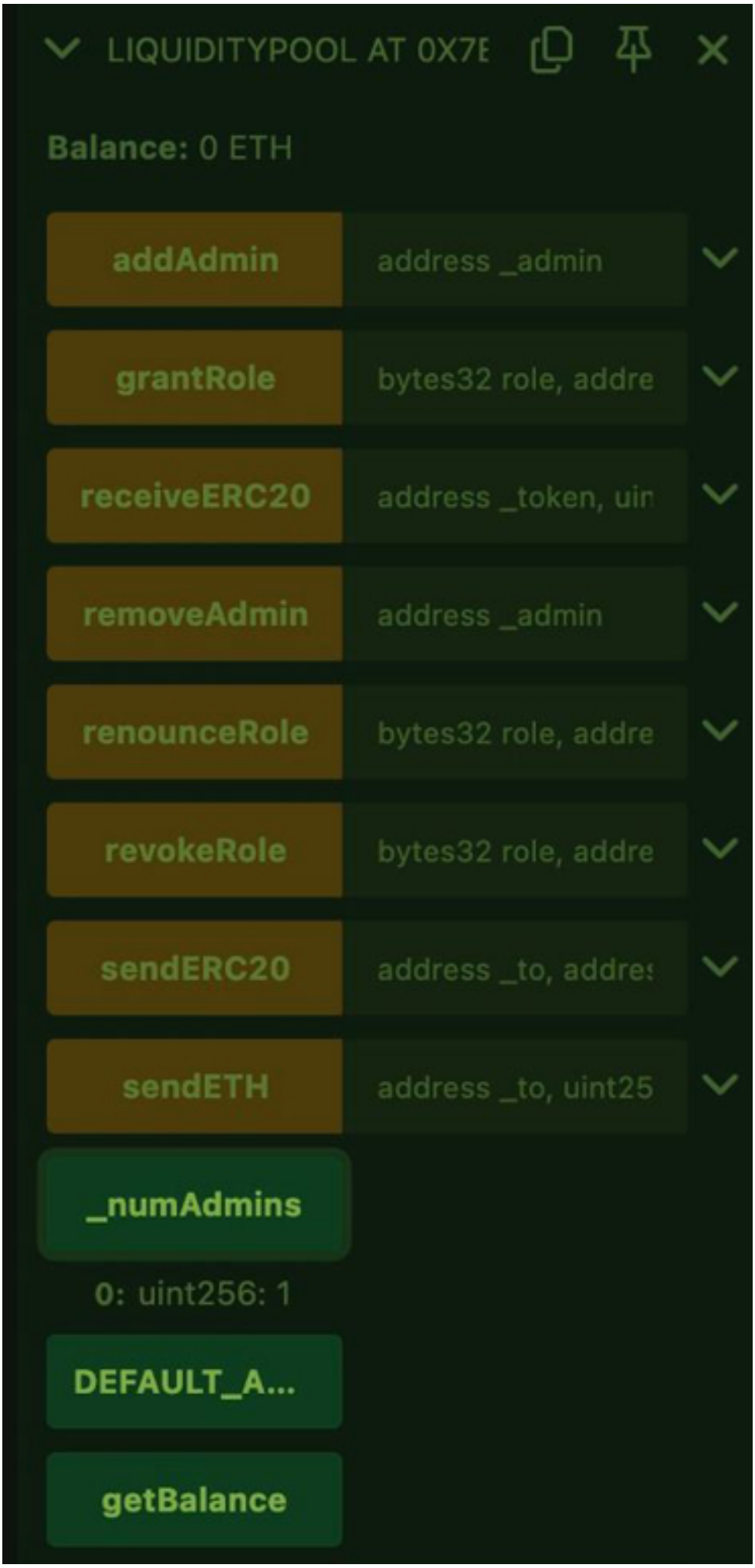
PC

Reproduce:

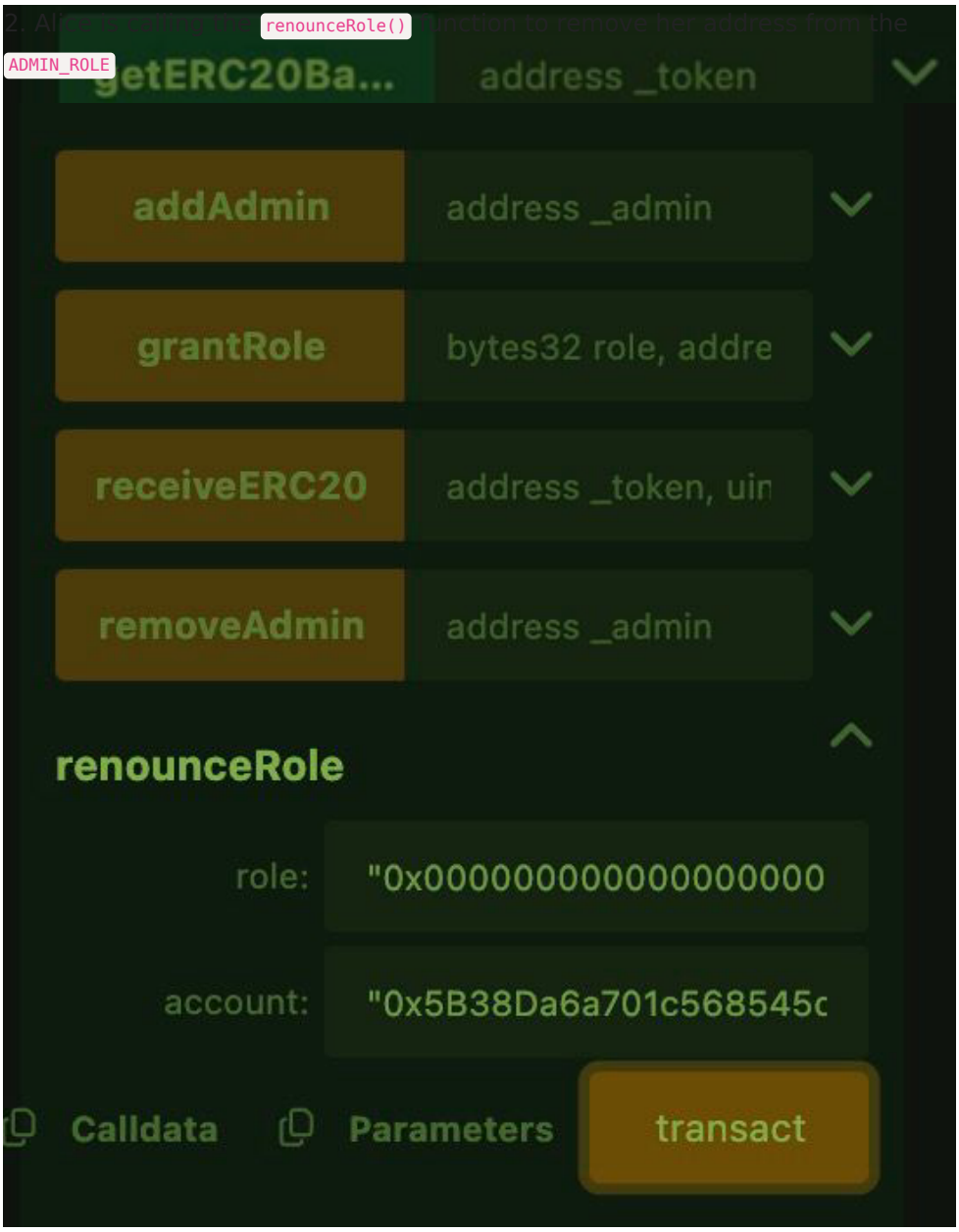
1. Alice, who is the deployer of the `LiquidityPool`, is deploying the contract.
2. The number of admins is increased in the contract's `constructor()`, which can be seen and stored at the `_numAdmins`.
3. Alice is calling `AccessControl::renounceRole()` to delete her address from the list of addresses attached to the `ADMIN_ROLE`.
4. Now the `LiquidityPool` is missing any addresses for the `ADMIN_ROLE`, which makes the `ADMIN_ROLE` functionality unusable.

Results:

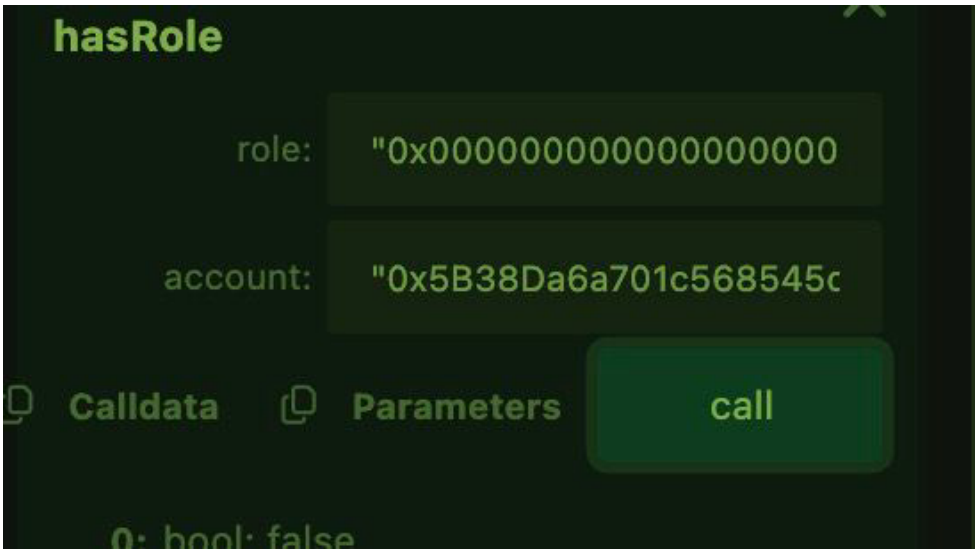
1. Number of admins after the deployment is 1.


LIQUIDITYPOOL AT 0X7E
Balance: 0 ETH
addAdmin
address _admin
grantRole
bytes32 role, addre
receiveERC20
address _token, uin
removeAdmin
address _admin
renounceRole
bytes32 role, addre
revokeRole
bytes32 role, addre
sendERC20
address _to, addres
sendETH
address _to, uint25
_numAdmins
0: uint256: 1
DEFAULT_A...
getBalance

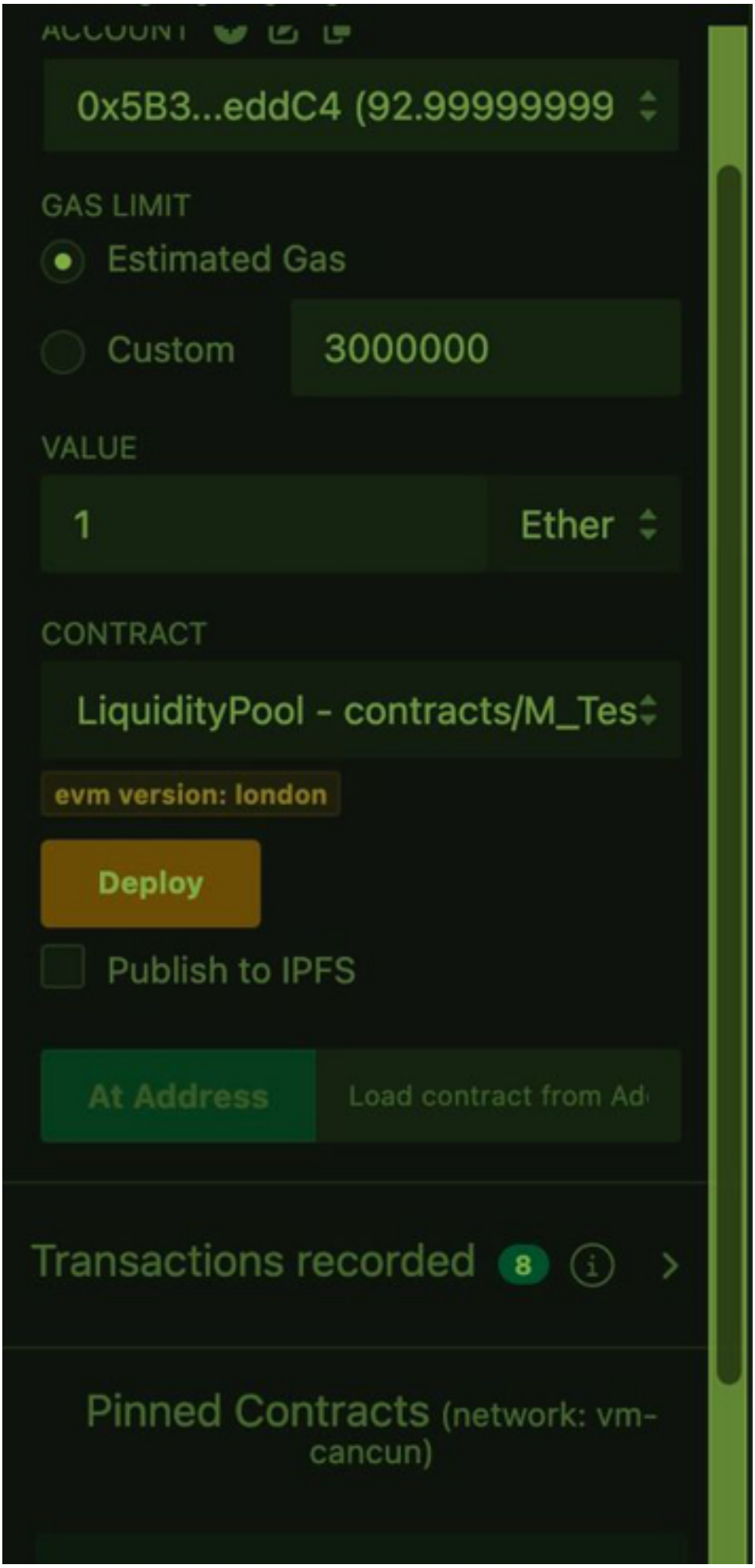
2. Alice is calling the `renounceRole()` function to remove her address from the `ADMIN_ROLE`

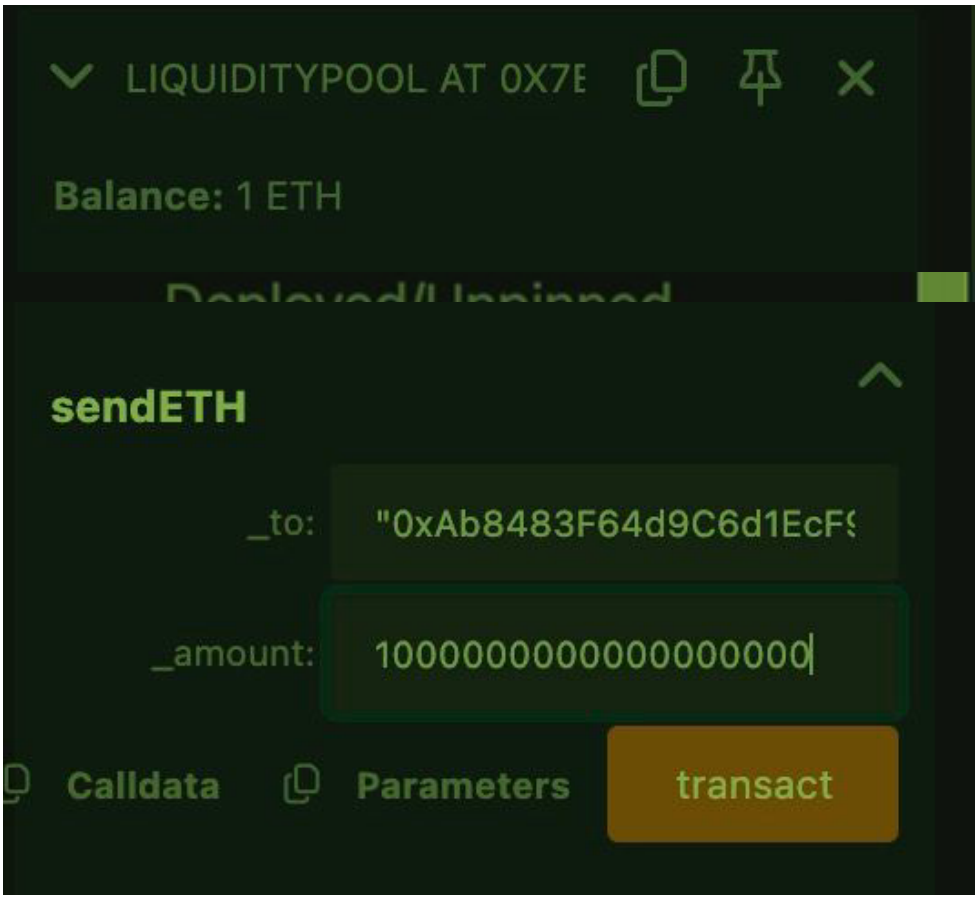


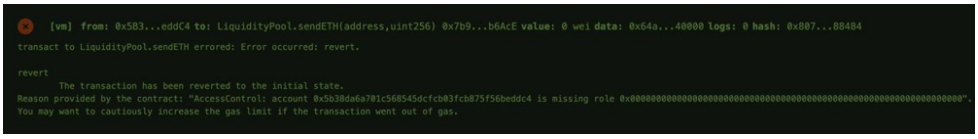
3. The number of admins stored at `_numAdmins` remains the same, since the variable is not decreased in the `renounceRole()` function. However, Alice now lacks the `ADMIN_ROLE`, which can be seen from the picture below where the hasRole() is returning false.



4. If Alice transfers 1 ether to the `LiquidityPool` and will try to withdraw it, the transaction will revert since the contract does not have any admins. It is demonstrated below.


ACCOUNT
0x5B3...eddC4 (92.99999999
GAS LIMIT
Estimated Gas
Custom
3000000
VALUE
1
Ether
CONTRACT
LiquidityPool - contracts/M_Tes
evm version: london
Deploy
Publish to IPFS
At Address
Load contract from Ad
Transactions recorded
8
Pinned Contracts (network: vm-cancun)


LIQUIDITYPOOL AT 0X7E
Balance: 1 ETH
sendETH
_to:
"0xAb8483F64d9C6d1EcF
_amount:
100000000000000000
Calldata
Parameters
transact


[vm] from: 0x5B3...eddC4 to: LiquidityPool.sendETH(address,uint256) 0x7b9...b6AcE value: 0 wei data: 0x64a...40000 logs: 0 hash: 0x807...88484
transact to LiquidityPool.sendETH errored: Error occurred: revert.
revert
The transaction has been reverted to the initial state.
Reason provided by the contract: "AccessControl: account 0x5b38da6a701c568545dcfcb03fcb875f56beddc4 is missing role 0x0000000000000000000000000000000000000000000000000000000000000000".
You may want to cautiously increase the gas limit if the transaction went out of gas.

F-2024-4225 - Lack of Refund Functionality Leads To Loss Of Funds - Low

Description:

The function `_initiateWithdrawal()`, which is used in the `receive()`, `withdraw()`, `withdrawTo()` and `fastWithdrawTo()` has the condition `msg.value >= requiredTotal`, where `uint256 requiredTotal = _amount + flatFee`.

The condition ensures that the number of assets sent for the withdrawal is **greater than or equal** to the `flatFee` and the amount to be withdrawn. Moreover, `L1AviBridge::_initiateETHDeposit()` has the same problem.

```
/// @custom:legacy
/// @notice Internal function to initiate a withdrawal from L2 to L1 to a target account on
/// @param _l2Token     Address of the L2 token to withdraw.
/// @param _from        Address of the withdrawer.
/// @param _to          Recipient account on L1.
/// @param _amount      Amount of the L2 token to withdraw.
/// @param _minGasLimit Minimum gas limit to use for the transaction.
/// @param _extraData   Extra data attached to the withdrawal.
function _initiateWithdrawal(
    address _l2Token,
    address _from,
    address _to,
    uint256 _amount,
    uint32 _minGasLimit,
    bool _fastBridge,
    bytes memory _extraData
)
    internal
{
    if (_fastBridge) {
        require(allowedTokens[_l2Token], "L2AviBridge: token not allowed to be fast bridged"
        // For ETH, _amount is the amount of ETH to send to the user, and we need it + the f
        if (_l2Token == Predeploys.LEGACY_ERC20_ETH) {
            uint256 requiredTotal = _amount + flatFee;
            require(msg.value >= requiredTotal, "L2AviBridge: insufficient ETH value");
        } else {
            // ERC20 tokens are just the amount of tokens to send to the user, so we require
            require(msg.value == flatFee, "L2AviBridge: insufficient ETH value");
        }
    }
    address _trueTo = _to;
    if (_fastBridge) {
        // Send the flat fee to the l1FeeRecipient, so we can use it to make the transfer fa
        _initiateBridgeETH(_from, flatFeeRecipient, flatFee, _minGasLimit, "");
        // Send the slow bridged tokens to our liquidity pool
        _trueTo = payable(liquidityPool);
       }
    if (_l2Token == Predeploys.LEGACY_ERC20_ETH) {
        _initiateBridgeETH(_from, _trueTo, _amount, _minGasLimit, _extraData);
        _emitFastWithdrawlInitiated(address(0), _l2Token, _from, _to, _amount, _fastBridge,
    } else {
        address l1Token = OptimismMintableERC20(_l2Token).remoteToken();
```

```
        _initiateBridgeERC20(_l2Token, l1Token, _from, _trueTo, _amount, _minGasLimit, _extr
        _emitFastWithdrawlInitiated(l1Token, _l2Token, _from, _to, _amount, _fastBridge, _ex
    }
}
```

However, the scenario where the Ether amount exceeds the `requiredTotal` is not properly addressed: any excess Ether should be returned to the user, somehow. Consequently, the surplus Ether remains locked within the contract.

Assets:

- L1AviBridge.sol [https://github.com/AviatorAC/skybridge]
- L2AviBridge.sol [https://github.com/AviatorAC/skybridge]

Status: Fixed

Classification

Impact: 2/5

Likelihood: 2/5

Exploitability: Independent

Complexity: Medium

Severity: Low

Recommendations

Remediation: Consider implementing a refund mechanism for excess Ether or replacing the `>=` operator with `==` in `_initiateWithdrawal()`.

Resolution: Fixed in commit ID `a53e33d`: Fees were removed from the `L2AviBridge.sol`.

F-2024-4254 - Incorrect EIP712 Constructor Name - Low

Description:

The contract `L1AviBridge`'s `constructor()` calls `EIP712`'s `constructor()` too, which requires the following parameters:

```
/**
 * @dev Initializes the domain separator and parameter caches.
 *
 * The meaning of `name` and `version` is specified in
 * https://eips.ethereum.org/EIPS/eip-712#definition-of-domainseparator[EIP 712]:
 *
 * - `name`: the user readable name of the signing domain, i.e. the name of the DApp or the
 * - `version`: the current major version of the signing domain.
 *
 * NOTE: These parameters cannot be changed except through a xref:learn::upgrading-smart-cor
 * contract upgrade].
 */
constructor(string memory name, string memory version) {
    bytes32 hashedName = keccak256(bytes(name));
    bytes32 hashedVersion = keccak256(bytes(version));
    bytes32 typeHash = keccak256(
        "EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)"
    );
    _HASHED_NAME = hashedName;
    _HASHED_VERSION = hashedVersion;
    _CACHED_CHAIN_ID = block.chainid;
    _CACHED_DOMAIN_SEPARATOR = _buildDomainSeparator(typeHash, hashedName, hashedVersion);
    _CACHED_THIS = address(this);
    _TYPE_HASH = typeHash;
}
```

As indicated by `EIP712`'s NatSpec, the `name` parameter corresponds to the name of the signing domain. Nevertheless, the contract introduces its own name rather than the signing domain name:

```
constructor(address payable _liquidityPool, address _token, address payable _optimismPortal
```

If the signing domain name does not match the recorded name, the signature verification may not work correctly,

Assets:

- L1AviBridge.sol [https://github.com/AviatorAC/skybridge]

Status: Fixed

Classification

Impact: 2/5

Likelihood: 3/5

Exploitability: Dependent

Complexity: Medium

Severity: Low

Recommendations

Remediation: Use the signing domain's name to setup the `EIP712` constructor instead of the contract name.

Resolution: Fixed in commit ID `a53e33d`: the EIP712 domain name defined in the constructor was updated to `Skybridge`. However, it should be noted that the signer is off-chain and cannot be validated within this audit scope, thus relying on the development team to use the correct name.

F-2024-4260 - Users Lose Their Fees If The LiquidityPool is Empty - Low

Description:

The development team provided the following information regarding Supersonic Bridge withdrawals:

> In the event the LiquidityPool does not have sufficient funds to cover a user's withdraw, we expect the transaction to fail execution and the user will be required to use the standard withdrawal instead

Thus if the `LIQUIDITY_POOL` does not have enough assets, the `withdraw()` function will revert. As a consequence, the fees the user paid for such the supersonic withdrawal will be lost. Users will then need to trigger a new withdrawal via Legacy Bridge, paying additional fees and waiting for up to 7 days.

This will affect user experience and result in a loss of funds.

The function `L1AviBridge::withdraw()` can be found below. This function includes `LIQUIDITY_POOL's` function `sendETH()` and `sendERC20()`, which will revert if no funds are available.

```
/// @notice Withdraw ETH from the bridge.
/// @param _txn The signed transation message
/// @param _signature The signature of the transaction
function withdraw(
    AviFastWithdrawal calldata _txn,
    bytes calldata _signature
) external {
    require(backendUser != address(0), "invalid backend user");

    (bytes32 r, bytes32 s, uint8 v) = splitSignature(_signature);
    bytes32 structHash = keccak256(abi.encode(_WITHDRAWAL_TYPEHASH,_txn.l1_token,_txn.l2_tok
    address signer = ECDSA.recover(_hashTypedDataV4(structHash), v, r, s);
    require(signer == backendUser, "invalid signature");
    require(fastBridgeNonces[_txn.to] == _txn.nonce, "invalid nonce");
    fastBridgeNonces[_txn.to] += 1;
    // Transfer the tokens, ETH if the l1_token is address(0)
    if (_txn.l1_token == address(0)) {
    LIQUIDITY_POOL.sendETH(_txn.to, _txn.amount);
} else {
    LIQUIDITY_POOL.sendERC20(_txn.to, _txn.l1_token, _txn.amount);
}
    bool success = SafeCall.call(optimismPortal, gasleft(), 0, _txn.proof_transaction);
    require(success, "failed to call optimism portal");
}
```

The function `L2AviBridge::_initiateWithdrawal()`, which is triggered for all withdrawals can be found below. The code snippet below clearly demonstrates the payment of the flatFee fees, which users pay for using the fast bridge withdrawal option. However, if the `LiquidityPool` is empty, their fees are not recovered.

```
/// @custom:legacy
/// @notice Internal function to initiate a withdrawal from L2 to L1 to a target account on
/// @param _l2Token     Address of the L2 token to withdraw.
/// @param _from        Address of the withdrawer.
/// @param _to          Recipient account on L1.
/// @param _amount      Amount of the L2 token to withdraw.
/// @param _minGasLimit Minimum gas limit to use for the transaction.
/// @param _extraData   Extra data attached to the withdrawal.
function _initiateWithdrawal(
    address _l2Token,
    address _from,
    address _to,
    uint256 _amount,
    uint32 _minGasLimit,
    bool _fastBridge,
    bytes memory _extraData
)
    internal
{
    if (_fastBridge) {
        require(allowedTokens[_l2Token], "L2AviBridge: token not allowed to be fast bridged"
        // For ETH, _amount is the amount of ETH to send to the user, and we need it + the f
        if (_l2Token == Predeploys.LEGACY_ERC20_ETH) {
            uint256 requiredTotal = _amount + flatFee;
            require(msg.value >= requiredTotal, "L2AviBridge: insufficient ETH value");
        } else {
            // ERC20 tokens are just the amount of tokens to send to the user, so we require
            require(msg.value == flatFee, "L2AviBridge: insufficient ETH value");
        }
    }
    address _trueTo = _to;
    if (_fastBridge) {
        // Send the flat fee to the l1FeeRecipient, so we can use it to make the transfer fa
        _initiateBridgeETH(_from, flatFeeRecipient, flatFee, _minGasLimit, "");
        // Send the slow bridged tokens to our liquidity pool
        _trueTo = payable(liquidityPool);
    }
    if (_l2Token == Predeploys.LEGACY_ERC20_ETH) {
        _initiateBridgeETH(_from, _trueTo, _amount, _minGasLimit, _extraData);
        _emitFastWithdrawlInitiated(address(0), _l2Token, _from, _to, _amount, _fastBridge,
    } else {
        address l1Token = OptimismMintableERC20(_l2Token).remoteToken();
        _initiateBridgeERC20(_l2Token, l1Token, _from, _trueTo, _amount, _minGasLimit, _extr
        _emitFastWithdrawlInitiated(l1Token, _l2Token, _from, _to, _amount, _fastBridge, _ex
    }
}
```

Assets:

- L1AviBridge.sol [https://github.com/AviatorAC/skybridge]
- L2AviBridge.sol [https://github.com/AviatorAC/skybridge]

Status: Fixed

Classification

Impact: 2/5

Likelihood: 2/5

Exploitability: Independent

Complexity: Simple

Severity: Low

Recommendations

Remediation: It is recommended to consider the re-design of the system by introducing the refund functionality, so that the fees for using the `fastBridge` option are recovered if the liquidity pool does not have enough funds.

Resolution: Fixed in commit ID `a53e33d`: the fee payment of the fast withdrawal method was removed from L2's `_initiateWithdrawal()` and added into L1's `fastWithdrawal()`.

```
function _initiateWithdrawal(
    address _l2Token,
    address _from,
    address _to,
    uint256 _amount,
    uint32 _minGasLimit,
    bool _fastBridge,
    bytes memory _extraData
)
    internal
{
    require(_to != address(0), "L2AviBridge: cannot transfer to the zero address");

    address _trueTo = _to;

    if (_fastBridge) {
        require(allowedTokens[_l2Token], "L2AviBridge: token not allowed to be fast bridged"

        // Send the slow bridged tokens to our liquidity pool
        _trueTo = payable(liquidityPool);
    }

    if (_l2Token == Predeploys.LEGACY_ERC20_ETH) {
        _initiateBridgeETH(_from, _trueTo, _amount, _minGasLimit, _extraData);
        _emitFastWithdrawlInitiated(address(0), _l2Token, _from, _to, _amount, _fastBridge,
    } else {
        address l1Token = OptimismMintableERC20(_l2Token).remoteToken();
        _initiateBridgeERC20(_l2Token, l1Token, _from, _trueTo, _amount, _minGasLimit, _extr
        _emitFastWithdrawlInitiated(l1Token, _l2Token, _from, _to, _amount, _fastBridge, _ex
    }
}

function fastWithdraw(
```

```
    AviFastWithdrawal calldata _txn,
    bytes calldata _signature
)
    external
    payable
{
    require(backendUser != address(0), "L1AviBridge: invalid backend user");

    (bytes32 r, bytes32 s, uint8 v) = splitSignature(_signature);
    bytes32 structHash = keccak256(abi.encode(_WITHDRAWAL_TYPEHASH, _txn.l1_token, _txn.l2_t
    address signer = ECDSA.recover(_hashTypedDataV4(structHash), v, r, s);

    require(signer == backendUser, "invalid signature");
    require(fastBridgeNonces[_txn.to] == _txn.nonce, "invalid nonce");
    require(msg.value == flatFee, "AviBridge: insufficient value for fee");

    (bool feeSent, ) = address(flatFeeRecipient).call{value: flatFee}("");
    require(feeSent, "AviBridge: failed to send fee");

    fastBridgeNonces[_txn.to] += 1;

    // Transfer the tokens, ETH if the l1_token is address(0)
    if (_txn.l1_token == address(0)) {
        LIQUIDITY_POOL.sendETH(_txn.to, _txn.amount);
    } else {
        LIQUIDITY_POOL.sendERC20(_txn.to, _txn.l1_token, _txn.amount);
    }

    bool success = SafeCall.call(optimismPortal, gasleft(), 0, _txn.proof_transaction);
    require(success, "failed to call optimism portal");

    emit FastWithdraw(_txn.amount, _txn.l1_token, _txn.to, msg.sender);
}
```

F-2024-4262 - Non-Standard ERC20 Tokens Result in Loss of Funds - Low

Description:

The system assumes that transferring a certain number of tokens results in the exact same number of tokens being received. This assumption fails with fee-on-transfer tokens, which deduct a fee during each transfer, and rebasing tokens, which automatically adjust the token balance.

This will affect both on-chain and off-chain balance tracking of the system related to the contracts `AviBridge`, `L1AviBridge` and `L2AviBridge`.

There will be a mismatch between the `amount` indicated by the user (after fes) and the actual amount of tokens transferred, as explained in the following example:

1. A user would like to use a Supersonic withdraw of `100 tokens` (after fees).
2. If there is a 10% fee-on-transfer, the contract will receive `90 tokens` but the system will record `100 tokens`.
3. `100 tokens` will be sent to the user in the destination chain, from the liquidity pool.
4. The protocol lost `10 tokens`.

This mismatch also affects:

- The accounting of the system via mapping `deposits[_localToken][_remoteToken]`, as well as any other off-chain accounting.
- For rebasing tokens, the balance may change unexpectedly, leading to tokens being stuck in the contract.
- If the token's behavior does not match the contract's assumptions, transactions may revert.

The situation can be shown in the following function, where a transfer of ERC20 tokens takes place without taking into account the previously mentioned situations.

`AviBridge::_initiateBridgeERC20()`

```
function _initiateBridgeERC20(
    address _localToken,
    address _remoteToken,
    address _from,
    address _to,
    uint256 _amount,
    uint32 _minGasLimit,
    bytes memory _extraData
)
    internal
{
    if (_isOptimismMintableERC20(_localToken)) {
        require(
        _isCorrectTokenPair(_localToken, _remoteToken),
        "AviBridge: wrong remote token for Optimism Mintable ERC20 local token"
        );
        OptimismMintableERC20(_localToken).burn(_from, _amount);
    } else {
```

```
        IERC20(_localToken).safeTransferFrom(_from, address(this), _amount);
        deposits[_localToken][_remoteToken] = deposits[_localToken][_remoteToken] + _amount;
    }


    // Emit the correct events. By default this will be ERC20BridgeInitiated, but child
    // contracts may override this function in order to emit legacy events as well.
    _emitERC20BridgeInitiated(_localToken, _remoteToken, _from, _to, _amount, _extraData);


    MESSENGER.sendMessage(
        address(OTHER_BRIDGE),
        abi.encodeWithSelector(
        this.finalizeBridgeERC20.selector,
        // Because this call will be executed on the remote chain, we reverse the order of
        // the remote and local token addresses relative to their order in the
        // finalizeBridgeERC20 function.
        _remoteToken,
        _localToken,
        _from,
        _to,
        _amount,
        _extraData
        ),
        _minGasLimit
    );
}
```

Assets:

- L1AviBridge.sol [https://github.com/AviatorAC/skybridge]
- L2AviBridge.sol [https://github.com/AviatorAC/skybridge]
- AviBridge.sol [https://github.com/AviatorAC/skybridge]

Status: Fixed

Classification

Impact: 4/5

Likelihood: 1/5

Exploitability: Independent

Complexity: Medium

Severity: Low

Recommendations

Remediation: Consider calculating the balance of the recipient before and after the transfer of ERC20 tokens to calculate the actual number of tokens that were transferred, so it will be recorded to the mapping value.

Alternatively, implement a whitelist mechanism to only allow tokens that are compatible with the system.

Resolution: Fixed in commit ID `a53e33d`: a balance calculation before and after ERC20 transfers was introduced in order to record the actual amount of tokens transferred during deposits.

F-2024-4237 - onlyEOA Modifier Does Not Prevent Contracts From Calling the Bridge's Functionality - Info

Description:

The contracts L1AviBridge, L2AviBridge and AviERC721Bridge are using onlyEOA protection to make sure that the caller is an EOA and not a contract, hence the following list of functions cannot be called by the contracts:

- `L1AviBridge:: receive(), bridgeETH(), bridgeERC20()`
- `L2AviBridge:: withdraw(), receive()`
- `AviERC721Bridge:: bridgeERC721()`

However, due to the nature of the EVM, this protection can be bypassed if one of the functions above is called from the caller's `constructor()`. During the `constructor()` call, the external codesize of the contract equals zero, hence the defined condition can be skipped. It can be shown in the Proof Of Concept below.

Assets:

- universal/AviERC721Bridge.sol [https://github.com/AviatorAC/skybridge]
- L1AviBridge.sol [https://github.com/AviatorAC/skybridge]
- L2AviBridge.sol [https://github.com/AviatorAC/skybridge]

Status: Accepted

Classification

Impact: 1/5

Likelihood: 3/5

Exploitability: Independent

Complexity: Simple

Severity: Info

Recommendations

Remediation: Consider using the `msg.sender == tx.origin` condition to ensure that the caller is an EOA account.

Resolution: The risk is accepted by the client, since it is a feature of the original forked project

Evidences

Reproduce:

1. Alice creates a contract for the call.
2. Alice uses the contract to call the protected functionality of the bridge.

Results:

1. `Caller` is a contract which is using constructor() to bypass the onlyEOA condition. `Bridge` is a contract which is protected by the onlyEOA.

```
contract Bridge {
    modifier onlyEOA {
        require(!Address.isContract(msg.sender));
        _;
    }

    function tst() external onlyEOA {    2741 gas

    }
}

contract Caller {
    constructor(address _to) {    infinite gas 25200 gas
        console.log("CODE: ", address(this).code.length);
        (bool res,) = _to.call(abi.encodeWithSignature("tst()"));
        require(res);
    }
}
```

2. The result of the call can be seen in the screenshot below.

```
[vm] from: 0x5B3...eddC4 to: Caller.(constructor) value: 0 wei data: 0x608...e31bf logs: 0 hash: 0xd46...d229f
console.log:
CODE: 0
```

Observation Details

F-2024-4212 - Missing Events For Critical Data - Info

Description:

Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes with timelocks that allow users to evaluate them and consider if they would like to engage/exit based on how they perceive the changes as affecting the trustworthiness of the protocol or profitability of the implemented financial services. The alternative of directly querying the on-chain contract state for such changes is not considered practical for most users/usages.

The following functions do not emit any events:

`L1AviBridge::setBridgingFee()`, `setPaused()`, `setOtherBridge()`, `setAviTokenAddress()`, `withdraw()`

`AviBridge::setFlatFeeRecipient(), setFlatFee()`

`L1AviERC721Bridge::setPaused()`, `setOtherBridge()`

`AviERC721Bridge::setFlatFee()`

`L2AviBridge::addAllowedToken(),removeAllowedToken()`

`LiquidityPool:: sendETH()`

The absence of events in these functions means that there is no on-chain traceability or transparency when the data is updated.

Assets:

- universal/AviERC721Bridge.sol [https://github.com/AviatorAC/skybridge]
- L1AviBridge.sol [https://github.com/AviatorAC/skybridge]
- L1AviERC721Bridge.sol [https://github.com/AviatorAC/skybridge]
- LiquidityPool.sol [https://github.com/AviatorAC/skybridge]
- L2AviBridge.sol [https://github.com/AviatorAC/skybridge]
- AviBridge.sol [https://github.com/AviatorAC/skybridge]

Status: Fixed

Recommendations

Remediation:

To enhance transparency and traceability, it is recommended to emit events in key functions. This will allow users and external services to monitor and react to changes. Ensure that every critical action, especially those modifying contract states or handling funds, emits an event.

Resolution:

Fixed in commit ID `a53e33d`: events were implemented in the reported functions.

F-2024-4218 - Unfinished Code Reduces Transparency And Code Quality - Info

Description:

The `SkyBridgeERC721` contract contains three functions: `totalSupply()`, `tokenByIndex(),` and `tokenOfOwnerByIndex()`, which are currently unfinished. These functions are essential for providing complete ERC-721 functionality, particularly for enumerating tokens. The absence of implementations for these functions reduces the quality and transparency of the contract, potentially leading to issues in token management and user interactions. The code snippets with unimplemented code can be found below:

- `totalSupply()`:

```
function totalSupply() external view returns (uint256) {


}
```

- `tokenByIndex(uint256 index)`

```
function tokenByIndex(uint256 index) external view returns (uint256) {


}
```

- `tokenOfOwnerByIndex(address owner, uint256 index)`

```
function tokenOfOwnerByIndex(address owner, uint256 index) external view returns (uint256)


}
```

Furthermore, the contract `L2AviBridge` and ``L2AviERC721Bridge `have the function` paused()` which is unfinished as well. It can be seen below.

```
function paused() public view override returns (bool) { }
```

Assets:

- SkyBridgeERC721.sol [https://github.com/AviatorAC/skybridge]

Status: Fixed

Recommendations

Remediation: Consider implementing the aforementioned functions to increase both code quality and transparency.

Resolution: Fixed in commit ID `d974780`: the reported functions were removed from the code.

F-2024-4219 - Unused Imports Decrease Code Quality and Readability - Info

Description:

The following dependencies are imported in the contracts, but are not used anywhere in the system:

`L1AviBridge`: `ISemver`, `Constants`.

`L1AviERC721Bridge`: `ISemver`, `Predeploys`, `Constants`, `SuperchainConfig`.

`L2AviBridge`: `ISemver`, `Constants`.

`L2AviERC721Bridge`: `ISemver`.

`AviERC721Bridge`: `SuperchainConfig`.

Importing contracts will add a lot of noise to the contract code and may lead to confusion, decreasing code readability and, thus, its quality.

Additionally, `ISemver` is inherited by some of the reported contracts, increasing deployment code and deployment cost: `L1AviBridge`, `L1AviERC721Bridge`, `L2AviBridge`, `L2AviERC721Bridge`.

Assets:

- universal/AviERC721Bridge.sol [https://github.com/AviatorAC/skybridge]
- L1AviBridge.sol [https://github.com/AviatorAC/skybridge]
- L1AviERC721Bridge.sol [https://github.com/AviatorAC/skybridge]
- L2AviBridge.sol [https://github.com/AviatorAC/skybridge]
- L2AviERC721Bridge.sol [https://github.com/AviatorAC/skybridge]

Status: Fixed

Recommendations

Remediation: It is recommended to remove unused imports for better readability and code quality.

Resolution: Fixed in commit ID `a53e33d`: unused imports were deleted from the contracts.

F-2024-4220 - State Variable Visibility Best Practice Violation - Info

Description: The state variable `_isPaused` has no visibility defined. Functions and state variables' `visibility` should be set explicitly. Visibility levels should be specified consciously.

```
/// @notice if the bridge is paused
bool _isPaused = false;
```

Assets:

- L1AviBridge.sol [https://github.com/AviatorAC/skybridge]
- L1AviERC721Bridge.sol [https://github.com/AviatorAC/skybridge]

Status: Fixed

Recommendations

Remediation: Consider defining the visibility of the reported state variables.

Resolution: Fixed in commit ID `a53e33d`: the visibility of the reported state variable was defined.

F-2024-4221 - Redundant Calculation Results in Waste of Gas - Info

Description:

The function `_initiateETHDeposit()` calculates the `bridgeFee` twice instead of reusing it, accessing storage variables additional times:

```
function _initiateETHDeposit(address _from, address _to, uint32 _minGasLimit, bytes memory
    uint256 totalFee = msg.value * bridgingFee / 1000 + flatFee;
    require(msg.value >= totalFee, "AviBridge: insufficient ETH value");

    uint256 bridgeFee = msg.value * bridgingFee / 1000;
    ...
}
```

As a consequence, the gas cost of the function execution is increased.

Assets:

- L1AviBridge.sol [https://github.com/AviatorAC/skybridge]

Status: Fixed

Recommendations

Remediation:

It is recommended to first calculate `bridgeFee` and then reuse for the `totalFee` calculation.

Resolution:

Fixed in commit ID `a53e33d`: the calculation of `bridgeFee` is now performed earlier and reused for `totalFee`.

F-2024-4222 - Usage of receive() Function for Bridging may Revert - Info

Description:

As described in Solidity Documentation, the `receive()` function may only rely on `2300 Gas` when called by methods such as `transfer()` as well as any other limited Gas call such as `call()`.

Although the protocol introduces `onlyEOA` modifiers, it cannot be totally guaranteed that only EOA will call such methods.

L1AviBridge:

```
receive() external payable override onlyEOA {
    _initiateETHDeposit(msg.sender, msg.sender, RECEIVE_DEFAULT_GAS_LIMIT, bytes(""));
}
```

L2AviBridge:

```
receive() external payable override onlyEOA {
    _initiateWithdrawal(
        Predeploys.LEGACY_ERC20_ETH, msg.sender, msg.sender, msg.value, RECEIVE_DEFAULT_GAS_
    );
}
```

Assets:

- L1AviBridge.sol [https://github.com/AviatorAC/skybridge]
- L2AviBridge.sol [https://github.com/AviatorAC/skybridge]

Status: Accepted

Recommendations

Remediation: Consider using calls to `_initiateETHDeposit()` and `_initiateWithdrawal()` only with the corresponding bridging methods instead of relying on the `receive()` functions.

Resolution: The development team accepted the finding and the risks arising from it.

F-2024-4223 - Deprecated Optimism Predeploys Address Causing Unexpected Results - Info

Description:

`L2AviBridge`'s methods `receive()`, `finalizeDeposit()` and `_initiateWithdrawal()` use Optimism's `Predeploys` library `LEGACY_ERC20_ETH` address.

However, such address is deprecated and should not be used as seen in `Predeploys`.:

```
/// @notice Address of the LegacyERC20ETH predeploy. Deprecated. Balances are migrated to t
///         state trie as of the Bedrock upgrade. Contract has been locked and write functic
///         can no longer be accessed.
address internal constant LEGACY_ERC20_ETH = 0xDeadDeAddeAddEAddeadDEaDDEAdDeaDDeAD0000;
```

As a consequence, calls to `finalizeDeposit()` may not work as expected with `_l2Token` and `_l1Token` since it could not correctly be filtered out by its checks:

```
function finalizeDeposit(...,address _l2Token,...)
    external
    payable
    virtual
{
    if (_l1Token == address(0) && _l2Token == Predeploys.LEGACY_ERC20_ETH) {
        finalizeBridgeETH(_from, _to, _amount, _extraData);
    } else {
        finalizeBridgeERC20(_l2Token, _l1Token, _from, _to, _amount, _extraData);
    }
}
```

Similarly, in `_initiateWithdrawal()`, `_l2Token` and checks will not work as expected:

```
function _initiateWithdrawal(address _l2Token,...)
    internal
{
    if (_fastBridge) {
        require(allowedTokens[_l2Token], "L2AviBridge: token not allowed to be fast bridged"

            // For ETH, _amount is the amount of ETH to send to the user, and we need it + t
        if (_l2Token == Predeploys.LEGACY_ERC20_ETH) {
            uint256 requiredTotal = _amount + flatFee;
            require(msg.value >= requiredTotal, "L2AviBridge: insufficient ETH value");
        } else {
            // ERC20 tokens are just the amount of tokens to send to the user, so we require
            require(msg.value == flatFee, "L2AviBridge: insufficient ETH value");
        }
    }

    ...

    if (_l2Token == Predeploys.LEGACY_ERC20_ETH) {
        _initiateBridgeETH(_from, _trueTo, _amount, _minGasLimit, _extraData);
```

```
        _emitFastWithdrawlInitiated(address(0), _l2Token, _from, _to, _amount, _fastBridge,
    } else
        address l1Token = OptimismMintableERC20(_l2Token).remoteToken();z
        _initiateBridgeERC20(_l2Token, l1Token, _from, _trueTo, _amount, _minGasLimit, _extr
        _emitFastWithdrawlInitiated(l1Token, _l2Token, _from, _to, _amount, _fastBridge, _ex
    }
}
```

The `LEGACY_ERC20_ETH` value is also used in `_emitETHBridgeInitiated()` and `_emitETHBridgeFinalized()`. But in these cases it is only used for logging, which will not result in the severe consequences of the aforementioned functions.

More information on this deprecated address can be found in Optimism's Documentation, which states:

> The upgrade to Bedrock migrates all ether out of this contract and moves it to its native representation. All of the stateful methods in this contract will revert after the Bedrock upgrade. This contract is deprecated and its usage should be avoided.

The contract implementation of the address `Predeploys.LEGACY_ERC20_ETH` can be found here, in Optimism's Github. Some examples of functions being deprecated and non-functional are the following:

```
/// @custom:blocked
/// @notice Burns some amount of ETH.
function burn(address, uint256) public virtual override {
    revert("LegacyERC20ETH: burn is disabled");
}

/// @custom:blocked
/// @notice Transfers some amount of ETH.
function transfer(address, uint256) public virtual override returns (bool) {
    revert("LegacyERC20ETH: transfer is disabled");
}

/// @custom:blocked
/// @notice Approves a spender to spend some amount of ETH.
function approve(address, uint256) public virtual override returns (bool) {
    revert("LegacyERC20ETH: approve is disabled");
}

/// @custom:blocked
/// @notice Transfers funds from some sender account.
function transferFrom(address, address, uint256) public virtual override returns (bool) {
    revert("LegacyERC20ETH: transferFrom is disabled");
}
```

Assets:

- L2AviBridge.sol [https://github.com/AviatorAC/skybridge]

Status: Accepted

Classification

Impact:	3/5
Likelihood:	3/5

Recommendations

Remediation: Use the updated version of Optimism's `LEGACY_ERC20_ETH`.

Resolution: The development team accepted the finding and the risks arising from it, since the Optimism Standard Bridge forked by the project also uses it and current usage is only querying and comparing the `LEGACY_ERC20_ETH` contract address.

F-2024-4226 - Unused Functionality Increases Deployment Cost and Decreases Code Quality - Info

Description:

The modifier `onlyBackend()` is implemented in `AviERC721Bridge`. However, it is not used either in the contract scope or in child contracts:

```
modifier onlyBackend() {
    require(msg.sender == backendUser,
        "AviBridge: function can only be called by backend"
    );
    _;
}
```

Additionally, the contract includes the function `setBackend()`, which is also unused in the context of ERC721 bridging:

```
function setBackend(address _backend) external onlyRole(DEFAULT_ADMIN_ROLE) {
    require(_backend != address(0), "Invalid address.");
    require(_backend != backendUser, "This address is already the backend.");

    emit BackendChanged(backendUser, _backend);
    backendUser = _backend;
}
```

Unused code increases contract deployment gas cost and decreases readability, decreasing the overall code quality of the contracts.

Assets:

- universal/AviERC721Bridge.sol [https://github.com/AviatorAC/skybridge]

Status: Fixed

Recommendations

Remediation: It is recommended to remove unused code from the contracts.

Resolution: Fixed in commit ID `a53e33d`: the unused `backEnd` functionality was removed from the code.

F-2024-4229 - Old Solidity Version May Result in Unsafe Code - Info

Description:

The project uses the floating `pragma ^0.8.0`.

This may result in the contracts being deployed using the wrong `pragma` version, which is different from the one they were tested with. For example, they might be deployed using an outdated `pragma` version which may include bugs that affect the system negatively.

Additionally, using such an outdated version of Solidity can lead to several issues, including missing out on important bug fixes, security enhancements, and improved language features introduced in later versions. Staying up-to-date with recent Solidity versions is crucial for maintaining the security, efficiency, and overall quality of smart contracts.

Assets:

- libraries/AviPredeploys.sol [https://github.com/AviatorAC/skybridge]

Status: Accepted

Recommendations

Remediation: It is recommended to use one of the latest solidity versions that is also compatible with the rest of the contracts. Additionally, it is recommended to work with the most recent versions of the imported libraries and dependencies. Note that updating the solidity version requires a revision of the dependencies too.

Resolution: The development team accepted the finding and the risks arising from it.

F-2024-4238 - Code Style Inconsistency - Info

Description: `AviERC721Bridge::bridgeERC721()` has a specific require statement to make sure that the caller is an EOA and not a contract. The function can be shown in the code snippet below.

```
function bridgeERC721(
    address _localToken,
    address _remoteToken,
    uint256 _tokenId,
    bytes calldata _extraData
)
    external
    payable
{
    // Modifier requiring sender to be EOA. This prevents against a user error that would oc
    // if the sender is a smart contract wallet that has a different address on the remote c
    // (or doesn't have an address on the remote chain at all). The user would fail to recei
    // the NFT if they use this function because it sends the NFT to the same address as the
    // caller. This check could be bypassed by a malicious contract via initcode, but it tak
    // care of the user error we want to avoid.
    require(!Address.isContract(msg.sender), "ERC721Bridge: account is not externally owned"
    _initiateBridgeERC721(_localToken, _remoteToken, msg.sender, msg.sender, _tokenId, _extr
}
```

However, other contracts have a specific modifier called `onlyEOA` for the same purpose. The list of such contacts and their functions can be seen below:

- `L1AviBridge:: receive(), bridgeETH(), bridgeERC20()`
- `L2AviBridge:: withdraw(), receive()`

Assets:

- universal/AviERC721Bridge.sol [https://github.com/AviatorAC/skybridge]
- L1AviBridge.sol [https://github.com/AviatorAC/skybridge]
- L2AviBridge.sol [https://github.com/AviatorAC/skybridge]

Status: Accepted

Recommendations

Remediation: Consider using the `onlyEOA` modifier consistently across the codebase to improve code readability and quality.

Resolution: The development team accepted the finding and the risks arising from it.

F-2024-4251 - Method supportsInterface() Lacks the Override Keyword - Info

Description:

The contract SkyBridgeERC20 inherits OpenZeppelin's IERC165 interface verification and defines a new `supportsInterface()` function. However, it does not include the `override` keyword.

```
function supportsInterface(bytes4 _interfaceId) external pure returns (bool) {
    bytes4 iface1 = type(IERC165).interfaceId;
    // Interface corresponding to the legacy L2StandardERC20.
    bytes4 iface2 = type(ILegacyMintableERC20).interfaceId;
    // Interface corresponding to the updated OptimismMintableERC20 (this contract).
    bytes4 iface3 = type(IOptimismMintableERC20).interfaceId;
    return _interfaceId == iface1 || _interfaceId == iface2 || _interfaceId == iface3;
}
```

Assets:

- SkyBridgeERC20.sol [https://github.com/AviatorAC/skybridge]

Status: Fixed

Recommendations

Remediation: Introduce the `override` keyword in the method `supportsInterface()`.

Resolution: Fixed in commit ID `a53e33d`: the `override` keyword was added into `supportsInterface()`:

```
function supportsInterface(bytes4 _interfaceId) external pure override returns (bool) {
    bytes4 iface1 = type(IERC165).interfaceId;
    // Interface corresponding to the legacy L2StandardERC20.
    bytes4 iface2 = type(ILegacyMintableERC20).interfaceId;
    // Interface corresponding to the updated OptimismMintableERC20 (this contract).
    bytes4 iface3 = type(IOptimismMintableERC20).interfaceId;
    return _interfaceId == iface1 || _interfaceId == iface2 || _interfaceId == iface3;
}
```

F-2024-4252 - NatSpec does not Match Method Functionality - Info

Description:

The function `withdraw()` is used for bridging either native or ERC20 tokens. However, its NatSpec only mentions ETH.

Function NatSpec should match method functionality.

```
/// @notice Withdraw ETH from the bridge.
/// @param _txn The signed transation message
/// @param _signature The signature of the transaction
function withdraw(
    AviFastWithdrawal calldata _txn,
    bytes calldata _signature
) external {
    ...

    // Transfer the tokens, ETH if the l1_token is address(0)
    if (_txn.l1_token == address(0)) {
        LIQUIDITY_POOL.sendETH(_txn.to, _txn.amount);//@audit review liquidity pool interact
    } else {
        LIQUIDITY_POOL.sendERC20(_txn.to, _txn.l1_token, _txn.amount);
    }
}
```

Assets:

- L1AviBridge.sol [https://github.com/AviatorAC/skybridge]

Status: Fixed

Recommendations

Remediation: Update the `withdraw()` NatSpec so that it also mentions ERC20 bridging.

Resolution: Fixed in commit ID `a53e33d`: the NatSpec was updated to mention ERC20 bridging.

F-2024-4253 - Usage of Draft EIP712 Dependency - Info

Description: The contract `L1AviBridge` imports and inherits OpenZeppelin's `EIP712` contract. However, it is a draft version and thus it is not recommended to use it but to use a newer definitive version.

```
import { EIP712 } from "@openzeppelin/contracts/utils/cryptography/draft-EIP712.sol";
```

Assets:

- L1AviBridge.sol [https://github.com/AviatorAC/skybridge]

Status: Fixed

Recommendations

Remediation: Use an up-to-date, non-draft version of OpenZeppelin's `EIP712` contract.

Resolution: Fixed in commit ID `a53e33d`: the imported EIP712 contract was updated to a non-draft version.

F-2024-4255 - Incorrect Bridge Address Initialization - Info

Description: The contract `L1AviERC721Bridge` initializes the state variable `OTHER_BRIDGE` in the `constructor()` of `AviERC721Bridge` with `address(0)` instead of a proper address.

```
constructor(address payable _liquidityPool) AviERC721Bridge(address(0)) {...}

constructor(address _otherBridge) {
    OTHER_BRIDGE = _otherBridge;
    _setupRole(DEFAULT_ADMIN_ROLE, msg.sender); // make the deployer admin
    _numAdmins++;
}
```

Assets:

- L1AviERC721Bridge.sol [https://github.com/AviatorAC/skybridge]

Status: Accepted

Recommendations

Remediation: It is recommended to use a proper address in the `constructor()` of `AviERC721Bridge`.

Resolution: The development team accepted the finding and the risks arising from it. The following feedback was provided by the development team:

> Bridge address needs to be set after all contracts deployed. This is intentional, as we cannot assign an address to OTHER_BRIDGE in the constructor until both bridges are deployed.

F-2024-4257 - Missing flatFee Payment in Legacy Withdraw - Info

Description:

According to the provided documentation and conversations with the Aviator team:

> There should be a flat fee on **all deposits and all withdrawals.**

Although `flatFee` is checked to be added into the `msg.value` of the call, there is no on-chain mechanism that sends such `flatFee` when the legacy bridge is used (i.e. ! `_fastBridge`):

```
function _initiateWithdrawal(...)
    internal
{
    if (_fastBridge) {//@audit question liquidity fee only applies to L2 to L1 bridging?
        require(allowedTokens[_l2Token], "L2AviBridge: token not allowed to be fast bridged"

        // For ETH, _amount is the amount of ETH to send to the user, and we need it + the f
        if (_l2Token == Predeploys.LEGACY_ERC20_ETH/* @audit-ok deprecated according to libr
            uint256 requiredTotal = _amount + flatFee;
            require(msg.value >= requiredTotal, "L2AviBridge: insufficient ETH value");//@au
        } else {
            // ERC20 tokens are just the amount of tokens to send to the user, so we require
            require(msg.value == flatFee, "L2AviBridge: insufficient ETH value");
        }
    }

    address _trueTo = _to;

    if (_fastBridge) {
        ...
    }

    if (_l2Token == Predeploys.LEGACY_ERC20_ETH) {
        _initiateBridgeETH(_from, _trueTo, _amount, _minGasLimit, _extraData);
        _emitFastWithdrawlInitiated(address(0), _l2Token, _from, _to, _amount, _fastBridge,
    } else {
        address l1Token = OptimismMintableERC20(_l2Token).remoteToken();
        _initiateBridgeERC20(_l2Token, l1Token, _from, _trueTo, _amount, _minGasLimit, _extr
        _emitFastWithdrawlInitiated(l1Token, _l2Token, _from, _to, _amount, _fastBridge, _ex
    }
}
```

Assets:

- L2AviBridge.sol [https://github.com/AviatorAC/skybridge]

Status: Fixed

Recommendations

Remediation: Add a mechanism to harvest the `flatFees` in Legacy Withdraws.

Resolution: Fixed in commit ID `d974780`: a fee payment mechanism was added into the Legacy Withdraw flow.

```
function _initiateWithdrawal(
    address _l2Token,
    address _from,
    address _to,
    uint256 _amount,
    uint32 _minGasLimit,
    bool _fastBridge,
    bytes memory _extraData
)
    internal
{
    require(_to != address(0), "L2AviBridge: cannot transfer to the zero address");

    address _trueTo = _to;

    if (_fastBridge) {
        require(allowedTokens[_l2Token], "L2AviBridge: token not allowed to be fast bridged"

        // Send the slow bridged tokens to our liquidity pool
        _trueTo = payable(liquidityPool);
    } else {
        // For ETH, we need to include the flat fee in the required total
        if (_l2Token == Predeploys.LEGACY_ERC20_ETH) {
            uint256 requiredTotal = _amount + flatFee;
            require(msg.value == requiredTotal, "L2AviBridge: insufficient ETH value");
        } else {
            // For ERC20 tokens, require the flat fee to be sent along with the call
            require(msg.value == flatFee, "L2AviBridge: insufficient ETH value");
        }

        // If we are on a normal slow bridge, collect the fee on L2. For supersonic withdraw
        (bool feeSent, ) = flatFeeRecipient.call{value: flatFee}("");
        require(feeSent, "L2AviBridge: Failed to transfer flat fee");
    }

    if (_l2Token == Predeploys.LEGACY_ERC20_ETH) {
        _initiateBridgeETH(_from, _trueTo, _amount, _minGasLimit, _extraData);
        _emitFastWithdrawlInitiated(address(0), _l2Token, _from, _to, _amount, _fastBridge,
    } else {
        address l1Token = OptimismMintableERC20(_l2Token).remoteToken();
        _initiateBridgeERC20(_l2Token, l1Token, _from, _trueTo, _amount, _minGasLimit, _extr
        _emitFastWithdrawlInitiated(l1Token, _l2Token, _from, _to, _amount, _fastBridge, _ex
    }
}
```

F-2024-4271 - Bridge Contract Should have Specific Access Control - Info

Description:

The method `withdraw()` calls the `LiquidityPool's` functions `sendERC20()` and `sendETH()`:

```
function withdraw(
    AviFastWithdrawal calldata _txn,
    bytes calldata _signature
) external {
    ...
    // Transfer the tokens, ETH if the l1_token is address(0)
    if (_txn.l1_token == address(0)) {
        LIQUIDITY_POOL.sendETH(_txn.to, _txn.amount);
    } else {
        LIQUIDITY_POOL.sendERC20(_txn.to, _txn.l1_token, _txn.amount);
    }

    bool success = SafeCall.call(optimismPortal, gasleft(), 0, _txn.proof_transaction);
    require(success, "failed to call optimism portal");
}
```

However, the aforementioned methods are only callable by the `DEFAULT_ADMIN_ROLE`, which is a critical role by the system. Thus it is required for the system to grant said role to `L1AviBridge`:

```
function sendERC20(address _to, address _token, uint256 _amount) external onlyRole(DEFAULT_
    IERC20(_token).safeTransfer(_to, _amount);
}

function sendETH(address _to, uint256 _amount) external onlyRole(DEFAULT_ADMIN_ROLE) {
    require(_to != address(0), "LiquidityPool: _to address cannot be zero");
    (bool sent, ) = _to.call{value: _amount}("");
    require(sent, "failed to send ether");
}
```

Assets:

- L1AviBridge.sol [https://github.com/AviatorAC/skybridge]
- LiquidityPool.sol [https://github.com/AviatorAC/skybridge]

Status: Fixed

Recommendations

Remediation:

It is encouraged to keep a coarse-grained access control in the system by defining an specific role for the `L1AviBridge` to be checked in `sendERC20()` and `sendETH()`, instead of granting this contract the `DEFAULT_ADMIN_ROLE`.

Resolution: Fixed in commit ID `a53e33d`: a new role `BRIDGE_ROLE` was defined for the access control of `sendERC20()` and `sendETH()`.

F-2024-4273 - Mismatch Between Off-Chain Messages and Function Selectors - Info

Description:

When a deposit or withdrawal of tokens is performed, the methods `_initiateBridgeETH()` and `_initiateBridgeERC()` will be called. Such functions include off-chain messaging via Optimism's `Messenger` contract in order to perform the cross-chain messaging between the origin chain and the remote chain:

```
function _initiateBridgeERC20(...)
    internal
{
    ...
    MESSENGER.sendMessage(
        address(OTHER_BRIDGE),
        abi.encodeWithSelector(
            this.finalizeBridgeERC20.selector,
            // Because this call will be executed on the remote chain, we reverse the order
            // the remote and local token addresses relative to their order in the
            // finalizeBridgeERC20 function.
            _remoteToken,
            _localToken,
            _from,
            _to,
            _amount,
            _extraData
        ),
        _minGasLimit
    );
}

function _initiateBridgeETH(...)
    internal
{
    ...
    MESSENGER.sendMessage{ value: _amount }(
        address(OTHER_BRIDGE),
        abi.encodeWithSelector(this.finalizeBridgeETH.selector, _from, _to, _amount, _extraD
        _minGasLimit
    );
}
```

Both Messages will target the methods `finalizeBridgeERC20()` and `finalizeBridgeETH()` in the destination chain. Nevertheless, the functions that should be triggered in the destination chain are:

- In L1AviBridge → finalizeETHWithdrawal(), finalizeERCWithdrawal(), withdraw().
- In L2AviBridge → finalizeDeposit().

Note that the scope of this audit is exclusively on-chain, and off-chain mechanisms interacting with this finding cannot be evaluated. Be the case the

system implements a method to properly handle this situation, this issue could be marked as Mitigated.

Assets:

- AviBridge.sol [https://github.com/AviatorAC/skybridge]

Status: Accepted

Recommendations

Remediation: Implement a mechanism to properly target the correct functions via Optimism Messenger.

Resolution: The development team accepted the finding and the risks arising from it.

Disclaimers

Hacken Disclaimer

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

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

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

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

Technical Disclaimer

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

Appendix 1. Severity Definitions

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

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

[hknio/severity-formula]

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

Appendix 2. Scope

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

Scope Details	
Repository	https://github.com/AviatorAC/skybridge
Commit	d140bb2
Whitepaper	FlightPaper
Requirements	FlightPaper and Internal technical documentation
Technical Requirements	NatSpec comments and Internal technical documentation

Contracts in Scope
./factories/SkyBridgeERC20.sol
./factories/SkyBridgeERC20Factory.sol
./factories/SkyBridgeERC721.sol
./factories/SkyBridgeERC721Factory.sol
./factories/IOptimismMintableERC721.sol
./factories/IOptimismMintableERC20.sol
./L1/L1AviBridge.sol
./L1/L1AviERC721Bridge.sol
./L1/LiquidityPool.sol
./L2/L2AviBridge.sol
./L2/L2AviERC721Bridge.sol
./libraries/AviPredeploys.sol
./universal/AviBridge.sol
./universal/AviERC721Bridge.sol