Three Sigma Labs

Code Audit

layer3

Layer3 CUBE Escrow A multi-token reward distribution system

Disclaimer

Code Audit

Layer3 CUBE Escrow A multi-token reward distribution system

Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgment of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code's manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

Table of Contents

Code Audit

Layer3 CUBE Escrow A multi-token reward distribution system

Table of Contents

Summary

Code Audit

Layer3 CUBE Escrow A multi-token reward distribution system

Summary

Three Sigma Labs audited Layer3 in a 3 days engagement. The audit was conducted from 11-3-2024 to 13-3-2024.

Protocol Description

The Layer3 CUBE Escrow System extends the functionality of the CUBE smart contract by enabling minters to receive rewards in various token formats, including ERC20, ERC721, and ERC1155, along with native currency. This system introduces a practical method for token reward distribution, broadening the utility and engagement of the CUBE ecosystem.

Scope

Code Audit

Layer3 CUBE Escrow A multi-token reward distribution system

Scope

Filepath	nSLOC	ERCs
src/escrow/Factory.sol	112	-
src/escrow/Escrow.sol	136	ERC721Holder, ERC1155Holder
Sum	248	-

Assumptions

External dependencies are considered secure.

Methodology

Code Audit

Layer3 CUBE Escrow A multi-token reward distribution system

Methodology

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

Taxonomy

In this audit we report our findings using as a guideline Immunefi's vulnerability taxonomy, which can be found at immunefi.com/severity-updated/. The final classification takes into account the severity, according to the previous link, and likelihood of the exploit. The following table summarizes the general expected classification according to severity and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

Severity / Likelihood	LOW	MEDIUM	HIGH
NONE	None		
LOW	Low		
MEDIUM	Low	Medium	Medium
HIGH	Medium	High	High
CRITICAL	High	Critical	Critical

Project Dashboard

Code Audit

Layer3 CUBE Escrow A multi-token reward distribution system

Project Dashboard

Application Summary

Name	Layer3
Commit	13fd4d17
Language	Solidity
Platform	Ethereum, EVM L2s

Engagement Summary

Timeline	11-3-2024 to 13-3-2024
№ of Auditors	1
Review Time	3 days

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	0	0	0
High	0	0	0
Medium	2	2	0
Low	5	3	2

None	20	20	0

Category Breakdown

Suggestion	11
Documentation	0
Bug	7
Optimization	9
Good Code Practices	0

Code Maturity Evaluation

Code Audit

Layer3 CUBE Escrow A multi-token reward distribution system

Code Maturity Evaluation

Code Maturity Evaluation Guidelines

Category	Evaluation
Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system.
Arithmetic	The proper use of mathematical operations and semantics.
Centralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Code Stability	The extent to which the code was altered during the audit.
Upgradeability	The presence of parameterizations of the system that allow modifications after deployment.
Function Composition	The functions are generally small and have clear purposes.
Front-Running	The system's resistance to front-running attacks.
Monitoring	All operations that change the state of the system emit events, making it simple to monitor the state of the system. These events need to be correctly emitted.
Specification	The presence of comprehensive and readable codebase documentation.
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage.

Code Maturity Evaluation Results

Category	Evaluation
Access Controls	Satisfactory. The codebase has a strong access control mechanism.
Arithmetic	Satisfactory. The codebase uses Solidity version >0.8.0 implementing safe arithmetic.
Centralization	Weak. The admins/owners of the protocol are able to withdraw funds.
Code Stability	Satisfactory. The codebase was stable during the audit.
Upgradeability	Satisfactory. Most contracts are setup up to allow for upgradability.
Function Composition	Satisfactory. Functionalities are well split into different contracts and helpers.
Front-Running	Satisfactory. No front-running issues were identified.
Monitoring	Moderate. Some events should be added for state changing occurrences.
Specification	Satisfactory. The code matched the design specifications.
Testing and Verification	Moderate. Unit tests were present for most functionality but fuzzing and invariant tests could be performed.

Findings

3S-L3-M01

Some ERC20 tokens revert on zero transfers, making **`Factory.withdrawFunds`** always revert

Id	3S-L3-M01
Classification	Medium
Severity	Medium
Likelihood	Medium
Category	Bug
Status	Addressed in #9e7a790.

Description

The **`Factory.withdrawFunds`** function will pass a zero value as **`rakeBps`** into Escrow.withdrawERC20. As a consequence, **`Escrow._rakePayoutERC20`** will perform an ERC20 transfer of zero amount. Even though this is not the common pattern, some ERC20 tokens actually revert the execution in a zero amount transfer (e.g. **`LEND`**). This should not be a blocker to whitelist a token, but any Escrow contract with a token behaving like this will eventually cause funds to be stuck in the contract, given that the withdrawal of tokens will be reverted due to the zero **`rake`** transfer in **`withdrawERC20`**.

Recommendation

Skip the **`rake`** transfer if its value is zero:

```
uint256 rake = amount * rakeBps / 10_000;
if (rake > 0) {
    _rakePayoutERC20(token, rake);
}
```

3S-L3-M02

Lack of admin change syncing between **`Factory`** and **`Escrow`** can lead to privileged actions by old admin

Id	3S-L3-M02
Classification	Medium
Severity	Medium
Likelihood	Medium
Category	Bug
Status	Addressed in #2b21647.

Description

The Factory.createEscrow function deploys a new **`Escrow`** contract. The **`admin`** input will be set in the Factory's **`s_escrow_admin`** mapping, but it will also be registered as the **`DEFAULT_ADMIN_ROLE`** in the Escrow contract. The setting of that input address as **`admin`** in two different places is confusing and can lead to problematic situations. Let's consider two different scenarios:

- Scenario 1: the initial **`admin`** can call **`Factory.updateEscrowAdmin`** to update the **`s_escrow_admin[questId]`** value. The reason for this can be a migration to a more secure wallet, or maybe the original one is suspect of having become compromised. But because the function in the Factory contract doesn't change the **`admin`** in the Escrow contract (remove the role from the previous address and grant it to the new one), the old **`admin`** address will still have powers to change the token whitelist in the Escrow contract.

- Scenario 2: the initial **`admin`** can call **`Escrow.grantRole`** and **`Escrow.renounceRole`** to migrate the **`admin`** powers on the Escrow contract. But because the Factory has its own **`s_escrow_admin`** mapping tracker, it will not account for this change, and the old **`admin`** address will still have powers to withdraw the Escrow funds.

Recommendation

To avoid any confusing and problematic situations, consider having one single place where the Escrow **`admin`** is set. We suggest that the **`admin`** gets set as a specific address in the

Escrow contract, and that the Factory contract reads from it whenever it needs to know that present value.

3S-L3-L01

ERC1155 reward distributions don't charge fees

Id	3S-L3-L01
Classification	Low
Severity	Medium
Likelihood	Low
Category	Bug
Status	Acknowledged

Description

The Factory.distributeRewards function can be used to distribute ERC1155 rewards. It will call **`Escrow.withdrawERC1155`** and it will provide the right amount and token id.

The issue is that no **`rake`** fees will be enforced on ERC1155 transfers, even though ERC1155 can work like fungible tokens (e.g. tokens from the Enjin ecosystem). So distributing ERC1155 fungible tokens through the Escrow will never accrue fees to the treasury.

```
    function withdrawERC1155(address token, address to, uint256 amount,
uint256 tokenId)
        external
        onlyOwner
    {
        if (!s_whitelistedTokens[token]) {
            revert Escrow__TokenNotWhitelisted();
        }
        // @audit no fee being collected as a portion of `amount`
        IERC1155(token).safeTransferFrom(address(this), to, tokenId, amount,
"0x00");
    }
```

Since there doesn't seem to exist a lot of fungible ERC1155 tokens, it is unlikely that this ever becomes an issue.

Recommendation

If the **`amount`** input is greater than 1 (in other words, it's a fungible token), implement the same **`rake`** calculation and transferring logic as the one done on the **`withdrawERC20`** and **`withdrawNative`** functions.

3S-L3-L02

Privileged addresses can be set to address zero

Id	3S-L3-L02
Classification	Low
Severity	Low
Likelihood	Low
Category	Bug
Status	Addressed in #9e7a790.

Description

The new owner in Escrow.changeOwner and the new escrow admin in Factory.updateEscrowAdmin can be set to the zero address. Crucially, this shouldn't be allowed, since the change of each one of those values to the zero address leads to funds getting stuck in an Escrow contract.

Recommendation

Consider doing proper input validation in these critical functions.

3S-L3-L03

Recipient address of **`Escrow.withdrawNative`** can perform gas griefing to the transaction executor

Id	3S-L3-L03
Classification	Low
Severity	Low
Likelihood	Medium
Category	Bug
Status	Addressed in #2b21647.

Description

The Escrow.withdrawNative function is used by the owner (in a normal flow, the Factory contract) to send native funds to a given recipient address. When being called through **`Factory.distributeRewards`** the **`to`** address can be any address that is eligible for reward distribution.

Because a low-level call forwards 63/64 of the remaining call gas, the **`to`** address can grief the transaction executor by unnecessarily spending a large amount of gas in its fallback, during the **`withdrawNative`** call:

```
// @audit `to` address can spend a large amount of gas
(bool rewardSuccess,) = payable(to).call{value: amount - rake}("");
if (!rewardSuccess) {
    revert Escrow__NativePayoutError();
}
```

Since the **`distributeRewards`** function is only meant to be called by the **`CUBE`** contract through its **`mintCubes`** function, which inserts the call into a _for_ loop, there could also be a situation where one of the **`cubeData`** recipients spends all of the call gas, thus achieving a denial-of-service attack. However, since **`CUBE.sol`** is not in scope, it's unclear what the impact of such a potential gas griefing attack could be, or whether or not this potential DoS is a problem in the larger context of the protocol.

Recommendation

Consider limiting the gas being forwarded to the low-level call to the **`to`** address. Furthermore, consider getting a security review on **`CUBE.sol`** which accounts for the interaction with **`Factory`**.

3S-L3-L04

The **`Escrow.changeOwner`** function cannot be called by the **`Factory`** as the owner

Id	3S-L3-L04
Classification	Low
Severity	Low
Likelihood	High
Category	Bug
Status	Acknowledged

Description

In the normal flow of the Factory contract, deployed Escrow contracts will have the Factory as the owner. The Escrow contract has a number of **`onlyOwner`** functions which are called by the Factory. However, the Factory doesn't implement any logic to call **`Escrow.changeOwner`**. In other words, given that the Factory will be the owner, it is not possible to call this function nor to change the owner of an Escrow contract.

Recommendation

Consider adding a **`changeEscrowOwner`** function in the Factory contract to allow the calling of this function. If changing the owner is not desirable, simply remove the unused function.

3S-L3-L05

Events in `withdrawFunds` and `distributeRewards` have the wrong `TokenType` value

Id	3S-L3-L05
Classification	Low
Severity	Low
Likelihood	High
Category	Bug
Status	Addressed in #9e7a790.

Description

In the Factory.withdrawFunds function, the `tokenType` value determines the action to be taken on the Escrow contract. An `EscrowWithdrawal` event gets emitted for each valid `tokenType`.

The issue is that the `tokenType` value passed on the event does not correspond to the `tokenType` value passed in the function.

```
        if (tokenType == TokenType.NATIVE) {
            IEscrow(escrow).withdrawNative(to, escrow.balance, 0);
            // @audit TokenType.NATIVE == 3
            emit EscrowWithdrawal(msg.sender, to, address(0), 0,
escrow.balance, 0, questId);
        }
```

The same issue exists in the `distributionRewards` function when emitting the `TokenPayout` event.

Recommendation

Instead of hardcoding a number in the event emission, make sure to pass in the `tokenType` function input to avoid any mistakes.

3S-L3-N01

Revert string can be replaced by a custom error in **`_safeTransferERC20`**

Id	3S-L3-N01
Classification	None
Category	Optimization
Status	Addressed in #2b21647.

Description

If the transfer is unsuccessful in the Escrow._safeTransferERC20 function, the call will be reverted with an error string. This is the only place in the codebase in scope which doesn't use custom errors, which are cheaper.

Recommendation

Instead of using an error string, revert the transaction with a custom error.

3S-L3-N02

Missing function definitions in the **`IEscrow`** interface

Id	3S-L3-N02
Classification	None
Category	Suggestion
Status	Addressed in #9e7a790.

Description

The IEscrow interface defines the functions being implemented in the Escrow contract. However, some functions present in the implementation are lacking definition:

- **`escrowNativeBalance`**

- **`escrowERC721BalanceOf`**

- **`removeTokenFromWhitelist`**

Recommendation

Consider adding the definition of the missing functions in the **`IEscrow`** interface.

3S-L3-N03

No check to see if escrow exists in **`distributeRewards`**

Id	3S-L3-N03
Classification	None
Category	Suggestion
Status	Addressed in #9e7a790.

Description

The **`Factory.withdrawFunds`** function checks that the **`questId`** input maps to an existing escrow, otherwise it gracefully reverts:

```
address escrow = s_escrows[questId];
if (escrow == address(0)) {
    revert Factory__NoQuestEscrowFound();
}
```

However, distributeRewards doesn't perform that check, leading to a panic error.

Recommendation

Consider performing the same escrow existing check in **`distributeRewards`**.

3S-L3-N04

No need to pass **"0x00"** as data in **withdrawERC1155**

Id	3S-L3-N04
Classification	None
Category	Optimization
Status	Addressed in #9e7a790.

Description

The Escrow.withdrawERC1155 function calls **safeTransferFrom** on the ERC1155 token. The data parameter is being passed as **"0x00"**, which is a non empty string with no valuable information. This is unnecessary, and it's cheaper to just pass an empty string.

Recommendation

Pass an empty string as data to **IERC1155.safeTransferFrom**:

```
        IERC1155(token).safeTransferFrom(address(this), to, tokenId, amount,
"");
```

3S-L3-N05

`_safeTransferERC20` will succeed if `token` is not a contract

Id	3S-L3-N05
Classification	None
Category	Suggestion
Status	Addressed in #9e7a790.

Description

The Escrow._safeTransferERC20 internal function performs proper safety checks to make sure the token transfer was successful. However, it doesn't check that the input `token` address is an actual contract. In such situations, the low-level call will be successful and `data` will be empty, thus the function will not revert.

Given that this function is only called by `withdrawERC20` on tokens that have been whitelisted by the Escrow admin, there doesn't seem to be any meaningful impact in the protocol coming from this missing check. That being said, the `_safeTransferERC20` function might still be used in the future by other contracts with the false assumption that it is properly checking token transfers.

Recommendation

Consider checking the token is an actual contract by inspecting its code size. This is the same approach followed by OpenZeppelin's safeERC20 library.

3S-L3-N06

Nothing prevents the rake value from being greater than **`amount`** in **`withdrawERC20`** and **`withdrawNative`**

Id	3S-L3-N06
Classification	None
Category	Suggestion
Status	Addressed in #daa2990.

Description

In the Escrow.withdrawERC20 function, there is an assumption that **`rakeBps`** will be smaller than 10_000 basis points. But the code doesn't enforce this. If the **`rakeBps`** input is greater than 10_000, the transaction will revert with a panic underflow in the **`amount - rake`** subtraction.

The same issue exists in the Escrow.withdrawNative function.

Recommendation

Consider checking that **`rakeBps`** is not greater than 10_000, and revert the execution with a proper explanatory error.

3S-L3-N07

No check for zero address in **`tokenWhitelist`** changing functions

Id	3S-L3-N07
Classification	None
Category	Suggestion
Status	Addressed in #9e7a790.

Description

There's no proper input validation in Escrow.addTokenToWhitelist to avoid the addition of the zero address as a whitelisted token. Even though this function is access controlled, setting the zero value to true in a mapping can still potentially lead to odd behaviors in the contract.

Recommendation

Consider reverting the transaction if **`token == address(0)`**.

3S-L3-N08

Various state-changing functions don't emit events

Id	3S-L3-N08
Classification	None
Category	Suggestion
Status	Addressed in #9e7a790.

Description

There are many functions in the Factory and in the Escrow contracts which change state but don't emit events:

- **`Factory.updateEscrowAdmin`**

- **`Escrow.changeOwner`**

- **`Escrow.addTokenToWhitelist`**

- **`Escrow.removeTokenFromWhitelist`**

- **`Escrow.withdrawERC721`**

- **`Escrow.withdrawERC1155`**

Though **`withdrawERC721`** and **`withdrawERC1155`** end up emitting events inside the token contracts, it is still advisable to emit events in the context of the Escrow contract.

Recommendation

Emit events in all state-changing functions.

3S-L3-N09

Only one step to change ownership of an Escrow contract

Id	3S-L3-N09
Classification	None
Category	Suggestion
Status	Addressed in #2b21647.

Description

The Escrow owner is the only privileged address capable of withdrawing funds from the contract. To change ownership, the original owner needs to call **`changeOwner`**, which completes the ownership transfer in a single call. Given that passing a wrong address to this function would lead to funds being stuck in the contract, it is recommended to implement a 2-step ownership transferring mechanism, like OpenZeppelin's Ownable2Step.

Recommendation

Consider implement a 2-step ownership transferring mechanism to avoid critical errors in ownership changing.

3S-L3-N10

The **`ADMIN_ROLE`** constant in the Escrow contract is left unused

Id	3S-L3-N10
Classification	None
Category	Suggestion
Status	Addressed in #9e7a790.

Description

There is an ADMIN_ROLE constant in the Escrow contract which is left unused. Its name entails that a new type of administrative role, different from **`DEFAULT_ADMIN_ROLE`**, was conceived to be used somewhere in the code. If this role is not going to be used, the constant should be removed.

Recommendation

Remove the unused **`ADMIN_ROLE`** constant.

3S-L3-N11

Amounts are being indexed in the **`NativeTransfer`** event

Id	3S-L3-N11
Classification	None
Category	Suggestion
Status	Addressed in #9e7a790.

Description

The NativeTransfer event in the Escrow contract is indexing two **`uint256`** values: **`amount`** and **`rake`**. As a general rule of thumb, it is not advisable to index cryptocurrency amounts in events, since off-chain listeners will typically not be listening to specific amount values.

Recommendation

Consider removing the indexation of **`amount`** and **`rake`** in the **`NativeTransfer`** event:

```
    event NativeTransfer(
        address indexed to, uint256 amount, uint256 rake, address
rakePayoutAddress
    );
```

3S-L3-N12

Missing parameter indexation in the **`EscrowERC20Transfer`** event

Id	3S-L3-N12
Classification	None
Category	Suggestion
Status	Addressed in #9e7a790.

Description

The events in the Factory and in the Escrow contracts index token addresses and function callers or targets. But the **`EscrowERC20Transfer`** is an exception to this, presenting no indexed parameters. This makes it impossible to filter events based on some of the event arguments.

Recommendation

Consider indexing **`token`** and **`to`** in the **`EscrowERC20Transfer`** event:

```
    event EscrowERC20Transfer(
        address indexed token, address indexed to, uint256 amount, uint256
rake, address rakePayoutAddress
    );
```

3S-L3-N13

Missing **`override`** keyword for interface inherited methods

Id	3S-L3-N13
Classification	None
Category	Suggestion
Status	Addressed in #9e7a790.

Description

The contracts are not implementing their interface methods with **`override`** keywords. For example, the Factory contract implements all its interface functions, but it doesn't add the **`override`** keyword to any of the functions. This means the compiler won't check that the implemented functions are properly defined on the interface, making it prone to spelling errors and function signature mismatching.

Recommendation

Add the **`override`** keyword to those contracts functions as a best practice and for compiler checks.

3S-L3-N14

Function **withdrawNative** will call **i_treasury** even if **rakeBps** is zero

Id	3S-L3-N14
Classification	None
Category	Optimization
Status	Addressed in #9e7a790.

Description

When an **admin** calls Factory.withdrawFunds with **tokenType** set to **NATIVE**, the Factory will call **Escrow.withdrawNative** and pass **rakeBps** as zero. But even with **rakeBps** set to zero, the withdrawNative function will still do a call to the **i_treasury**, even though the value being passed is zero:

```
// rake payment
uint256 rake = amount * rakeBps / 10_000;
(bool rakeSuccess,) = payable(i_treasury).call{value: rake}("");
```

This can also happen if **Factory.distributeRewards** passes zero in the **rakeBps** input parameter.

Recommendation

Skip the low-level call if the calculated **rake** value is zero:

```
uint256 rake = amount * rakeBps / 10_000;
if (rake > 0) {
    (bool rakeSuccess,) = payable(i_treasury).call{value: rake}("");
    if (!rakeSuccess) {
        revert Escrow__NativeRakeError();
    }
}
```

3S-L3-N15

`escrow.balance` can be cached to save gas

Id	3S-L3-N15
Classification	None
Category	Optimization
Status	Addressed in #9e7a790.

Description

In the Factory.withdrawFunds function, the entire balance of the **`escrow`** address will be withdrawn if **`tokenType == TokenType.NATIVE`**. The _if_ statement uses **`escrow.balance`** twice - one for the **`IEscrow(escrow).withdrawNative`** call and one for the **`EscrowWithdrawal`** event emission. Under the hood, this uses the _BALANCE_ opcode. And even though the second usage will be cheaper due to the address being "warm", gas can still be saved if the balance value gets cached.

Recommendation

Consider only using **`escrow.balance`** once and caching its value:

```
        if (tokenType == TokenType.NATIVE) {
            uint256 escrowBalance = escrow.balance;
            IEscrow(escrow).withdrawNative(to, escrowBalance, 0);
            emit EscrowWithdrawal(msg.sender, to, address(0), 0,
escrowBalance, 0, questId);
        }
```

3S-L3-N16

All **`if`** statements will unnecessarily be checked in **`withdrawFunds`** and **`distributeRewards`**

Id	3S-L3-N16
Classification	None
Category	Optimization
Status	Addressed in #9e7a790.

Description

Both in Factory.withdrawFunds and in Factory.distributeRewards, there are four _if_ statements checking the **`tokenType`** input, and specific "withdrawing" logic runs accordingly. But even if **`tokenType == TokenType.NATIVE`** returns true, which is the first _if_ condition, all other _if_ statements will unnecessarily be evaluated.

Recommendation

Replace the usage of simple _if_ statements with a sequence of _if else_ blocks:

```
if (tokenType == TokenType.NATIVE) {
    // ...
}
else if (tokenType == TokenType.ERC20) {
    // ...
}
else if (tokenType == TokenType.ERC721) {
    // ...
}
else if (tokenType == TokenType.ERC1155) {
    // ...
}
```

3S-L3-N17

The `whitelistedTokens` array can be passed as `calldata` in `Factory.createEscrow`

Id	3S-L3-N17
Classification	None
Category	Optimization
Status	Addressed in #9e7a790.

Description

The function Factory.createEscrow is used to deploy a new `Escrow` contract and map it to a specific quest id. The `whitelistedTokens` array input is using the `memory` keyword, so the array will be copied from the calldata into memory. This is unnecessary.

Recommendation

Use the `calldata` keyword instead:

```
function createEscrow(
    uint256 questId,
    address admin,
    address[] calldata whitelistedTokens,
    address treasury
) external onlyRole(DEFAULT_ADMIN_ROLE) {
```

3S-L3-N18

s_cube variable in **Factory** can be immutable

Id	3S-L3-N18
Classification	None
Category	Optimization
Status	Addressed in #9e7a790.

Description

The **s_cube** variable on the Factory contract gets initialized through the **initialize** function. However, there's no logic in the **Factory** contract allowing the variable to be changed in the future. In other words, the contract is unnecessarily using a storage slot (and consequently perform storage reads).

Recommendation

Since this contract is to be used under a proxy, consider changing the **s_cube** variable to immutable and setting it in the constructor upon deployment.

3S-L3-N19

Unnecessary external call usage for `escrowERC20Reserves` and `escrowNativeBalance`

Id	3S-L3-N19
Classification	None
Category	Optimization
Status	Addressed in #9e7a790.

Description

In the Escrow.withdrawERC20, there's a check to understand if the contract has enough balance to perform the withdrawal operation. Because `Escrow.escrowERC20Reserves` is an `external` function, it can only be internally called using `this.escrowERC20Reserves`, which makes use of the _CALL_ opcode. However, this gas cost could be avoided by changing the function's visibility to `public`. This will allow the code to _jump_ to the function logic instead of performing an unnecessary external call.

The same issue exists in the function `withdrawNative`, which uses the `external` function `Escrow.escrowNativeBalance`.

Recommendation

Change the visibility of `escrowERC20Reserves` and `escrowNativeBalance` to `public`, and remove the `this.` usage in the functions `withdrawERC20` and `withdrawNative`.

3S-L3-N20

Array length is read on every loop run in the **`Escrow`** constructor

Id	3S-L3-N20
Classification	None
Category	Optimization
Status	Addressed in #9e7a790.

Description

In the Escrow constructor, there's a _for_ loop to register each **`tokenAddr[i]`** value as a whitelisted token. But the _for_ loop is reading the array length on each run, because the loop condition is checked every time.

Recommendation

Cache the array length:

```
uint256 length = tokenAddr.length;
for (uint256 i = 0; i < length;) {
    s_whitelistedTokens[tokenAddr[i]] = true;
    unchecked {
        ++i;
    }
}
```