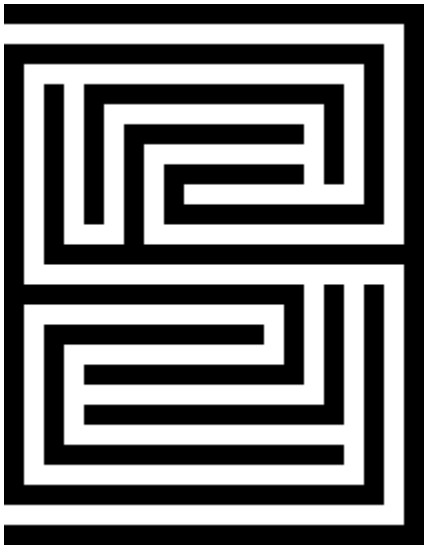

SKALE IMA Security Review Public Report

PROJECT: SKALE IMA Security Review

June 2021

Prepared For:

Chadwick Strange | SKALE Labs

chadwick@skalelabs.com

Prepared By:

Jonathan Haas | Bramah Systems, LLC.

jonathan@bramah.systems

Table of Contents

SKALE IMA Protocol Review

Executive Summary

Scope of Engagement

Bramah Systems, LLC was engaged in June of 2021 to perform a comprehensive security review of the SKALE IMA smart contracts (specific contracts denoted within the appendix). Our review was conducted over a period of three business days by both members of the Bramah Systems, LLC. executive staff.

Bramah Systems completed the assessment using manual, static and dynamic analysis techniques.

Timeline

Review Commencement: June 15, 2021

Report Delivery: June 18, 2021

Engagement Goals

The primary scope of the engagement was to evaluate and establish the overall security of the SKALE IMA protocol, with a specific focus on trading actions. In specific, the engagement sought to answer the following questions:

- Is it possible for an attacker to steal or freeze tokens?
- Does the Solidity code match the specification as provided?
- Is there a way to interfere with the contract mechanisms?
- Are the arithmetic calculations trustworthy?

Contract Specification

Specification was provided in the form of code comments. The contracts were provided via GitHub (commit hash **1811a93b0bca47aa7c999974c3025211c997228b**)

Overall Assessment

Bramah Systems was engaged to evaluate and identify any potential security concerns within the codebase of the SKALE IMA protocol. During the course of our engagement, Bramah Systems found multiple instances wherein the team deviated materially from established best practices and procedures of secure software development within DLT. The team has since addressed these issues with a resolution or risk acceptance.

Disclaimer

As of the date of publication, the information provided in this report reflects the presently held, commercially reasonable understanding of Bramah Systems, LLC.'s knowledge of security patterns as they relate to the SKALE IMA Protocol, with the understanding that distributed ledger technologies ("DLT") remain under frequent and continual development, and resultantly carry with them unknown technical risks and flaws. The scope of the review provided herein is limited solely to items denoted within "Scope of Engagement" and contained within "Directory Structure". The report does NOT cover, review, or opine upon security considerations unique to the Solidity compiler, tools used in the development of the protocol, or distributed ledger technologies themselves, or to any other matters not specifically covered in this report.

The contents of this report must NOT be construed as investment advice or advice of any other kind. This report does NOT have any bearing upon the potential economics of the SKALE IMA protocol or any other relevant product, service or asset of SKALE IMA or otherwise. This report is not and should not be relied upon by SKALE IMA or any reader of this report as any form of financial, tax, legal, regulatory, or other advice.

To the full extent permissible by applicable law, Bramah Systems, LLC. disclaims all warranties, express or implied. The information in this report is provided "as is" without warranty, representation, or guarantee of any kind, including the accuracy of the information provided. Bramah Systems, LLC. makes no warranties, representations, or guarantees about the SKALE IMA Protocol. Use of this report and/or any of the information provided herein is at the users sole risk, and Bramah Systems, LLC. hereby disclaims, and each user of this report hereby waives, releases, and holds Bramah Systems, LLC. harmless from, any and all liability, damage, expense, or harm (actual, threatened, or claimed) from such use.

Timeliness of Content

All content within this report is presented only as of the date published or indicated, to the commercially reasonable knowledge of Bramah Systems, LLC. as of such date, and may be superseded by subsequent events or for other reasons. The content contained within this report is subject to change without notice. Bramah Systems, LLC. does not guarantee or warrant the accuracy or timeliness of any of the content contained within this report, whether accessed through digital means or otherwise.

Bramah Systems, LLC. is not responsible for setting individual browser cache settings nor can it ensure any parties beyond those individuals directly listed within this report are receiving the most recent content as reasonably understood by Bramah Systems, LLC. as of the date this report is provided to such individuals.

General Recommendations

Best Practices & Solidity Development Guidelines

Solidity version misses more recent security updates

The Solidity version used, **0.6.12**, is multiple versions out of date and lacks security updates that have been introduced in later Solidity versions.

Resolution: This concern has been resolved in the PR:
https://github.com/skalenetwork/IMA/pull/678

Unbounded for loops can result in resource exhaustion

Throughout the protocol, there are a number of for loops that have a dynamic length (for example, within **connectSchain**). Unless an upper bound is provided to these loops, they can result in resource exhaustion after too many iterations through the loop.

Resolution: This concern has been resolved in the PR:
https://github.com/skalenetwork/IMA/pull/699

TODO Item remains in source

Multiple lines reference TODO items that should be removed prior to a production deployment. These items should be removed accordingly.

Resolution: This concern has been resolved in the PR:
https://github.com/skalenetwork/IMA/pull/658

Commented out code should be removed.

The multiple instances of commented out code that exist within

proxy/contracts/schain/bls/FieldOperations.sol should be removed.

Resolution: This concern has been resolved in the PR:
https://github.com/skalenetwork/IMA/pull/659

Usage of transfer considered against best practice

The Istanbul hardfork repriced certain opcodes, breaking many contracts that are running on mainnet today. As transfer is now considered against best practices, **sendValue** (by OpenZeppelin) should be used. As the primary difference between **sendValue** and **transfer** is that **sendValue** will forward all gas to the recipient, because of this, one must be more aware of possible reentrancy vulnerabilities. Consider using the **ReentrancyGuard** and **PullPayment** contracts to mitigate this impact.

Resolution: This concern has been resolved in the PR:
https://github.com/skalenetwork/IMA/pull/691

Sensitive variable changes should emit an event

Sensitive functions that change aspects that can materially impact flow of funds to users or the protocol (such as **setMinTransactionGas**) should emit an event.

Resolution:This concern has been resolved in the PR:
https://github.com/skalenetwork/IMA/pull/678

Specific Recommendations

Unique to the SKALE IMA Protocol

Double negative in withoutWhitelist could lead to confusion

One enables the whitelist of tokens by setting the **withoutWhitelist[schainName]** to false. This could lead to potential confusion, and it is suggested that instead **WhitelistedChain** or similar be used, wherein a whitelisted address will be marked as true to distinguish that it is whitelisted.

Resolution: This concern has been resolved in the PR: https://github.com/skalenetwork/IMA/pull/696

Boolean toggles can be combined

The **enableWhitelist** and **disableWhitelist** functions as well as the **enableAutomaticDeploy** and **disableAutomaticDeploy** functions can be combined into a function that reads the active state of the value and toggles it appropriately in order to save gas. This would prevent a scenario in which an individual attempts to whitelist an already whitelisted schain.

Resolution: The SKALE team provided the following: “Whitelisting and Automatic deploy are major states. To avoid confusion and ensure Dapp Developers using IMA set the correctly intended state, we recommend using different functions to enable/disable states. Further, enable/disableWhitelist tends to be more human-readable and explicit.”

KillProcess.Active could be rewritten for clarity

Currently when a schain is not being killed it falls within the **KillProcess.Active** distinction.

This state could be modified to make it clear that the schain is not actively being killed, which the name could suggest.

Resolution: This concern has been resolved in the PR: https://github.com/skalenetwork/IMA/pull/697

Require messages could be rewritten for clarity

The following require statements can have their message rewritten for clarity. In each instance, the variables associated with the required statement do not suggest a clear correlation with the message.

Within **proxy/contracts/mainnet/MessageProxyForMainnet.sol**:

```
require(
        schainHash != MAINNET_HASH,
        "SKALE chain name is incorrect. Inside in MessageProxy"
    );
```

Clarity can be introduced to the second sentence.

Within **proxy/contracts/schain/TokenManagers/TokenManagerEth.sol**:

```
require(ethErc20 != newEthErc20Address, "The same address");
```

Clarity can be given as to why the same address causes a failure here.

Within **proxy/contracts/schain/CommunityLocker.sol**:

require(activeUsers[message.receiver] != message.isActive, "User statuses must be different");

Clarity can be introduced as to whether or not this applies beyond **isActive**.

Resolution: This concern has been resolved in the PR https://github.com/skalenetwork/IMA/pull/677

Hardcoded gas costs should dynamically generated

The message proxy contains a number of hardcoded gas costs. Due to the constantly changing nature of the Ethereum blockchain and prior instances in which various opcodes changed relevant gas associated, we suggest that these values be dynamically generated.

headerMessageGasCost = 70000;

messageGasCost = 8790;

gasLimit = 1000000;

Resolution: This risk has been acknowledged by the SKALE team.

Potential for integer overflow

The following statement can overflow if contractOnSchain.totalSupply() added to the amount exceeds the integer limit. This can result in the **require** statement incorrectly evaluating that the total supply was not exceeded, despite having done so.

require(

```
        contractOnSchain.totalSupply() + amount <=
totalSupplyOnMainnet[contractOnSchain],
        "Total supply exceeded"
    );
```

Resolution: This concern has been resolved in the PR: https://github.com/skalenetwork/IMA/pull/698

Mainnet name is case sensitive

The **MAINNET_NAME** is case sensitive, which can result in issue a different hash depending upon the value provided (by default, “Mainnet”). This should be made clear in the comments.

Resolution: This is expected and intentional by the system design.

Blocking mechanism relies upon address construct

Blocking of individuals from performing a transfer without paying the requisite gas fees (instead taking from the community pool) are blocked on a per address basis. While this block could prevent certain griefing actions, an individual (while paying the initial gas fees) could repeatedly attack the protocol with new addresses.

Resolution: The team provided the following justification ”To use a new address adversary will have to deposit ETH to the system. Old accounts are blocked with positive balance before they run out of ETH to pay for gas.
Therefore [the] described attack is theoretical and is expensive.”

Bramah concurs that successful execution of this attack would require additional funds, and resultantly poses less concern.

Toolset Warnings

Unique to the SKALE IMA Protocol

Overview

In addition to our manual review, our process involves utilizing static analysis and formal methods in order to perform additional verification of the presence of security vulnerabilities (or lack thereof). An additional part of this review phase consists of reviewing any automated unit testing frameworks that exist.

The following sections detail warnings generated by the automated tools and confirmation of false positives where applicable.

Compilation Warnings

No compilation warnings were encountered during the course of our audit.

Test Coverage

The contracts possess a number of functional unit tests encompassing various stages of the application lifecycle.

Static Analysis Coverage

The contract repository underwent heavy scrutiny with multiple static analysis agents, including:

- Securify
- MAIAN
- Mythril
- Oyente
- Slither

Directory Structure

At time of review, the directory structure of the SKALE IMA smart contracts repository appeared as it does below. Our review, at request of SKALE IMA, covers the Solidity code (*.sol) as of commit hash **1811a93b0bca47aa7c999974c3025211c997228b**

.

```
├── Messages.sol
├── Migrations.sol
├── extensions
│   ├── ERC721ReferenceMintAndMetadataMainnet.sol
│   ├── ERC721ReferenceMintAndMetadataSchain.sol
│   └── interfaces
│       ├── MessageProxyClient.sol
│       ├── MessageReceiver.sol
│       └── MessageSender.sol
├── interfaces
│   ├── IMainnetContract.sol
│   ├── IMessageProxy.sol
│   └── IMessageReceiver.sol
├── mainnet
│   ├── CommunityPool.sol
│   ├── DepositBox.sol
│   ├── DepositBoxes
│   │   ├── DepositBoxERC1155.sol
│   │   ├── DepositBoxERC20.sol
│   │   ├── DepositBoxERC721.sol
│   │   └── DepositBoxEth.sol
│   ├── Linker.sol
│   ├── MessageProxyForMainnet.sol
```

```
│   ├── SkaleManagerClient.sol
│   └── Twin.sol
├── schain
│   ├── CommunityLocker.sol
│   ├── KeyStorage.sol
│   ├── MessageProxyForSchain.sol
│   ├── TokenManager.sol
│   ├── TokenManagerLinker.sol
│   ├── TokenManagers
│   │   ├── TokenManagerERC1155.sol
│   │   ├── TokenManagerERC20.sol
│   │   ├── TokenManagerERC721.sol
│   │   └── TokenManagerEth.sol
│   ├── bls
│   │   ├── FieldOperations.sol
│   │   ├── Precompiled.sol
│   │   └── SkaleVerifier.sol
│   └── tokens
│       ├── ERC1155OnChain.sol
│       ├── ERC20OnChain.sol
│       ├── ERC721OnChain.sol
│       └── EthErc20.sol
└── test
    ├── ConfigReader.sol
    ├── KeyStorageMock.sol
    ├── Logger.sol
    ├── MessageProxyForMainnetTester.sol
    ├── MessageProxyForSchainTester.sol
    ├── MessageProxyForSchainWithoutSignature.sol
```

```
├── MessagesTester.sol
├── PrecompiledMock.sol
├── ReceiverGasLimitMainnetMock.sol
├── ReceiverGasLimitSchainMock.sol
├── ReceiverMock.sol
├── SkaleVerifierMock.sol
├── TestContractManager.sol
├── TestNodes.sol
├── TestSchains.sol
├── TestSchainsInternal.sol
└── TestWallets.sol
```

10 directories, 53 files