Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Sygma

Sygma Bridge Handlers

Veridise Inc.
September 13, 2024

► **Prepared For:**

ChainSafe
https://buildwithsygma.com/

► **Prepared By:**

Ajinkya Rajput
Evgeniy Shishkin
Jon Stephens

► **Contact Us:**

contact@veridise.com

► **Version History:**

Sep. 13, 2024	V1
Sep. 09, 2024	Initial Draft

Contents

Contents iii

1 Executive Summary 1

2 Project Dashboard 3

3 Security Assessment Goals and Scope 5

3.1 Security Assessment Goals 5
3.2 Security Assessment Methodology & Scope 5
3.3 Classification of Vulnerabilities 7

4 Vulnerability Report 9

4.1 Detailed Description of Issues 10
4.1.1 V-SBH-VUL-001: Leftover DMR tokens are not properly cleaned up . . . 10
4.1.2 V-SBH-VUL-002: Resources can be overwritten by mistake 12
4.1.3 V-SBH-VUL-003: Insufficient input arguments check 13
4.1.4 V-SBH-VUL-004: Missing address zero-checks 14
4.1.5 V-SBH-VUL-005: Missing events on state updates 15
4.1.6 V-SBH-VUL-006: convertToInternalBalance return value inconsistency . 16
4.1.7 V-SBH-VUL-007: Inconsistency in bytes encoding may lead to errors . . 17
4.1.8 V-SBH-VUL-008: Typos and incorrect comments 18

Executive Summary 1

From Aug. 21, 2024 to Aug. 28, 2024, ChainSafe engaged Veridise to conduct a security assessment of several modules of their Sygma Bridge Handlers, together with smart contracts of the Sprinter product.

Veridise conducted the security assessment over 12 person-days, with 2 security analysts reviewing the project over 6 days on commit `204f1378`. The security assessment methodology involved a tool-assisted analysis of the program source code performed by Veridise security analysts as well as thorough code review.

Project Summary. This security assessment addressed two loosely related products that are briefly described here.

Sprinter Escrow Contracts. These contracts are a part of a system called `Sprinter`. The system is designed to help users perform actions with their tokens that are potentially distributed across multiple chains. For instance, a user might have several valuable tokens (such as stablecoins) spread across different chains, and they wish to purchase an NFT on some chain they don't even have a wallet for. In such scenarios, `Sprinter` will assist in aggregating all the funds on the target chain and executing a required purchase transaction on a destination smart contract. The Sprinter Escrow Contracts are responsible for accumulating funds and executing the expected call when enough funds get accumulated.

Sygma Bridge Handlers. The `Sygma Bridge` is a software that allows users to conveniently transfer their on-chain assets, such as ERC20-compatible tokens and NFTs, between different blockchains. Handlers are modules that are responsible for handling specific types of assets.

- The `ERC20Handler` module is a handler responsible for swaps of ERC20-compatible tokens.
- The `DefaultMessageReceiver` module is a component of `ERC20Handler` that allows users to perform complex, programmable actions on their swapped tokens in the destination chain. These actions include swaps, ERC20 transfers, and native Eth transfers, all within a single transaction.

Please note that the integration of handlers into the `Sygma Bridge` was not within the scope of this security assessment.

Code Assessment. The ChainSafe developers provided the source code of all the smart contracts for the code review. The source code appears to be original. It contains some low-level documentation in a form of the README file and documentation comments on functions and storage variables. A more general, high-level documentation of the project was also available on the project's website. However, there was no specific documentation that explained the higher-level purpose of the smart contracts.

The source code contains a test suite, which implements several integration tests exercising the most critical execution paths.

Summary of Issues Detected. The security assessment uncovered 8 issues. The most severe issue among those is V-SBH-VUL-001, which, in some rare circumstances, may allow an attacker to steal some of the user tokens. The security analysts also identified 3 low-severity issues, 3 warnings, and 1 informational finding . The developers of Sygma Bridge Handlers have fixed most of the findings.

Recommendations. After conducting the security assessment of the protocol, the security analysts had a few suggestions to improve the Sygma Bridge Handlers:

1. Provide a high-level description of each handler that would clarify its purpose and the most important aspects of its behavior in the form of comments.
2. Provide tests for the `DefaultMessageReceiver` contract.
3. The protocol accepts arguments as byte arrays in several places, however the exact format is not always specified. It is suggested to clearly state the expected format of those byte arrays, and to mention a potential issue of using `abi.encode` function due to its alignment behavior in this setting.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
Sygma Bridge Handlers	`204f1378`	Solidity	Ethereum

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Aug. 21–Aug. 28, 2024	Manual & Tools	2	12 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Acknowledged	Fixed
Critical-Severity Issues	0	0	0
High-Severity Issues	0	0	0
Medium-Severity Issues	1	1	1
Low-Severity Issues	3	3	2
Warning-Severity Issues	3	3	3
Informational-Severity Issues	1	1	1
TOTAL	8	8	7

Table 2.4: Category Breakdown.

Name	Number
Data Validation	3
Maintainability	3
Logic Error	1
Missing/Incorrect Events	1

3 Security Assessment Goals and Scope

3.1 Security Assessment Goals

The engagement was scoped to provide a security assessment of several ChainSafe smart contracts. During the security assessment, the security analysts aimed to answer questions such as:

- Are there any Solidity-specific code defects that could affect the security of the system?
- Is the any divergence between the described business logic and the program code implementation?
- Is the testing coverage sufficient?

Specifically to `Gopher Escrow Contracts`, the security analysts aimed to answer questions such as:

- Is the `Gopher Factory Contract` susceptible to a front-running attack, when attacker could deploy a target `Gopher Escrow` contract before the factory does this?
- Is the request hashing implementation susceptible to any kind of manipulations?
- Is it possible to execute a user's request several times, by replaying an already sent request?
- Are there any reentrancy possibilities?

Specifically to `Sygma Bridge Handler` contracts, the security analysts aimed to answer questions such as:

- Does there exist a context in which the execution of a swap request may result in an asset loss for the `ERC20Handler` and `DefaultMessageReceiver`?
- Does the input data get properly validated before being processed?
- Are all the required safeguards, such as zero address protection, in place?

3.2 Security Assessment Methodology & Scope

Security Assessment Methodology. To address the questions above, the security assessment involved a combination of human experts and automated program analysis & testing tools. In particular, the security assessment was conducted with the aid of the following techniques:

- *Static analysis.* To identify potential common vulnerabilities, security analysts leveraged the custom smart contract analysis tool `Vanguard`, as well as the open-source tool `Slither`. These tools are designed to find instances of common smart contract vulnerabilities, such as reentrancy, uninitialized variables, unused functions, etc.

Scope. The scope of this security assessment consists of two parts.

The first part is called `Gopher Escrow Contracts`* of the Sprinter product, consisting of two files:

* They were renamed into Sprinter Escrow Contracts recently

- `GopherEscrow.sol`
- `GopherEscrowFactory.sol`

The second part is `Sygma Bridge Handlers` consisting of the following files:

- `ERC20Handler.sol`
- `DefaultMessageReceiver.sol`
- `ERCHandlerHelpers.sol`
- `ExcessivelySafeCall.sol`
- `ERC20Safe.sol`

Please note that these modules constitute an important part of the `Sygma Bridge`, however, the question of proper *integration* of those modules into the Bridge was not part of this security assessment scope.

Methodology. Veridise security analysts reviewed the reports of previous audits for Sygma Bridge Handlers, inspected the provided tests, and read the Sygma Bridge Handlers documentation. They then began a code review of the code assisted by static analyzers. During the security assessment, the Veridise security analysts asked both technical and contextual questions about the system on Telegram, and the developers provided prompt and helpful responses.

3.3 Classification of Vulnerabilities

When Veridise security analysts discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise.

The severity of a vulnerability is evaluated according to the Table 3.1.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

The likelihood of a vulnerability is evaluated according to the Table 3.2.

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

The impact of a vulnerability is evaluated according to the Table 3.3:

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Vulnerability Report

In this section, the vulnerabilities found during the security assessment are presented. For each issue found, the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.) is specified. Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-SBH-VUL-001	Leftover DMR tokens are not properly . . .	Medium	Fixed
V-SBH-VUL-002	Resources can be overwritten by mistake	Low	Acknowledged
V-SBH-VUL-003	Insufficient input arguments check	Low	Fixed
V-SBH-VUL-004	Missing address zero-checks	Low	Fixed
V-SBH-VUL-005	Missing events on state updates	Warning	Fixed
V-SBH-VUL-006	convertToInternalBalance return value . . .	Warning	Fixed
V-SBH-VUL-007	Inconsistency in bytes encoding may lead . . .	Warning	Fixed
V-SBH-VUL-008	Typos and incorrect comments	Info	Fixed

4.1 Detailed Description of Issues

4.1.1 V-SBH-VUL-001: Leftover DMR tokens are not properly cleaned up

Severity	Medium	**Commit**	204f137
Type	Logic Error	**Status**	Fixed
File(s)	DefaultMessageReceiver.sol		
Location(s)	performActions		
Confirmed Fix At	37d6af8		

The `ERC20Handler` contract is responsible for facilitating the swap operation of ERC20 tokens between different blockchains. In addition to the standard functionality of a swap, this contract allows users to perform a complex series of token management operations as part of the same transaction. To execute this series of operations, the user provides an array of actions, each specifying the type of operation to be performed next.

All such operations are carried out using the `DefaultMessageReceiver` contract, which acts as an intermediary between the destination user's wallet and the Bridge. Note that the same contract is used for all transactions, regardless of the user performing the swap on the Sygma Bridge.

Since anyone can operate on this contract, it is essential that after the last user has completed their swap operation and has used the `DefaultMessageReceiver` function, no funds should remain in the balance of this contract. Otherwise, a malicious user could take what is left by performing the corresponding actions.

The developers of `DefaultMessageReceiver` have implemented countermeasures to protect against this: both the remaining native ETH funds and the swapped ERC-20 tokens are sent to the addresses provided by the user.

However, there is another type of funds that also need to be managed carefully: approved ERC20 tokens that are sent to the `DefaultMessageReceiver` when external calls are made from within the handler. These tokens are not returned to the sender and remain on the balance of the `DefaultMessageReceiver`, so they can potentially be taken out by a malicious user. This functionality is not present.

```
function performActions(
    address tokenSent,
    address payable receiver,
    uint256 startingNativeBalance,
    Action[] memory actions
) external {
    if (msg.sender != address(this))
        revert InsufficientPermission();

    uint256 numActions = actions.length;
    for (uint256 i = 0; i < numActions; i++) {
        // ...skipped...
        approveERC20(IERC20(actions[i].tokenSend),
                     actions[i].approveTo,
                     type(uint256).max);
        // ...skipped
    }
    if (address(this).balance > startingNativeBalance) {
        transferNativeBalance(receiver);
    }
    transferBalance(tokenSent, receiver);
    returnLeftOvers(actions, receiver);
}
```

Snippet 4.1: Snippet from `performActions()`

This contract intends to maintains an invariant that before and after all actions are performed, all balances of the `DefaultMessageReceiver` contract are zero. To do that, the contract returns any leftover tokens to `reciever` address as shown below. However, the `tokenSend` tokens are not returned.

```
function returnLeftOvers(Action[] memory actions, address payable receiver)
  internal {
    for (uint256 i; i < actions.length; i++) {
        transferBalance(actions[i].tokenReceive, receiver);
        approveERC20(IERC20(actions[i].tokenSend), actions[i].approveTo, 0);
    }
}
```

Snippet 4.2: Snippet from `returnLeftOvers()`

Impact Since the balance of the `DefaultMessageReceiver` contract for `tokenSend` tokens may be used only partially, the contract will keep accumulating tokens, and since it approves max balance in its approve call, an attacker can steal all of these leftover tokens.

Recommendation Return `tokenSend` tokens in `returnLeftOvers()`

Developer Response The developers have implemented a fix for this issue.

4.1.2 V-SBH-VUL-002: Resources can be overwritten by mistake

Severity	Low	**Commit**	204f137
Type	Data Validation	**Status**	Acknowledged
File(s)	`ERC20Handler.sol`		
Location(s)	setResource		
Confirmed Fix At	N/A		

Sygma Bridge uses a unified system of token identifiers across different blockchain networks, known as resources. Each network has its own unique token address for each identifier, which is assigned when the resource is initialized. To the best of our understanding, those identifiers are not meant to be changed after being initialized, or at least such a change would be a rare occurrence. There is no mechanism in place to prevent setting an already-initialized resource in the `ERC20Handler`.

Impact If, by mistake, the identifier is reinitialized with the wrong token address, it may at worst lead to an incorrect swap operation.

Recommendation If a change to an already initialized resource is expected, it is recommended to create a separate function that can be called in those rare cases. Otherwise, it is recommended to implement a check that prevents the resource from being reinitialized.

Developer Response The developers acknowledged the issue, but decide to rely on the operational procedures to mitigate it.

4.1.3 V-SBH-VUL-003: Insufficient input arguments check

Severity	Low	**Commit**	204f137
Type	Data Validation	**Status**	Fixed
File(s)	`DefaultMessageHandler.sol`		
Location(s)	performActions()		
Confirmed Fix At	a94faf9		

The following places in the code were identified to have insufficient input data validation.

- In the `DefaultMessageReceiver` contract, the `performActions` function does not check the `receiver` address not be equal to `address(0)` or the `DefaultMessageReceiver` address.
- In the `ERC20Handler` contract, the function `executeProposals` allows the destination address length to be of an arbitrary length. However, for the EVM chains, this has to always be equal to 20 bytes.

Impact It is highly unlikely that providing arguments that would violate the above-mentioned constraints will occur. However, if this does happen, it can lead to two possible outcomes: tokens being locked, in the case of sending them to `address(0)`, or being stolen, in the case where the address of the `DefaultMessageReceiver` is specified as the recipient.

Recommendation It is recommended to implement the aforementioned checks.

Developer Response The developers implemented a fix for the `ERC20Handler` part of the issue, while the behavior of the `DefaultMessageReceiver` was left intact to follow the original intent.

4.1.4 V-SBH-VUL-004: Missing address zero-checks

Severity	Low	**Commit**	204f137
Type	Data Validation	**Status**	Fixed
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	a94faf9		

Description The following functions take addresses as arguments, but do not validate that the addresses are non-zero:

- `ERC20Handler.sol`
  - `setResource()`: `contractAddress` parameter is not validated
  - `withdraw()`: `recipient` address parameter encoded in the byte string is not validated

Impact Using a zero address by mistake is very likely to cause undesirable consequences, such as loss of tokens or system malfunction.

Recommendation It is recommended to implement zero address checks.

Developer Response The developers have implemented a fix for this issue.

4.1.5 V-SBH-VUL-005: Missing events on state updates

Severity	Warning	**Commit**	204f137
Type	Missing/Incorrect Eve	**Status**	Fixed
File(s)	`GopherEscrow.sol , DefaultMessageHandler.sol`		
Location(s)	See description		
Confirmed Fix At	324cff6		

Upon a state update, it is strongly recommended that developers emit an event to indicate that a change was made. Doing so allows both external users and protocol administrators to monitor the protocol for a variety of reasons, including for potentially suspicious activity. It is therefore critical for significant changes to the protocol to be accompanied with events to enable this monitoring.

It was identified that the following functions do not emit events:

1. In the `GopherEscrow` contract, the function `execute` does not emit an event.
2. In the `DefaultMessageReceiver` contract, the function `performActions` does not emit an event on a successful action execution, despite the event named `ActionPerformed` being defined in the contract.

Impact Insufficient event production hinders monitoring of the system both from the perspective of operators and DApps.

Recommendation It is recommended to implement the aforementioned events.

Developer Response The developers have implemented a fix for this issue.

4.1.6 V-SBH-VUL-006: convertToInternalBalance return value inconsistency

Severity	Warning	**Commit**	204f137
Type	Maintainability	**Status**	Fixed
File(s)	ERCHandlerHelpers.sol		
Location(s)	convertToInternalBalance()		
Confirmed Fix At	a94faf9		

The `ERC20Handler` contract uses two utility functions `convertToInternalBalance()` and `convertToExternalBalance()` to normalize the decimals of the users balances to `defaultDecimals` .

While `convertToExternalBalance()` returns an `uint256` value and is the final balance amount after normalizing, `convertToInternalBalance()` returns a byte string.

Moreover, if the token already has `defaultDecimals`, `convertToInternalBalance()` returns an empty string, otherwise it returns new balance in byte string encoded format.

```
function convertToInternalBalance(address tokenAddress, uint256 amount) internal view
     returns(bytes memory) {
    Decimals memory decimals = _tokenContractAddressToTokenProperties[tokenAddress].
    decimals;
    uint256 convertedBalance;
    if (!decimals.isSet) {
        return "";
    } else if (decimals.externalDecimals >= defaultDecimals) {
        convertedBalance =  amount / (10 ** (decimals.externalDecimals -
    defaultDecimals));
    } else {
        convertedBalance = amount * (10 ** (defaultDecimals - decimals.
    externalDecimals));
    }

    return abi.encodePacked(convertedBalance);
}
```

Snippet 4.3: Snippet from `convertToInternalBalance()`

Impact This could cause confusion for users, as the returned value depends on which code path is executed.

Recommendation To avoid possible confusion, it is recommended to make the function always return the same value on all execution paths. This value should be the adjusted balance.

Developer Response The developers have implemented a fix for this issue.

4.1.7 V-SBH-VUL-007: Inconsistency in bytes encoding may lead to errors

Severity	Warning	**Commit**	204f137
Type	Maintainability	**Status**	Fixed
File(s)	ERC20Handler.sol		
Location(s)	setResource , executeProposals		
Confirmed Fix At	a94faf9		

The `ERC20Handler` contract uses a bytes array to pass parameter values to functions in several parts of the code. If the input parameters are encoded using the `abi.encode()` function, the functions might not work correctly because of different assumptions about the encoding scheme. Since there is no specification of the encoding scheme anywhere, developers might easily overlook it and assume that the encoding follows the standard `abi.encode()` method, which is usually the case for `bytes` arguments.

Consider the following example:

```
function setResource(bytes32 resourceID, address contractAddress, bytes calldata args
    )
external onlyBridge {
    _setResource(resourceID, contractAddress);
    if (args.length > 0) {
        uint8 externalTokenDecimals = uint8(bytes1(args));
        _setDecimals(contractAddress, externalTokenDecimals);
    }
}
```

Snippet 4.4: Snippet from `ERC20Handler` contract

If the supplied value for the `externalTokenDecimals` get encoded using the standard `abi.encode(decimals, (uint8))` , the above code will return `0` for decimals value, since `abi.encode()` does alignment of the data to the 32 bytes boundary.

Another instance of this issue was identified in the `executeProposals` function. This function expects a bytes-like data argument containing all the necessary information encoded according to a specific layout specified in comments. However, it can be easily overlooked that the expected format differs from what the `abi.encode()` function provides.

Impact If developers assume that the data argument is expected to be encoded using the standard `abi.encode()` scheme, this could potentially lead to asset loss.

Recommendation If there is a valid reason for using a custom-made encoding scheme instead of the standard `abi.encode()` function, it is important to clearly state this in comments and other relevant documentation, to rule out any possible confusion.

Developer Response The developers have implemented a fix for this issue.

4.1.8 V-SBH-VUL-008: Typos and incorrect comments

Severity	Info	**Commit**	204f137
Type	Maintainability	**Status**	Fixed
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	a94faf9		

Description In the following locations, security analysts identified minor typos and potentially misleading comments:

- `ERC20HandlerHelpers.sol`
  - `convertToInternalBalance()`: The comment describing the function is either incorrect or inaccurate.
- `ERC20Handler.sol`
  - `deposit()`: The comment describing return value is inaccurate. The return data is empty only if the decimals value is not set for a token.

Impact Misleading comments can hinder understanding of the codebase by developers and potential users.

Recommendation It is recommended to fix the comments.

Developer Response The developers have implemented a fix for this issue.