SKALE contest
Findings & Analysis Report

2022-11-11

Table of contents

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit contest outlined in this document, C4 conducted an analysis of the SKALE smart contract system written in Solidity. The audit contest took place between February 18—March 3 2022.

Wardens

29 Wardens contributed reports to the SKALE contest:

1. cmichel
2. kirk-baird
3. WatchPug (jtp and ming)
4. leastwood
5. hubble (ksk2345 and shri4net)
6. 0x1f8b
7. IIIIIII
8. cccz
9. kyliek
10. gzeon
11. defsec
12. jayjonah8
13. csanuragjain
14. robee
15. ye0lde

16. TerrierLover
17. rfa
18. kenta
19. saian
20. 0xwags
21. CertoraInc (egjlmn1, OriDabush, ItayG, and shakedwinder)
22. m_smirnova2020
23. d4rk
24. Tomio

This contest was judged by Alex the Entreprenerd.

Final report assembled by itsmetechjay and liveactionllama.

Summary

The C4 analysis yielded an aggregated total of 13 unique vulnerabilities. Of these vulnerabilities, 2 received a risk rating in the category of HIGH severity and 11 received a risk rating in the category of MEDIUM severity. Of these, 2 HIGH severity and 1 MEDIUM severity item were mitigated by SKALE.

Additionally, C4 analysis included 14 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 15 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the C4 SKALE contest repository, and is composed of 33 smart contracts written in the Solidity programming language and includes 2,679 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website.

High Risk Findings (2)

[H-01] Reentrancy in `MessageProxyForSchain` leads to replay attacks

Submitted by cmichel

The `postIncomingMessages` function calls `_callReceiverContract(fromChainHash, messages[i], startingCounter + 1)` which gives control to a contract that is potentially attacker controlled *before* updating the `incomingMessageCounter`.

```
    for (uint256 i = 0; i < messages.length; i++) {
        // @audit re-entrant, can submit same postIncomingMessages a
        _callReceiverContract(fromChainHash, messages[i], startingCo
    }
    connectedChains[fromChainHash].incomingMessageCounter += message
```

The attacker can re-enter into the `postIncomingMessages` function and submit the same messages again, creating a replay attack. Note that the `startingCounter` is the way how messages are prevented from replay attacks here, there are no further nonces.

Proof of Concept

Attacker can submit two cross-chain messages to be executed:

1. Transfer 1000 USDC
2. A call to their attacker-controlled contract, could be masked as a token contract that allows re-entrance on `transfer`.

Some node submits the `postIncomingMessages(params)` transaction, transfers 1000 USDC, then calls the attackers contract, who can themself call `postIncomingMessages(params)` again, receive 1000 USDC a second time, and stop the recursion.

Recommended Mitigation Steps

Add a `messageInProgressLocker` modifier to `postIncomingMessages` as was done in `MessageProxyForMainnet`.

cstrangedk (SKALE) resolved:

Resolved via https://github.com/skalenetwork/IMA/pull/1054

[H-02] Gas pricing can be used to extort funds from users of SChain owner

Submitted by kirk-baird, also found by leastwood

https://github.com/skalenetwork/ima-c4-audit/blob/main/contracts/mainnet/CommunityPool.sol#L82-L112

https://github.com/skalenetwork/ima-c4-audit/blob/main/contracts/mainnet/MessageProxyForMainnet.sol#L235-L250

Impact

The function `refundGasByUser()` can be exploited by the message sender to drain nodes and SChain owners of their balances when processing incoming messages.

When a node collates a set of exits from an SChain to Ethereum, they are submitted on-chain via `MessageProxyForMainnet.sol`. For each message to a registered contract the user is required to pay for the refund via `CommunityPool.refundGasByUser()`.

The issue occurs in `CommunityPool.refundGasByUser()` as the amount to be refunded is calculated as `uint amount = tx.gasprice * gas;`, where `gas` is the gas used by the message. Since `tx.gasprice` is set by the node and there is no upper bounds on the price. Since EIP1559 the gas price is `BaseFee + Tip` and although `Base` is predetermined `Tip` is any arbitrary non-zero integer.

The attack is for a node to set an excessively high `tx.gasprice` which will be refunded out of the balance of the user who initiated the outgoing transaction or if that user has insufficient balance then from the SChain owner. Since the node submitting the transaction is refunded for their gas they do not lose from setting a higher gas price.

The impact of the attack is that the user requesting the exit and/or the SChain owner may have their ETH balances depleted to refund the submitter. The impact is worsened as if the user has insufficient balance a message will be sent to the SChain preventing them from making further exits until they have sufficient balance.

Note a similar issue may be seen in `IWallets.refundGasBySchain()` depending on how the gas calculations are performed (they are not in scope but the `TestWallet` also uses `tx.gasprice` in the same manner).

Proof of Concept

Processing incoming messages in `MessageProxyForMainnet.sol`

```
        for (uint256 i = 0; i < messages.length; i++) {
            gasTotal = gasleft();
            if (isContractRegistered(bytes32(0), messages[i].des
```

```
                address receiver = _getGasPayer(fromSchainHash,
                _callReceiverContract(fromSchainHash, messages[i
                notReimbursedGas += communityPool.refundGasByUse
                    fromSchainHash,
                    payable(msg.sender),
                    receiver,
                    gasTotal - gasleft() + additionalGasPerMessa
                );
            } else {
                _callReceiverContract(fromSchainHash, messages[i
                notReimbursedGas += gasTotal - gasleft() + addit
            }
        }
```

Refunding gas in `CommunityPool.sol`

```
    function refundGasByUser(
        bytes32 schainHash,
        address payable node,
        address user,
        uint gas
    )
        external
        override
        onlyMessageProxy
        returns (uint)
    {
        require(node != address(0), "Node address must be set");
        if (!activeUsers[user][schainHash]) {
            return gas;
        }
        uint amount = tx.gasprice * gas;
        if (amount > _userWallets[user][schainHash]) {
            amount = _userWallets[user][schainHash];
        }
        _userWallets[user][schainHash] = _userWallets[user][scha
        if (!_balanceIsSufficient(schainHash, user, 0)) {
            activeUsers[user][schainHash] = false;
            messageProxy.postOutgoingMessage(
                schainHash,
                schainLinks[schainHash],
                Messages.encodeLockUserMessage(user)
            );
        }
```

```
            node.sendValue(amount);
            return (tx.gasprice * gas - amount) / tx.gasprice;
        }
```

Recommended Mitigation Steps

One solution to avoid excessive over refunding of gas fees is to use a gas price oracle rather than `tx.gasprice`.

An alternate solution is to set a maximum gas price and have some incentives for the node submitting at a gas price below the maximum.

cstrangedk (SKALE) resolved:

> Resolved via https://github.com/skalenetwork/IMA/pull/1165/

Medium Risk Findings (11)

[M-01] Transactions can be replayed when a connectedChain is removed and then reconnected

Submitted by WatchPug, also found by cmichel

https://github.com/skalenetwork/ima-c4-audit/blob/11d6a6ae5bf16af552edd75183791375e501915f/contracts/MessageProxy.sol#L313-L317

```
    function removeConnectedChain(string memory schainName) public v
        bytes32 schainHash = keccak256(abi.encodePacked(schainName))
        require(connectedChains[schainHash].inited, "Chain is not ir
        delete connectedChains[schainHash];
    }
```

https://github.com/skalenetwork/ima-c4-audit/blob/11d6a6ae5bf16af552edd75183791375e501915f/contracts/MessageProxy.sol#L402-L409

```
function _addConnectedChain(bytes32 schainHash) internal onlyCha
    require(!connectedChains[schainHash].inited,"Chain is alreac
    connectedChains[schainHash] = ConnectedChainInfo({
        incomingMessageCounter: 0,
        outgoingMessageCounter: 0,
        inited: true
    });
}
```

In the current implementation, when a connected chain is removed, the `incomingMessageCounter` and `outgoingMessageCounter` will be deleted.

And if it's reconnected again, the `incomingMessageCounter` and `outgoingMessageCounter` will be reset to `0`.

However, since the contract is using `connectedChains[fromChainHash].incomingMessageCounter` and `signature` to ensure that the message can only be processed once.

Impact

1. Once the `incomingMessageCounter` resets to `0`, all the past messages (transactions) can be replayed with the old signatures.
2. Another impact is that, for the particular reconnected schain, both inbound and outbound messages may not be able to be processed properly, as the counter is now out of sync with the remote schain.

https://github.com/skalenetwork/ima-c4-audit/blob/11d6a6ae5bf16af552edd75183791375e501915f/contracts/schain/MessageProxyForSchain.sol#L212-L221

```
require(
    _verifyMessages(
        _hashedArray(messages, startingCounter, fromChainName),
        signature
    ),
    "Signature is not verified"
);
require(
```

```
        startingCounter == connectedChains[fromChainHash].incomingMe
        "Starting counter is not qual to incoming message counter");
```

Recommendation

When removing and connecting a schain, instead of delete/reset the couter, consider leaving the counter as it is when `removeConnectedChain`, `_addConnectedChain` also should not reset the counter:

```
    function removeConnectedChain(string memory schainName) public v
        bytes32 schainHash = keccak256(abi.encodePacked(schainName))
        require(connectedChains[schainHash].inited, "Chain is not in
        connectedChains[targetChainHash].inited = false;
    }
```

```
    function _addConnectedChain(bytes32 schainHash) internal onlyCha
        require(!connectedChains[schainHash].inited,"Chain is alreac
        connectedChains[schainHash].inited = true;
    }
```

DimaStebaev (SKALE) disagreed with severity and commented:

> Acknowledged, and work is on the roadmap.

GalloDaSballo (judge) decreased severity to Medium and commented:

> I believe the finding to have validity, in that, a set of signed messages can be replayed if the chain is disconnected and then re-connected while maintaining the same validators.
>
> At this time I think Medium Severity (External Conditions Reliance) to be more appropriate and that the issue can be fully sidestepped by changing validators on a chain reconnect

[M-02] When transferring tokens native on SKALE to

Ethereum with `TokenManagerERC20.exitToMainERC20()`, the tokens on the schain will be frozen on `TokenManagerERC20`, but they will not receive tokens on Ethereum

Submitted by WatchPug

In the current implementation of `TokenManagerERC20`, it allows `exitToMainERC20(tokenOnSchain, amount)`.

At L277 of `TokenManagerERC20.sol` in `exitToMainERC20()`, if `tokenOnSchain` is minted on SKALE schain natively, there are no such require statement that prevents the target chain being mainnet, eg: `require(chainHash != MAINNET_HASH, "...")`

Therefore, a user can set mainnet as the target chain, and at L298 of `TokenManagerERC20.sol`, the tokens will be transferred to the contract, and at L308, send message to Ethereum mainnet.

https://github.com/skalenetwork/ima-c4-audit/blob/11d6a6ae5bf16af552edd75183791375e501915f/contracts/schain/TokenManagers/TokenManagerERC20.sol#L95-L104

```
    function exitToMainERC20(
        address contractOnMainnet,
        uint256 amount
    )
        external
        override
    {
        communityLocker.checkAllowedToSendMessage(msg.sender);
        _exit(MAINNET_HASH, depositBox, contractOnMainnet, msg.s
    }
```

https://github.com/skalenetwork/ima-c4-audit/blob/11d6a6ae5bf16af552edd75183791375e501915f/contracts/schain/TokenManagers/TokenManagerERC20.sol#L264-L313

```
    function _exit(
```

```
        bytes32 chainHash,
        address messageReceiver,
        address contractOnMainChain,
        address to,
        uint256 amount
    )
        private
    {
        bool isMainChainToken;
        ERC20BurnableUpgradeable contractOnSchain = clonesErc20[
        if (address(contractOnSchain) == address(0)) {
            contractOnSchain = ERC20BurnableUpgradeable(contract
            isMainChainToken = true;
        }
        require(address(contractOnSchain).isContract(), "No toke
        require(contractOnSchain.balanceOf(msg.sender) >= amount
        require(
            contractOnSchain.allowance(
                msg.sender,
                address(this)
            ) >= amount,
            "Transfer is not approved by token holder"
        );
        bytes memory data = Messages.encodeTransferErc20Message(
        if (isMainChainToken) {
            data = _receiveERC20(
                chainHash,
                address(contractOnSchain),
                msg.sender,
                amount
            );
            _saveTransferredAmount(chainHash, address(contractOn
            require(
                contractOnSchain.transferFrom(msg.sender, addres
                "Transfer was failed"
            );
        } else {
            require(
                contractOnSchain.transferFrom(msg.sender, addres
                "Transfer was failed"
            );
            contractOnSchain.burn(amount);
        }
        messageProxy.postOutgoingMessage(
            chainHash,
            messageReceiver,
```

```
            data
        );
    }
```

However, the `DepositBoxERC20` contract on Ethereum mainnet does not support such message from `TokenManagerERC20` on the schain:

The type of the message from schain `TokenManagerERC20` is `TRANSFER_ERC20_AND_TOKEN_INFO` (see L354 of TokenManagerERC20.sol) or `TRANSFER_ERC20_AND_TOTAL_SUPPLY` (see L362 of TokenManagerERC20.sol).

https://github.com/skalenetwork/ima-c4-audit/blob/11d6a6ae5bf16af552edd75183791375e501915f/contracts/schain/TokenManagers/TokenManagerERC20.sol#L339-L370

```
    function _receiveERC20(
        bytes32 chainHash,
        address erc20OnMainChain,
        address to,
        uint256 amount
    )
        private
        returns (bytes memory data)
    {
        ERC20BurnableUpgradeable erc20 = ERC20BurnableUpgradeabl
        uint256 totalSupply = erc20.totalSupply();
        require(amount <= totalSupply, "Amount is incorrect");
        bool isERC20AddedToSchain = _schainToERC20[chainHash].cc
        if (!isERC20AddedToSchain) {
            _addERC20ForSchain(chainHash, erc20OnMainChain);
            data = Messages.encodeTransferErc20AndTokenInfoMessa
                erc20OnMainChain,
                to,
                amount,
                _getErc20TotalSupply(erc20),
                _getErc20TokenInfo(erc20)
            );
        } else {
            data = Messages.encodeTransferErc20AndTotalSupplyMes
                erc20OnMainChain,
                to,
```

```
                amount,
                _getErc20TotalSupply(erc20)
            );
        }
        emit ERC20TokenReady(chainHash, erc20OnMainChain, amount
    }
```

`DepositBoxERC20` on Ethereum MAINNET can only process `TRANSFER_ERC20`. (see DepositBoxERC20.sol L155 and Messages.sol L270)

When getting a message with the type of `TRANSFER_ERC20_AND_TOKEN_INFO` or `TRANSFER_ERC20_AND_TOTAL_SUPPLY` from schain `TokenManagerERC20`, it will revert at L270 of Messages.sol.

As a result, the schain tokens will be frozen on TokenManagerERC20, but they will not receive tokens on Ethereum.

https://github.com/skalenetwork/ima-c4-audit/blob/11d6a6ae5bf16af552edd75183791375e501915f/contracts/mainnet/DepositBoxes/DepositBoxERC20.sol#L143-L164

```
    function postMessage(
        bytes32 schainHash,
        address sender,
        bytes calldata data
    )
        external
        override
        onlyMessageProxy
        whenNotKilled(schainHash)
        checkReceiverChain(schainHash, sender)
        returns (address)
    {
        Messages.TransferErc20Message memory message = Messages.
        require(message.token.isContract(), "Given address is nc
        require(ERC20Upgradeable(message.token).balanceOf(addres
        _removeTransferredAmount(schainHash, message.token, mess
        require(
            ERC20Upgradeable(message.token).transfer(message.rec
            "Transfer was failed"
        );
```

```
        return message.receiver;
    }
```

https://github.com/skalenetwork/ima-c4-audit/blob/11d6a6ae5bf16af552edd75183791375e501915f/contracts/Messages.sol#L267-L272

```
    function decodeTransferErc20Message(
        bytes calldata data
    ) internal pure returns (TransferErc20Message memory) {
        require(getMessageType(data) == MessageType.TRANSFER_ERC
        return abi.decode(data, (TransferErc20Message));
    }
```

Recommendation

Consider preventing moving schain native tokens to Ethereum MAINNET, for example: Add `require(chainHash != MAINNET_HASH, "...")` near L277 of TokenManagerERC20.sol.

cstrangedk (SKALE) disagreed with severity and commented:

> Valid issue, however disagree with severity as issue relates to `function incorrect as to spec`. Suggest `low severity`.

GalloDaSballo (judge) decreased severity to Medium and commented:

> Let me reason through the finding: -> The warden has shown a way for funds to be lost as long as a user uses a sChainNativeToken and burns them to bridge to mainnet (Potentially High)

> -> This is contingent on the configuration of the depositBoxErc20 (Potentially Med)

> I disagree with the sponsor argument in that while the code may not be as to spec, the functionality impaired as a medium to high impact.

> At this time I can rationalize the severity either as: -> It should be High because the given codebase “default” functionality has this flaw -> it should be Med because this is contingent on a configuration parameter

> Given the pre-context that the sChain could be set up by the admin to allow the misconfiguration to happen, at this time, I believe Medium Severity to be more appropriate as the impact is solely dependent upon the configuration of the chain which as explained in the readmes is mostly dependent on the admin.

cstrangedk (SKALE) commented:

> Mitigated in skalenetwork/IMA#1031.

[M-03] S2S Transfer from the origin schain to another schain with automatic deploy disabled can cause funds to be frozen

Submitted by WatchPug

When moving tokens that are native on the origin schain, to another schain, `TokenManagerERC20.sol#transferToSchainERC20()` will be called, which calls `_exit()` -> `_receiveERC20()`:

https://github.com/skalenetwork/ima-c4-audit/blob/11d6a6ae5bf16af552edd75183791375e501915f/contracts/schain/TokenManagers/TokenManagerERC20.sol#L289-L301

```
    if (isMainChainToken) {
        data = _receiveERC20(
            chainHash,
            address(contractOnSchain),
            msg.sender,
            amount
        );
        _saveTransferredAmount(chainHash, address(contractOnSchain),
        require(
            contractOnSchain.transferFrom(msg.sender, address(this),
            "Transfer was failed"
        );
    }
```

https://github.com/skalenetwork/ima-c4-audit/blob/11d6a6ae5bf16af552edd75183791375e501915f/contracts/schain/TokenManagers/TokenManagerERC20.sol#L351-L361

```
    bool isERC20AddedToSchain = _schainToERC20[chainHash].contains(e
    if (!isERC20AddedToSchain) {
        _addERC20ForSchain(chainHash, erc20OnMainChain);
        data = Messages.encodeTransferErc20AndTokenInfoMessage(
            erc20OnMainChain,
            to,
            amount,
            _getErc20TotalSupply(erc20),
            _getErc20TokenInfo(erc20)
        );
    }
```

However, on the target schain, while handling the inbound message with `postMessage()` -> `_sendERC20()`, when `contractOnSchain` is `false`, The transaction will fail with `"Automatic deploy is disabled"` when `automaticDeploy == false`:

https://github.com/skalenetwork/ima-c4-audit/blob/11d6a6ae5bf16af552edd75183791375e501915f/contracts/schain/TokenManagers/TokenManagerERC20.sol#L227-L235

```
     contractOnSchain = clonesErc20[fromChainHash][token];

    if (address(contractOnSchain) == address(0)) {
        require(automaticDeploy, "Automatic deploy is disabled");
        contractOnSchain = new ERC20OnChain(message.tokenInfo.name,
        clonesErc20[fromChainHash][token] = contractOnSchain;
        addedClones[contractOnSchain] = true;
        emit ERC20TokenCreated(fromChainHash, token, address(contrac
    }
```

As a result, any tokens that are locked in the origin schain by the user will be frozen in the contract.

Recommendation

Consider adding a `mapping` storage to cache whether `automaticDeploy` is enabled on a certain schain, the cache should be updated once the `automaticDeploy` is updated.

And only allows S2S transfer when `automaticDeploy` is enabled on the target schain.

To further avoid the edge case of: right after the user submitted the S2S transfer tx on the from schain, the target schain disabled `automaticDeploy` and the user's tokens can be frozen in the from schain. We can introduce a 24 hrs timelock for disabling `automaticDeploy`.

cstrangedk (SKALE) disputed and commented:

> Issue raised is acknowledged and work is assigned on the roadmap. SKALE Chain owners must ensure any mapped assets, either through manual or automatic mapping are compatible with their dApp(s). Manual mapping mode is the default mode for bridge operation.

GalloDaSballo (judge) decreased severity to Medium and commented:

> I agree with both sides of the argument, and because this is contingent on configuration and admin privilege, believe Medium Severity to be more appropriate

[M-04] TokenManagerERC20.sol uses transferFrom() instead of safeTransferFrom()

Submitted by jayjonah8, also found by cmichel and IIIIIII

https://github.com/skalenetwork/ima-c4-audit/blob/main/contracts/schain/TokenManagers/TokenManagerERC20.sol#L298

https://github.com/skalenetwork/ima-c4-audit/blob/main/contracts/schain/TokenManagers/TokenManagerERC20.sol#L30

3

Impact

In TokenManagerERC20.sol the _exit() function makes use of transferFrom() instead of using safeTransferFrom(). Tokens that don't correctly implement the latest EIP20 spec will be unusable in the protocol as they revert the transaction because of the missing return value.

Proof of Concept

https://github.com/skalenetwork/ima-c4-audit/blob/main/contracts/schain/TokenManagers/TokenManagerERC20.sol#L298

https://github.com/skalenetwork/ima-c4-audit/blob/main/contracts/schain/TokenManagers/TokenManagerERC20.sol#L303

Recommended Mitigation Steps

It's recommended to use OpenZeppelin's SafeERC20 versions with the safeTransfer and safeTransferFrom functions that handle the return value check as well as non-standard-compliant tokens.

cstrangedk (SKALE) disputed, disagreed with severity and commented:

> Issue is acknowledged and work is pending on the roadmap. No loss of funds is possible, only revert txn. In the meantime, SKALE Chain owners at their discretion can expand the bridge compatibility to use safeTransfer functions. Owners must evaluate token compatibility.

GalloDaSballo (judge) decreased severity to Medium and commented:

> I believe that given the "need for configuration" the finding cannot be of High Severity. Additionally, the tx will revert as such no loss of funds is possible.

> The tokens that will cause a revert (e.g. USDT) will simply be unusable.

> While the argument for configuration is correct in de-escalating to Medium, I don't believe it exempts the code from being properly scrutinized.

> If a user were to configure their chain to use TokenManagerERC20 they'd have revert on non returning tokens, for that reason I believe Medium Severity to be appropriate as this is contingent on configuration

[M-05] Schain owners can rug pull users' funds

Submitted by lllllll, also found by gzeon and kirk-baird

https://github.com/skalenetwork/ima-c4-audit/blob/11d6a6ae5bf16af552edd75183791375e501915f/contracts/mainnet/DepositBoxes/DepositBoxEth.sol#L138-L142

https://github.com/skalenetwork/ima-c4-audit/blob/11d6a6ae5bf16af552edd75183791375e501915f/contracts/mainnet/DepositBoxes/DepositBoxERC20.sol#L196-L200

https://github.com/skalenetwork/ima-c4-audit/blob/11d6a6ae5bf16af552edd75183791375e501915f/contracts/mainnet/DepositBoxes/DepositBoxERC721.sol#L183-L187

https://github.com/skalenetwork/ima-c4-audit/blob/11d6a6ae5bf16af552edd75183791375e501915f/contracts/mainnet/DepositBoxes/DepositBoxERC1155.sol#L261-L271

Impact

Once a chain has been killed the chain owner is able to call `getFunds()` on each of the deposit boxes and transfer funds/tokens wherever he/she wishes

Even if the owner is benevolent the fact that there is a rug vector available may negatively impact the protocol's reputation. See this example where a similar finding has been flagged as a high-severity issue. I've downgraded these instances to be a medium since it requires cooperation of the IMA mainnet admin.

Proof of Concept

```
    function getFunds(string calldata schainName, address payabl
        external
        override
        onlySchainOwner(schainName)
        whenKilled(keccak256(abi.encodePacked(schainName)))
```

https://github.com/skalenetwork/ima-c4-audit/blob/11d6a6ae5bf16af552edd75183791375e501915f/contracts/mainnet/DepositBoxes/DepositBoxEth.sol#L138-L142

```
    function getFunds(string calldata schainName, address erc20(
        external
        override
        onlySchainOwner(schainName)
        whenKilled(keccak256(abi.encodePacked(schainName)))
```

https://github.com/skalenetwork/ima-c4-audit/blob/11d6a6ae5bf16af552edd75183791375e501915f/contracts/mainnet/DepositBoxes/DepositBoxERC20.sol#L196-L200

```
    function getFunds(string calldata schainName, address erc721
        external
        override
        onlySchainOwner(schainName)
        whenKilled(keccak256(abi.encodePacked(schainName)))
```

https://github.com/skalenetwork/ima-c4-audit/blob/11d6a6ae5bf16af552edd75183791375e501915f/contracts/mainnet/DepositBoxes/DepositBoxERC721.sol#L183-L187

```
    function getFunds(
        string calldata schainName,
        address erc1155OnMainnet,
        address receiver,
        uint256[] memory ids,
        uint256[] memory amounts
    )
        external
```

```
override
onlySchainOwner(schainName)
whenKilled(keccak256(abi.encodePacked(schainName)))
```

https://github.com/skalenetwork/ima-c4-audit/blob/11d6a6ae5bf16af552edd75183791375e501915f/contracts/mainnet/DepositBoxes/DepositBoxERC1155.sol#L261-L271

Recommended Mitigation Steps

Add a long time lock for killing so users have plenty of time to get their funds out before a kill

DimaStebaev (SKALE) disputed and commented:

> Acknowledged, and timelock & eventual removal has been added to the roadmap This is not a vulnerability. We explicitly added this functionality to prevent loosing of funds in case of technical problems with SKALE chain. It requires SKALE chain owner and SKALE foundation cooperation to run this mechanism. It is going to be removed eventually after some time of stable work.

GalloDaSballo (judge) commented:

> While I empathize with the sponsor's side that the function is meant as a security mechanism, it does allow the chainOwner to pull all funds and transfer them to their own wallet.

> Because this is contingent on Admin Privilege, I believe Medium Severity to be appropriate

[M-06] Centralisation risk: admin role of `TokenManagerEth` can rug pull all Eth from the bridge

Submitted by kirk-baird

There is a Centralisation risk of the bridge where the `DEFAULT_ADMIN_ROLE` of `TokenManagerEth.sol` is able to modify the ERC20 token on the SChain to any

arbitrary address. This would allow the admin role to change the address to one where they have infinite supply, they could then call `exitToMain(amount)` equal to the balance of the DepositBox in the main Ethereum chain. After the message is process on the main Ethereum chain they will receive the entire Eth balance of that contract.

The rug pull attack occurs because there is a `DEFAULT_ADMIN_ROLE` which is set in the intiialisation to the `msg.sender` as seen in `initializeTokenManager()` below.

The `DEFAULT_ADMIN_ROLE` may then call `setEthErc20Address(IEthErc20 newEthErc20Address)` setting `newEthErc20Address` to any arbitrary contract they control.

Proof of Concept

```
function setEthErc20Address(IEthErc20 newEthErc20Address) ex
    require(hasRole(DEFAULT_ADMIN_ROLE, msg.sender), "Not au
    require(ethErc20 != newEthErc20Address, "Must be new add
    ethErc20 = newEthErc20Address;
}
```

```
function initializeTokenManager(
    string memory newSchainName,
    IMessageProxyForSchain newMessageProxy,
    ITokenManagerLinker newIMALinker,
    ICommunityLocker newCommunityLocker,
    address newDepositBox
)
    public
    virtual
    initializer
{
    require(newDepositBox != address(0), "DepositBox address

    AccessControlEnumerableUpgradeable.__AccessControlEnumer
    _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
    _setupRole(AUTOMATIC_DEPLOY_ROLE, msg.sender);
    _setupRole(TOKEN_REGISTRAR_ROLE, msg.sender);

    schainHash = keccak256(abi.encodePacked(newSchainName));
```

```
        messageProxy = newMessageProxy;
        tokenManagerLinker = newIMALinker;
        communityLocker = newCommunityLocker;
        depositBox = newDepositBox;

        emit DepositBoxWasChanged(address(0), newDepositBox);
    }
```

Recommended Mitigation Steps

Consider removing the function `setEthErc20Address()` as `ethErc20` is set in the `initialize()` function and does not need to be changed.

DimaStebaev (SKALE) disagreed with severity and commented:

> Acknowledged, and this can be done only by SKALE chain owner.

GalloDaSballo (judge) commented:

> Agree that the admin has the ability to rug users and agree with Med Severity

[M-07] `transferredAmount` on mainnet can be drained if a malicious account can mint more tokens on Schain

Submitted by kyliek, also found by cccz

https://github.com/skalenetwork/ima-c4-audit/blob/11d6a6ae5bf16af552edd75183791375e501915f/contracts/mainnet/DepositBoxes/DepositBoxERC20.sol#L45

https://github.com/skalenetwork/ima-c4-audit/blob/11d6a6ae5bf16af552edd75183791375e501915f/contracts/schain/tokens/ERC20OnChain.sol#L49-L50

https://github.com/skalenetwork/ima-c4-audit/blob/11d6a6ae5bf16af552edd75183791375e501915f/contracts/mainnet/DepositBoxes/DepositBoxERC20.sol#L95

https://github.com/skalenetwork/ima-c4-audit/blob/11d6a6ae5bf16af552edd75183791375e501915f/contracts/schain/tokens/ERC20OnChain.sol#L60-L63

Impact

Anyone on Schain that is able to mint more tokens, other than the mint action from `postMessage` in `tokenManagerERC20` by bridging tokens over, can potentially drain the locked tokens in `transferredAmount` in `depositBoxERC20` on mainnet by calling exit with the same amount of tokens in `transferredAmount[schainHash][token]`.

This will trap other users' funds on sChain and lost those funds on mainnet to the malicious attacker.

An example of proof of concept using `ERC20OnChian` is given below. This case may seem to be special as the deployer of the clone contract is malicious. However, this is a potential risk that generalises to other custom contracts with any `mint` functionality.

Proof of Concept (with ERC20)

- Malicious account deploys an ERC20 clone `ERC20OnChain` on Schain.
- By deployment, the malicious account is assigned the MINTER_ROLE on `ERC20OnChain` in the constructor.

```
    constructor(
        string memory contractName,
        string memory contractSymbol
    ) initializer
    {
        AccessControlEnumerableUpgradeable.__AccessControlEnumer
        ERC20Upgradeable.__ERC20_init(contractName, contractSymb
        ERC20BurnableUpgradeable.__ERC20Burnable_init();
        _setRoleAdmin(MINTER_ROLE, MINTER_ROLE);
        _setupRole(MINTER_ROLE, _msgSender());
    }
```

- This ERC20 clone get registered on `tokenMangagerERC20.sol` by function `addERC20TokenByOwner`.

```
    function addERC20TokenByOwner(
        string calldata targetChainName,
        address erc20OnMainChain,
        address erc20OnSchain
     )
```

- The malicious account wait for more users to deposit tokens on `depositBoxERC20.depositERC20`, which will increase the amount in `transferredAmount[schainHash][token]`

```
    function depositERC20(
        string calldata schainName,
        address erc20OnMainnet,
        uint256 amount
    )
```

- Malicious account mint to his account the amount equal to `transferredAmount[schainHash][token]` on `ERC20OnChain`

```
    function mint(address account, uint256 value) external overr
        require(hasRole(MINTER_ROLE, _msgSender()), "Sender is r
        _mint(account, value);
    }
```

- Malicious account calls `exitToMainERC20` in `TokenManagerERC20`.

```
    function exitToMainERC20(
        address contractOnMainnet,
        uint256 amount
    )
        external
        override
    {
        communityLocker.checkAllowedToSendMessage(msg.sender);
```

```
            _exit(MAINNET_HASH, depositBox, contractOnMainnet, msg.s
        }
```

- `transferredAmount[schainHash][token]` is drained to the malicious account on mainnet and victims' tokens get stranded on schain.

Recommended Mitigation Steps

Disable minting function to be called directly in `ERC20OnChain`. Only allow minting when bridging tokens over.

cstrangedk (SKALE) disputed and commented:

> Issue is acknowledged and work is pending on the roadmap. SKALE Chain owners must carefully manage MINTER_ROLE, and end-users carefully monitor role.

GalloDaSballo (judge) decreased severity to Medium and commented:

> I believe the sponsor reply sheds further light into how the system will be used in 99% of cases.
>
> That said we have to judge the codebase for how it could be exploited and used by malicious actors, and I do agree with he warden that if the MINTER_ROLE is granted to someone (let's say the admin) then they could use it to drain funds from the mainnet side of the bridge.
>
> Because this is contingent on setup and Admin Privilege, I believe Medium Severity to be more appropriate.
>
> Switching from a role based system to an immutable Minter Address (the bridge contract) can be used as mitigation, alternatively sChain end users will have to monitor for these types of changes and act accordingly

[M-08] `BURNER_ROLE` can burn any amount of EthErc20 from an arbitrary address

Submitted by cccz, also found by 0x1f8b

Impact

Same as https://github.com/code-423n4/2022-01-livepeer-findings/issues/194

```
function forceBurn(address account, uint256 amount) external
    require(hasRole(BURNER_ROLE, _msgSender()), "BURNER role
    _burn(account, amount);
}
```

Using the forceBurn() function of EthErc20, an address with BURNER_ROLE can burn an arbitrary amount of tokens from any address.

We believe this is unnecessary and poses a serious centralization risk.

Proof of Concept

https://github.com/skalenetwork/ima-c4-audit/blob/main/contracts/schain/tokens/EthErc20.sol#L64-L67

Recommended Mitigation Steps

Update forceBurn function for only owner can burn his tokens.

cstrangedk (SKALE) disputed and commented:

> Issue is acknowledged and work is pending on the roadmap. SKALE chain owners must carefully manage BURNER_ROLE, and end-users carefully monitor role.

GalloDaSballo (judge) commented:

> This finding boils down to the flexibility of the role system against a more immutable address being set on the constructor, ultimately the role is contingent on configuration and as such I believe the finding to be valid and of medium severity.

> In the majority of configurations this will be a non-issue, however end users may want to monitor how roles are set and how they change over time to avoid their tokens being burned

[M-09] Not compatible with Rebasing/Deflationary/Inflationary tokens

Submitted by 0x1f8b, also found by cmichel, kirk-baird, gzeon, and lllllll

The `DepositBoxERC20` contract do not appear to support rebasing/deflationary/inflationary tokens whose balance changes during transfers or over time. The necessary checks include at least verifying the amount of tokens transferred to contracts before and after the actual transfer to infer any fees/interest.

Recommended Mitigation Steps

Add support in contracts for such tokens before accepting user-supplied tokens Consider to check before/after balance on the vault.

cstrangedk (SKALE) disputed and commented:

> Issue is acknowledged and is contingent on SKALE Chain owner configuration and evaluation of compatibile tokens.

GalloDaSballo (judge) commented:

> Because this is reliant on configuration, I believe the finding to be valid and of medium severity.

> End users can verify if the DepositBoxes are properly handling rebasing tokens at the time they wish to bridge

[M-10] NFT owner can change token URI

Submitted by cmichel

In the `ERC721OnChain` implementation the *token owner* can set the token's URI using `setTokenURI`.

Usually, this is token URI points to data defining the NFT (attributes, images, etc.). It's usually set by the *contract* owner. A user that owns an NFT can just spoof any other NFT data by changing the token URI to any of the other NFTs.

Recommended Mitigation Steps

Disallow the owner of an NFT to change its token URI

DimaStebaev (SKALE) disputed and commented:

> Acknowledged, `ERC721OnChain` is a default implementation. If the token is sensitive to URI change SKALE chain owner can use another one. Not all ERC721 require that URI can't be changed.

GalloDaSballo (judge) commented:

> Because the argument for this being a setting can be made I am excluding a high severity.
>
> However the code was brought into scope, the implementation under scrutiny does allow the owner to change the URI which is a known admin privilege.
>
> For those reasons I believe the finding to be valid and of medium severity

[M-11] Loss of pending messages (if any) in case removeConnectedChain is called

Submitted by hubble

If there are any unprocessed messages to be executed or processed, while removeConnectedChain is called, then they may be stuck from getting processed on the other end. If these messages have transactions for any token transfer then it will get stuck or lost.

Proof of Concept

Contract : MessageProxy.sol Line : 313

```
function removeConnectedChain(string memory schainName) publ
    bytes32 schainHash = keccak256(abi.encodePacked(schainNa
    require(connectedChains[schainHash].inited, "Chain is nc
    delete connectedChains[schainHash];
```

```
}
```

Recommended Mitigation Steps

Check if there are any pending or unprocessed messages while removeConnectedChain is called and revert in that case. Better to implement some functionality like pause just locally for the chain to be removed, before the actual removeConnectedChain is called.

DimaStebaev (SKALE) commented:

> It duplicates #57

GalloDaSballo (judge) commented:

> I don't believe this to be a duplicate.
>
> I think the finding is valid in that because of the synchronicity of broadcasting messages, the chain could be removed before it receives all messages.
>
> This is a risk that end users do face when interacting with the system and the only use case I could think of would be for a malicious admin to deny certain operations.
>
> That said I don't believe there's any easy solution as this would have to be addressed at the meta level.
>
> I do think the finding is valid and of medium severity

cstrangedk (SKALE) commented: Issue is acknowledged and work is pending on the roadmap to prevent improper use of removeConnectedChain.

Low Risk and Non-Critical Issues

For this contest, 14 reports were submitted by wardens detailing low risk and non-critical issues. The **report highlighted below** by **defsec** received the top score from the judge.

The following wardens also submitted reports: kirk-baird, 0x1f8b, csanuragjain, kyliek, gzeon, robee, ye0lde, cmichel, 0xwags, jayjonah8, leastwood, rfa, *and* kenta.

[L-01] Front-runnable Initializers

All contract **initializers** were missing access controls, allowing any user to initialize the contract. By front-running the contract deployers to initialize the contract, the incorrect parameters may be supplied, leaving the contract needing to be redeployed.

Proof of Concept

1. Navigate to the following contracts.

```
59 https:
60
54 https:
```

2. initialize functions does not have access control. They are vulnerable to front-running.

Recommended Mitigation Steps

While the code that can be run in contract constructors is limited, setting the owner in the contract's constructor to the `msg.sender` and adding the `onlyOwner` modifier to all **initializers** would be a sufficient level of access control.

[L-02] Initializer reentrancy may lead to double initialization

Initializer functions that are invoked separate from contract creation (the most prominent example being minimal proxies) may be reentered if they make an untrusted non-view external call.

Once an initializer has finished running it can never be re-executed. However, an exception put in place to support multiple inheritance made reentrancy possible in the scenario described above, breaking the expectation that there is a single execution.

Note that upgradeable proxies are commonly initialized together with contract creation, where reentrancy is not feasible, so the impact of this issue is believed to be minor.

Proof of Concept

1. Go to contracts directory.
2. On the package.json, openzeppelin 4.3.2 is defined.

https://github.com/skalenetwork/ima-c4-audit/blob/main/package.json

Recommended Mitigation Steps

Avoid untrusted external calls during initialization.

A fix is included in the version v4.4.1 of @openzeppelin/contracts and @openzeppelin/contracts-upgradeable.

Reference

https://github.com/OpenZeppelin/openzeppelin-contracts/security/advisories/GHSA-9c22-pwxw-p6hx

[L-03] ERC1155Supply vulnerability in OpenZeppelin Contracts

When ERC1155 tokens are minted, a callback is invoked on the receiver of those tokens, as required by the spec. When including the ERC1155Supply extension, total supply is not updated until after the callback, thus during the callback the reported total supply is lower than the real number of tokens in circulation.

If a system relies on accurately reported supply, an attacker may be able to mint tokens and invoke that system after receiving the token balance but before the supply is updated.

Proof of Concept

1. Go to contracts directory.
2. On the package.json, openzeppelin 4.3.2 is defined.

https://github.com/skalenetwork/ima-c4-audit/blob/main/package.json

Recommended Mitigation Steps

A fix is included in version 4.3.3 of @openzeppelin/contracts and @openzeppelin/contracts-upgradeable.

Reference

https://github.com/OpenZeppelin/openzeppelin-contracts/security/advisories/GHSA-wmpv-c2jp-j2xg

[L-04] Missing zero-address check in the setter functions and initialize functions

Missing checks for zero-addresses may lead to infunctional protocol, if the variable addresses are updated incorrectly.

Proof of Concept

1. Navigate to the following contracts.

```
253 https:
```

Recommended Mitigation Steps

Consider adding zero-address checks in the discussed constructors: require(newAddr != address(0));.

DimaStebaev (SKALE) commented:

> L-01 and L-02 is described in #2 L-02: only SKALE chain owner are able to deploy smart contracts on it's SKALE chain and this actor is assumed as trusted.

GalloDaSballo (judge) commented:

L-01 : Front-runnable Initializers

> Agree with finding in lack of any mitigation am marking this valid. For the sponsor, this is how you can deploy and set data in one tx: https://github.com/Badger-Finance/badger-strategy-mix-v1/blob/c97eda8cb60d0dcfd62be956a2aab4c86353de65/contracts/proxy/AdminUpgradeabilityProxy.sol#L225

L-02 : Initializer reentrancy may lead to double initialization

> Finding is valid, and mitigation is to upgrade to newer OZ code

L-03 : ERC1155Supply vulnerability in OpenZeppelin Contracts

> Valid

L-04 : Missing zero-address check in the setter functions and initialize functions

> Agree

> e.g. -> Vulnerability in OZ -> See disclosure -> Upgrade to xyz

GalloDaSballo (judge) commented:

> Making this the winner of QA Reports, mostly because it offers some unique findings in a proper QA Format.

> In terms of Findings Kirk-Baird is basically there, I ended up making this report win as it was submitted as QA rather than an aggregate of downgraded findings.

> That said I believe this report could have had a couple extra findings to make it truly a winner.

> L-01: Front-runnable Initializers -> Low

> L-02 : Initializer reentrancy may lead to double initialization -> Low

> L-03 : ERC1155Supply vulnerability in OpenZeppelin Contracts -> Low

> L-04 : Missing zero-address check in the setter functions and initialize functions -> Low

Gas Optimizations

For this contest, 15 reports were submitted by wardens detailing gas optimizations. The report highlighted below by IIIIIII received the top score from the judge.

The following wardens also submitted reports: 0x1f8b, TerrierLover, saian, CertoraInc, robee, m_smirnova2020, rfa, WatchPug, d4rk, ye0lde, gzeon, Tomio, kenta, *and* kyliek.

To check the actual size of the reduction, `hardhat-gas-reporter` is used. (https://www.npmjs.com/package/hardhat-gas-reporter). At each result, it lists how many size of the gas is reduced after the change.

`mainnet/CommunityPool.sol`

[G-01] `refundGasByUser` function can use unchecked directory and slight refactor may reduce gas fee.

By refactoring following code, it can reduce gas cost of `CommunityPool.sol`. https://github.com/skalenetwork/ima-c4-audit/blob/main/contracts/mainnet/CommunityPool.sol#L97-L111

```
    uint amount = tx.gasprice * gas;
    if (amount > _userWallets[user][schainHash]) {
        amount = _userWallets[user][schainHash];
    }
    _userWallets[user][schainHash] = _userWallets[user][schainHash]
    if (!_balanceIsSufficient(schainHash, user, 0)) {
        activeUsers[user][schainHash] = false;
        messageProxy.postOutgoingMessage(
            schainHash,
            schainLinks[schainHash],
            Messages.encodeLockUserMessage(user)
        );
    }
    node.sendValue(amount);
    return (tx.gasprice * gas - amount) / tx.gasprice;
```

First, `_userWallets[user][schainHash] - amount` will never be less than 0 so can be wrapped by unchecked, since `amount` is always equal to or less than `_userWallets[user][schainHash]`. Second, `(tx.gasprice * gas - amount) / tx.gasprice` will not underflown so can be wrapped by unchecked directory, since `tx.gasprice * gas - amount` will be equal to or more than 0.

Here is an example of the modified code:

```
    uint amount = tx.gasprice * gas;
    if (amount > _userWallets[user][schainHash]) {
        amount = _userWallets[user][schainHash];
    }
    unchecked {
        _userWallets[user][schainHash] = _userWallets[user][schainHa
    }
    if (!_balanceIsSufficient(schainHash, user, 0)) {
        activeUsers[user][schainHash] = false;
        messageProxy.postOutgoingMessage(
            schainHash,
            schainLinks[schainHash],
            Messages.encodeLockUserMessage(user)
        );
    }
    node.sendValue(amount);
    unchecked {
        return (tx.gasprice * gas - amount) / tx.gasprice;
    }
```

It simply wraps the above mentioned code with unchecked directory.

Here is the comparison of the gas cost at CommunityPool.sol

- Before: 2114911
- After: 2100269
- Before - After: 14642 (About 0.6% reduction)

[G-02] `withdrawFunds` function can reduce the gas cost by reordering the condition of if statement.

In the if statement, it calls `_balanceIsSufficient` function first. But it can check `activeUsers[msg.sender][schainHash]` to reduce the gas cost slightly.

https://github.com/skalenetwork/ima-c4-audit/blob/main/contracts/mainnet/CommunityPool.sol#L174-L184

```
    if (
        !_balanceIsSufficient(schainHash, msg.sender, 0) &&
        activeUsers[msg.sender][schainHash]
    ) {
        activeUsers[msg.sender][schainHash] = false;
        messageProxy.postOutgoingMessage(
            schainHash,
            schainLinks[schainHash],
            Messages.encodeLockUserMessage(msg.sender)
        );
    }
```

Here is an example of the modification.

```
    if (
        activeUsers[msg.sender][schainHash] &&
        !_balanceIsSufficient(schainHash, msg.sender, 0)
    ) {
        activeUsers[msg.sender][schainHash] = false;
        messageProxy.postOutgoingMessage(
            schainHash,
            schainLinks[schainHash],
            Messages.encodeLockUserMessage(msg.sender)
        );
    }
```

Here is the comparison of the gas cost at CommunityPool.sol

- Before: 2114911
- After: 2113003
- Before - After: 1908 (About 0.09% reduction)

[G-03] `withdrawFunds` function can reduce gas cost by using unchecked

`_userWallets[msg.sender][schainHash] - amount` will never be less than 0, since it checks `amount <= _userWallets[msg.sender][schainHash]` at require function.

https://github.com/skalenetwork/ima-c4-audit/blob/main/contracts/mainnet/CommunityPool.sol#L171-L173

```
require(amount <= _userWallets[msg.sender][schainHash], "Balance
require(!messageProxy.messageInProgress(), "Message is in progre
_userWallets[msg.sender][schainHash] = _userWallets[msg.sender][
```

Here is an example of the modification.

```
require(amount <= _userWallets[msg.sender][schainHash], "Balance
require(!messageProxy.messageInProgress(), "Message is in progre
unchecked {
    _userWallets[msg.sender][schainHash] = _userWallets[msg.senc
}
```

Here is the comparison of the gas cost at CommunityPool.sol

- Before: 2114911
- After: 2107176
- Before - After: 7735 (About 0.36% reduction)

`mainnet/MessageProxy.sol`

[G-04] `i++` can be wrapped by `unchecked` directory.

`mainnet/MessageProxy.sol` and `mainnet/MessageProxyForMainnet.sol` contains for loop, but `i++` can be wrapped by unchecked directory to decrease the

gas cost.

https://github.com/skalenetwork/ima-c4-audit/blob/main/contracts/MessageProxy.sol#L221

https://github.com/skalenetwork/ima-c4-audit/blob/main/contracts/MessageProxy.sol#L515

https://github.com/skalenetwork/ima-c4-audit/blob/main/contracts/mainnet/MessageProxyForMainnet.sol#L118

https://github.com/skalenetwork/ima-c4-audit/blob/main/contracts/mainnet/MessageProxyForMainnet.sol#L235

Here is an example of the modification.

```
    for (uint256 i = 0; i < messages.length; ) {
        data = abi.encodePacked(
            data,
            bytes32(bytes20(messages[i].sender)),
            bytes32(bytes20(messages[i].destinationContract)),
            messages[i].data
        );
        unchecked {
            i++;
        }
    }
```

Here is the comparison of the gas cost at MessageProxyForMainnet.sol

- Before: 3403300
- After: 3388788
- Before - After: 14512 (About 0.4% reduction)

In addition to MessageProxy.sol, here are other opportunities to decrease gas cost by wrapping `i++` by unchecked directory at following files:

https://github.com/skalenetwork/ima-c4-audit/blob/main/contracts/mainnet/Linker.sol#L175

https://github.com/skalenetwork/ima-c4-audit/blob/main/contracts/mainnet/Linker.sol#L100

https://github.com/skalenetwork/ima-c4-audit/blob/main/contracts/mainnet/Linker.sol#L100

https://github.com/skalenetwork/ima-c4-audit/blob/main/contracts/mainnet/DepositBoxes/DepositBoxERC20.sol#L276

https://github.com/skalenetwork/ima-c4-audit/blob/main/contracts/mainnet/DepositBoxes/DepositBoxERC721.sol#L76

https://github.com/skalenetwork/ima-c4-audit/blob/main/contracts/mainnet/DepositBoxes/DepositBoxERC721.sol#L260

https://github.com/skalenetwork/ima-c4-audit/blob/main/contracts/mainnet/DepositBoxes/DepositBoxERC1155.sol#L79

https://github.com/skalenetwork/ima-c4-audit/blob/main/contracts/mainnet/DepositBoxes/DepositBoxERC1155.sol#L275

https://github.com/skalenetwork/ima-c4-audit/blob/main/contracts/mainnet/DepositBoxes/DepositBoxERC1155.sol#L398

https://github.com/skalenetwork/ima-c4-audit/blob/main/contracts/mainnet/DepositBoxes/DepositBoxERC1155.sol#L444

https://github.com/skalenetwork/ima-c4-audit/blob/main/contracts/mainnet/DepositBoxes/DepositBoxERC1155.sol#L459

yavrsky (SKALE) commented:

> We prefer not to use “unchecked” for security reasons, even if it is applicable there. Also only marginal gas improvements.

GalloDaSballo (judge) commented:

> To add #47 which is valued at 62500

GalloDaSballo (judge) commented:

> G-01: Unchecked should save 20 gas per operation * 4 = 80 gas
>
> G-02: Valid but ultimately saves gas on the “bad path”
>
> G-03: Saves 20 gas
>
> G-04 Saves 20 gas per iteration * 4 = 80
>
> 11 * 20 = 220
>
> Additional gas saved: 400
>
> Total Combined: 62900

Disclosures

Top