Zellic

Prepared for
Lu Zhang
Mysten Labs

Prepared by
Filippo Cremonese
Junyi Wang
Zellic

April 29, 2024

SuiBridge V1

Smart Contract Security Assessment

Contents

About Zellic 4

1. Overview 4

1.1. Executive Summary 5

1.2. Goals of the Assessment 5

1.3. Non-goals and Limitations 5

1.4. Results 5

2. Introduction 6

2.1. About SuiBridge V1 7

2.2. Methodology 7

2.3. Scope 9

2.4. Project Overview 9

2.5. Project Timeline 10

3. Detailed Findings 10

3.1. Reentrancy issue leading to contract takeover 11

3.2. Rate limiter could semipermanently lock big transfers 13

3.3. Potentially unsafe ERC-20 usage 15

3.4. Missing committee-size limit 16

3.5. Weak `claim_token` access control 17

4. Discussion 17

4.1. Bridge-design maturity 18

4.2. Deployment-procedure improvement suggestion 19

4.3. Minor issues 20

4.4. Code update review 20

5. Threat Model 27

5.1. Intro 28

5.2. Contract: SuiBridge 29

5.3. Introduction 33

5.4. The Security Model 35

6. Assessment Results 36

6.1. Disclaimer 37

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for Mysten Labs from March 11th to March 28th, 2024. During this engagement, Zellic reviewed SuiBridge V1's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Can funds be stuck in the bridge, whether permanently or temporarily?
- Is an attacker able to steal escrowed funds from the bridge?
- Is there any scenario where the bridge becomes locked-up?
- Can the bridge be attacked by malicious committee members?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components for interacting with the bridge
- Scenarios where a significant portion of the committee becomes malicious
- Key custody of committee members
- The security of the original issuing issuer of bridged tokens

The threat model of the bridge contracts assumes only non malicious, explicitly allowed assets are used. This is especially important for the Ethereum contracts, which were audited under the assumption that only the ERC20 assets referenced in the codebase will be supported by the bridge.

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

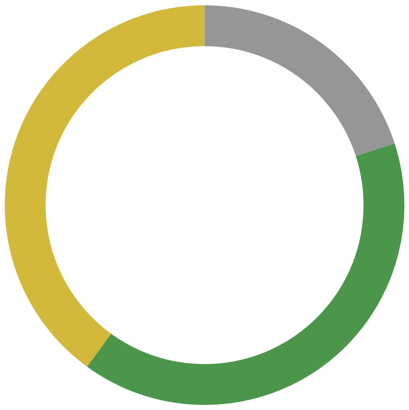
1.4. Results

During our assessment on the scoped SuiBridge V1 contracts, we discovered five findings. No critical issues were found. Two findings were of medium impact, two were of low impact, and the remaining finding was informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for Mysten Labs's benefit in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
🟥 Critical	0
🟧 High	0
🟨 Medium	2
🟩 Low	2
⬜ Informational	1



2. Introduction

2.1. About SuiBridge V1

Mysten Labs contributed the following description of SuiBridge V1:

SuiBridge is a trustless native bridge leveraging Sui's security model. Traditionally, Ethereum is the hub of the vast majority of blockchain digital assets and, in practice, most of the assets bridged into Sui have come from Ethereum. Hence, creating a direct connection to the world's main digital asset hub, together with Sui's security assumptions, delivers a trustworthy Sui Bridge aimed at further bootstrapping Sui's digital asset ecosystem.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped contracts itself. These observations — found in the Discussion (4. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

SuiBridge V1 Contracts

Repository	https://github.com/MystenLabs/sui ↗
Version	sui: `8f29360f304f10d283f7c5cca7501804c2efed49`
Programs	• bridge/evm/contracts/**.sol • bridge/move/**.move • crates/sui-framework/packages/bridge/sources/*.move • crates/sui-bridge/**.rs • sui-execution/latest/sui-adapter/src/execution_engine.rs
Type	Solidity, Move, Rust
Platform	EVM-compatible, Sui

2.4. Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of three person-weeks. The assessment was conducted over the course of three calendar weeks.

Contact Information

The following project manager was associated with the engagement:

Chad McDonald
Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

Filippo Cremonese
Engineer
fcremo@zellic.io

Junyi Wang
Engineer
junyi@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

March 11, 2024	Kick-off call
March 11, 2024	Start of primary review period
March 28, 2024	End of primary review period

3. Detailed Findings

3.1. Reentrancy issue leading to contract takeover

Target	CommitteeUpgradeable		
Category	Coding Mistakes	**Severity**	Critical
Likelihood	Low	**Impact**	Medium

Description

The bridge Ethereum contracts are upgradable and technically implement the UUPS standard. The upgradability feature is implemented in the common CommitteeUpgradeable contract, which is inherited by other bridge contracts and inherits from the OpenZeppelin UUPSUpgradeable standard contract.

To initiate an upgrade, users need to call the `upgradeWithSignatures` function with a message describing the new implementation address, as well as a signature from a majority of the bridge committee members authorizing the upgrade. The function verifies the message signature, sets the `_upgradeAuthorized` state variable to `true`, and calls the standard `upgradeToAndCall` function (inherited from UUPSUpgradeable).

In turn, `upgradeToAndCall` invokes `_authorizeUpgrade`, which must be overridden by the child contracts to perform the required checks and revert if the upgrade should not be performed. In the case of CommitteeUpgradeable, the function simply asserts that `_upgradeAuthorized` is true. Then, `upgradeToAndCall` will update the address of the implementation contract and invoke it with the calldata provided in the signed message to perform any needed initialization. Finally, `_upgradeAuthorized` is reset back to `false`.

This flow exposes a potential reentrancy issue in the upgrade process. During the time the new implementation contracts are invoked by `upgradeToAndCall`, `_upgradeAuthorized` is `true`, and therefore any call to `upgradeToAndCall` (which is a public function) will succeed and allow to upgrade the contract to any arbitrary implementation address.

Impact

This issue could lead to a full contract takeover, giving it a critical severity. However, we believe it is unlikely for this vulnerability to be exploited in practice, since `upgradeWithSignatures` requires a signature from a majority of the bridge committee members.

While the code as reviewed did not make calls to untrusted external contracts in its initialization function, it is not possible to foresee whether such a call will be made in a future upgrade, perhaps due to the introduction of new features or support for additional assets.

Recommendations

The issue can be remediated in several ways. One of the least invasive possibilities would be to reset `_upgradeAuthorized` to `false` directly in `_authorizeUpgrade`. With this change, upgrades would not be authorized when `upgradeToAndCall` invokes the new implementation contract.

Note that this assumes (as is the case in the current implementation) that UUPSUpgradeable only calls `_upgradeAuthorized` once.

Remediation

This issue has been acknowledged by Mysten Labs, and a fix was implemented in commit `1c6e9356` ↗.

3.2. Rate limiter could semipermanently lock big transfers

Target	Bridge Design		
Category	Protocol Risks	**Severity**	Medium
Likelihood	Low	**Impact**	Low

Description

The bridge implements a rate limiter, which prevents larger-than-intended outflows. This functionality is intended to limit the maximum impact of a potential bridge compromise, as it caps the maximum loss to a set limit within a certain 24-hour window, giving time to the bridge committee to pause the bridge and take corrective action.

The rate limit is enforced on the destination chain, and there is no functionality that prevents a transfer larger than the configured rate limit from occurring.

Impact

A user could initiate a cross-chain transfer of an amount larger than the configured rate limit. The funds would then be stuck, as the rate limit would always be lower than the amount the user is entitled on the destination chain.

The funds are only semipermanently stuck, as the bridge committee could intervene to raise the rate limit, allowing the transfer to be claimed.

Recommendations

Consider enforcing a hard limit to the transferred amount on the source chain. The limit would have to be smaller than or at most as big as the limit enforced on the destination chain. Since the rate limits on the various chains are independent from each other, this would require coordination between the settings on all the chains supported by the bridge.

We could not verify whether the official bridge UI prevented users from initiating a transfer larger than the rate limit. We recommend implementing this feature if it was not already implemented.

Remediation

Mysten Labs acknowledged the potential issue and opted to not make changes to address it, for the following reasons:

- the rate limits are intended to be set to a relatively large value (in the order of millions of USD per day), and it is unlikely users will perform such a large cross-chain transfer
- users would likely use the frontend which will prevent making transfers which are too large
- rate limits are not symmetrical, which complicates preventing oversized transfers from being initiated, as the sending side would need to be kept up-to-date about the receiving side rate limit
- in a worst case scenario, the bridge committee can vote to raise the rate limit to make a large transfer succeed

3.3. Potentially unsafe ERC-20 usage

Target	Bridge Solidity contracts		
Category	Code Maturity	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

The Solidity bridge contracts directly perform several ERC-20 calls, including `transfer` and `transferFrom`. This poses a risk if the bridge were to be used with assets that are not strictly ERC-20 compliant.

Impact

The assets currently supported by the bridge do not arise any exploitable issue. However, it is not possible to predict which assets will be supported in the future by the bridge. There exist several assets that do not strictly implement the ERC-20 standard. For instance, some assets do not return a boolean value on `transfer`.

Recommendations

We strongly suggest to use a library that transparently handles ERC-20 pitfalls, such as OpenZeppelin SafeERC20.

Remediation

This issue has been acknowledged by Mysten Labs, and a fix was implemented in commit `7b273baa` ↗.

3.4. Missing committee-size limit

Target	BridgeCommittee		
Category	Code Maturity	**Severity**	Informational
Likelihood	Low	**Impact**	Informational

Description

The `BridgeCommittee::initialize` function does not enforce a limit on the size of the committee. However, multiple functions assume that the committee has at most 256 members.

Impact

This issue could cause index aliasing into the `committeeIndex` array, causing signatures from different committee members to be treated as coming from the same member in `verifySignatures`.

We believe this issue is unlikely to manifest, which lowered the severity and impact.

Recommendations

Consider either enforcing a limit on the committee size or changing the code to support more than 256 committee members.

Remediation

This issue has been acknowledged by Mysten Labs, and a fix was implemented in commit `015f8d8b` ↗.

3.5. Weak `claim_token` access control

Target	bridge.move		
Category	Coding Mistakes	**Severity**	High
Likelihood	Low	**Impact**	Medium

Description

The `claim_token` function is called to finalize a bridge transfer, returning `Coin`s that represent the bridged assets.

To prevent unauthorized callers from claiming transfers on someone else's behalf, `claim_token` inspects the `TxContext` to ensure that the recipient of the transfer being claimed is `TxContext.signer`.

However, this field does not contain the address of the caller but rather the address of the user that has submitted the transaction from which the call originated.

Impact

If the user calls a malicious contract while they have a pending transfer ready to be claimed, the malicious contract can claim the transfer on behalf of the user.

Recommendations

Consider removing the `claim_token` function or implementing a stricter, more explicit access-control mechanism.

Remediation

Mysten Labs has acknowledged the potential issue and opted to not make changes, in order to preserve the ability to more easily compose the native bridge with other DeFi modules. Additionally, Mysten Labs argued that since exploiting this vulnerability requires a user to interact with a malicious contract, mitigating this issue by removing `claim_token` would not fully remove social engineering issues, where a user could still be tricked into signing a more complex programmable transaction that claims tokens and transfers them to an unintended destination.

4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1. Bridge-design maturity

This section documents features that are already planned or in our opinion likely to be implemented in future versions. The purpose of this is to discuss security-relevant aspects of these features.

Incentive system

The bridge currently lacks an incentive system that encourages honest and timely behavior from the committee members and discourages dishonest behavior.

Bridge committee members do not currently own fees when a transfer occurs, nor are they subject to lose any assets if they do not keep a good uptime or if they provide a signature for a malicious bridge message.

It is not clear why, at the moment, bridge committee members would spend resources to operate a honest bridge node. Therefore, designing and implementing a both positive and negative incentives system appears to be one possible solution, adopted by other established cross-chain bridges as well. We note that designing and implementing the incentives is not trivial, and we recommend a security review of the design and implementation if this feature was to be added.

Committee management

Currently, the bridge EVM contracts do not fully support bridge committee membership rotation. The bridge committee members are set at the time the bridge contracts are created. Committee members can be blocklisted (or reallowed), but there is no governance action to permanently remove a member from the committee, nor is there an action that allows to add a new member to the committee. There is also no action that allows to change the weight of a committee member.

It is our understanding that the development team intends to tie bridge committee membership to Sui validator-set membership. As this is not implemented in the version of the code under review, we did not have the possibility to evaluate the design or the implementation of this future feature. This feature is clearly critical to the security and reliability of the bridge, and we recommend an independent review before mainnet launch.

Governance actions' practicality

The current implementation of the bridge server requires a restart after a manual configuration change to explicitly authorize each individual governance action, such as updating an asset price or pausing/unpausing the bridge. This is not a fundamental limitation and does not technically represent a security issue.

Asset prices' oracle

The current version of the bridge uses semistatic prices to enforce rate limits expressed in dollar amounts. Asset prices are updated by invoking the bridge contracts with a message signed by a majority of the committee members' weight.

The development team plans to integrate an oracle in a future version of the code. While not immediately critical to the security of the bridge, this aspect is an important mitigation that can lower the impact of a possible exploitation of the bridge contracts. Additionally, integration with an external oracle could have an impact on the availability of the bridge, as the external oracle contracts and/or data would be needed for the bridge to operate.

Coin object chosen at startup

The `Coin` object used to pay for bridge calls is chosen at the time the bridge node is started, and it never changes. There is no way to switch to a different `Coin` without restarting the brige node.

No support for bridging Sui-native assets

The current version of the bridge does not support bridging of Sui-native assets. Initially, assets can only be transferred from Ethereum to Sui, locking the original assets into the Ethereum contract and minting an equivalent amount of coins on Sui. The coins can then be bridged back from Sui to Ethereum, burning the Sui coins and releasing the locked original assets on Ethereum.

For this reason, and unlike many other bridges, the Ethereum contracts do not support minting. However, it is our understanding that a future version of the bridge will support bridging Sui to other chains. This will require an update to the Ethereum contracts, impacting critical parts of the code. We recommend to perform an independent security review of these changes.

No support for unusual tokens

The bridge contracts do not explicitly support unconventional tokens such as ERC-777, rebasing tokens, fee-on-transfer tokens, or tokens with multiple entry points. The development team does not currently plan on supporting arbitrary ERC-20 tokens. For this reason, we limited our threat model to the tokens that are explicitly supported at the specific version of the code under review.

A more extensive collection of tokens with unusual behavior can be seen at github.com/d-xo/weird-erc20 ↗.

4.2. Deployment-procedure improvement suggestion

The Solidity deployment scripts deploy BridgeVault and BridgeLimiter instances and explicitly transfer ownership of these contracts to the SuiBridge contract. It is our understanding that only the SuiBridge contract is supposed to have ownership of the vault and limiter contracts. We suggest to consider deploying these two contracts in the SuiBridge constructor or initializer. This would ensure ownership is necessarily assigned to SuiBridge and prevent any possible deployment mistake that could assign ownership incorrectly to an incorrect or additional address.

4.3. Minor issues

The following minor issues do not constitute a security concern; we provide this list as a recommendation to increase code quality.

Incorrect comment

The comment on top of `get_current_seq_num_and_increment` is incorrect. The function does not verify the sequence number; it only increments the stored one and returns current one.

In the version of the code we reviewed, callers of `get_current_seq_num_and_increment` are correctly checking the returned nonce to match the provided one.

Abuse of HTTP GET method

The bridge server only provides HTTP GET endpoints. Parameters are passed as part of the URL path and not as query parameters. This is uncommon and error-prone and could increase the risk of some vulnerabilities, like URL path traversal/manipulation. We suggest to consider implementing HTTP POST-based endpoints.

Unused constant

The `BRIDGE_AUTHORITY_TOTAL_VOTING_POWER` is unused.

Sui treasury `add_new_token` sanity checks

The current version of the code does not support Sui-native assets. The `add_new_token` function does nothing if the `native_token` boolean parameter is `true`. We suggest to raise an error if `native_token == true`.

Confusing `blocklist_type` field name

The `message.move`'s `Blocklist` struct has a `blocklist_type` field, which is confusingly named.

```
public struct Blocklist has drop {
    blocklist_type: u8,
    validator_eth_addresses: vector<vector<u8>>
}
```

At the moment, this field is used as a boolean that indicates whether a validator is blocked or not. We suggest to change the field to a boolean and the name to `blocklisted`.

4.4. Code update review

This section discusses the changes between branch `bridge-z` (at commit `8f29360f304f10d283f7c5cca7501804c2efed49`) and branch `feature/native-bridge` (at commit `8e7d74545946d9c7bc711d8bac864d87755a04ec`). The commits were obtained via the command

```
git rev-list --reverse bridge-z..origin/feature/native-bridge
```

Commit `de982e095248a72dfdc3d85f9f0ad65221303bfb`

This commit makes the following changes to `bridge.move::approve_bridge_message`:

- renames `approve_bridge_message` into `approve_token_transfer`
- addresses a `FIXME` reminding the development team to require the bridge to not be paused inf `approve_token_transfer`
- moves message signature verification earlier in the function
- explicitly requires messages passes to `approve_token_transfer` to have type `TOKEN` (other types of messages would not have been actionable, the check adds additional guarantees and an explicit error)
- explicitly requires the source or target chain of the message to match the chain ID where the module is running

Additionally, the `bridge_records` field of the `BridgeInner` struct was renamed to `token_transfer_records`.

Commit `564b1a72d3a1d5d5965739e894aeaa3ab06a9f47`

This commit addresses an issue in `committee.move::execute_blocklist`, where while processing a blocklist, an index used to address the committee member list was not reset. This would have caused some entries in the blocklist to be ineffective, preventing some members form being blocked or unblocked.

The bug was not detected as it only occurs if the order of the committee member addresses given in the blocklist does not match the order in which the members appear in the list as maintained by the committee module. Tests were modified accordingly so that a blocklist given in inverse order is also tested.

Commit `f75c194c67a52fd86e7ebb5f3af198851206fb5a`

`AddTokensOnSui` action

This commit added support for the `AddTokensOnSui` action, requiring changes to multiple components.

- the bridge client was changed to support generating corresponding HTTP requests to bridge servers
- the bridge server was changes to add support for the `AddTokensOnSui` governance action, adding a corresponding endpoint
- additional internal rust modules (e.g. `sui_transaction_builder.rs`, `types.rs`) were modified to support the new action

The action is handled as other governance actions, and as such requires an explicit configuration change from bridge committee node operators to allow their nodes to sign a specific instance of the action. The actions requires a threshold of >50% for the action to be approved.

The on-chain Move modules were not substantially changed, as they already had support for the `AddTokensOnSui` action.

Test harness changes

The commit added tests for the `AddTokensOnSui` action, and also made some unrelated changes to the testsuite to increase efficiency, mainly by changing how the test harness sets up the state of the bridge.

Commit `53b1f538fcefff34ae327b3619dce27ceb3bc1cf`

This commit added a CI workflow (Github action) that runs the bridge EVM contracts solidity tests. The action runs when certain events occur, including a push to the `main` branche, but also when a pull request is opened.

The action configuration passes the `INFURA_API_KEY` secret to the step running the tests. A malicious pull request could add a test which leaks the secret.

Commit `e81a5615038fd30917a2e4899b590d8141fbb15e`

This bumps the version of the `h2` create (an HTTP2 library) from `0.3.24` to `0.3.26`, which contains a security fix for a denial of service issue.

The change was required by CI rules, and we could not confirm whether Mysten was impacted by the issue.

More info about the issue can be found at https://seanmonstar.com/blog/hyper-http2-continuation-flood/.

Commit `c315e26e7f6cf1610f6b6d39557e56a135683dbd`

This commit adds unit tests for several Move modules, including:

- tests for validating chain IDs and chain route pairs
- additional tests for the initialization functionality of the committee module,
- testing that the limiter module updates route limits correctly

Commit `dacbfcf0c4f057731d611b30e032cdb6faea4c02`

This commit adds support for the `AddTokensOnEvm` action. The commit required changes similar to the ones made in commit `f75c194c67a52fd86e7ebb5f3af198851206fb5a`:

- the bridge client was changed to support generating corresponding HTTP requests to bridge servers
- the bridge server was changes to add an endpoint for the `AddTokensOnEvm` governance action
- additional internal rust modules (e.g. `eth_transaction_builder.rs`, `types.rs`) were modified to support the new action

Continuing the similarities with `f75c194c67a52fd86e7ebb5f3af198851206fb5a`, the on-chain Solidity contracts were not changed at all, as they already implemented support for this governance action.

As with other governance actions, bridge committee node operators need to perform an explicit configuration change to authorize their node to sign a specific instance of the action. In order to be ratified, the action needs aggregate signatures with >50% of the bridge committee total weight.

Commit `9002ca6303f8442b2d7f1dbba001502c260f9d31`

This commit changes the test harness to use real mainnet asset addresses for the `AddTokensOnEvm` action.

Commit `d48a7bd74c09e2fd03e8cff29ad8a39c3fe0853d`

This commit fixes a build issue, and does not contain any security-relevant change.

Commit `9acb3721f1a2e47cfcc725680d0e59b58033c33e`

This commit introduced a new `get_token_transfer_action_signatures` function in `bridge.move`, which allows to get the on-chain recorded signatures for token transfer messages more efficiently. This change only impacts tests.

As a minor observation, we note that the `self: &mut Bridge` parameter is a mutable reference, but does not need to be mutable. We suggest to change it to an immutable reference.

Commit `906aa1d882b83b761218c8642899295db2bd0f02`

This commit relaxes two tests, which do not appear to be related to the bridge components reviewed in this report. It also bumps the Move toolchain version in one specific test (`typescript/test/e2e/data/dynamic_fields`). Both changes are not security-relevant.

Commit `801d52b909e9733a1a265e06e80b47a605c5faba`

This commit only updates a comment on `BridgeCommittee::initialize` to better reflect the logic of the contract.

Commit `faad7963502fc15aea02181d34b4cb67a7fad270`

The commit fixes a potential reentrancy issue in the Solidity contracts. Refer to finding 3.1. ↗ for a more in-depth discussion.

Commit `194e02b3fb248dc95d643c373c437112f6da3d08`

This commit switches to using OpenZeppelin SafeERC20 library instead of performing direct Solidity calls when operating on ERC20 assets. See finding 3.3. ↗ for a more in-depth discussion.

Commit `d86b9f7ac25aa9bc4deaa2602cfb751130ccdc3a`

This commit adds a check to `message.move::create_token_bridge_message` which constrains the message payload length to 64 bytes.

Commit `fa2f690a302501cbc5caff94adcac8a738eaf35f`

This commit implements three tests which were not yet implemented (marked as TODO), where a user initiates bridging of, respectively, USDC, USDT and wBTC.

Commit `8127794079963267fdb4fc02dd5e5744b8cc4d78`

This commit changes the internal decimal precision used by the bridge limiter functionality from 4 to 8 decimals. The amount of decimals used is just a convention, and as such the actual logic of the contract is unaffected and unchanged. The commit updated comments, deployment configurations and tests to reflect the new convention.

Commit `4efc7a4b58a92be1805d24748f4eaf49c6c60876`

This commit eliminates the distinction between chain IDs used for Sui devnet and local devnet. To have a uniform naming, now networks used for testing for both Sui and Ethereum are referred to as "custom" networks, internally represented with chain ID 2 (for Sui) and 12 (for Ethereum). The change is relatively trivial, and the most security critical check affected is the validation of the route for a transfer, which was correctly updated.

Commit `e587b7c4d9a818ad21b1c3b63f8aac292a6acab1`

This commit changes the bridge client to implement caching of the bridge shared object, which increases efficiency since the information about the bridge object (specifically, the initial version when the object was shared) can be queried only once when the bridge client is started.

Commit `8538e59630b7f802a95a566b9232e6c9984b0acd`

This commit only removes some outdated and now incorrect comments.

Commit `015f8d8bb89df32a40651458bd1fa6f06d83d553`

This commit addresses issue 3.4. ↗, by requiring the committee length to be at most 255 in `BridgeCommittee::initialize`, and also adds a unit test to detect possible regressions.

Commit `c056094ef23878d746a18203a98bfdaf595e78c5`

This commit heavily refactors the structures used to represent the bridge node configuration. It introduces separate data structures for Ethereum-related, Sui-related, and general settings.

It also addresses several TODO notes, adding sanity checks on the bridge configuration, including validation of chain IDs and routes to ensure a misconfigured bridge node would not sign messages for an unintended destination.

Commit `3d3112f55c5728f19855d7d6a7522ef7b406a5ac`

This commit contains a reformatting and general cleanup of the bridge Move contracts. The cleanup includes adoping some of the newly introduced Move 2024 features, including the syntactic sugar allowing to call a "method" using the dot syntax (e.g. switching from `vec_map::is_empty(committee::committee_members(&inner.committee))` to `inner.committee.committee_members().is_empty()`).

Several tests were moved from `bridge.move` into a separate file.

Commit `61e8c30b2c907914d99fd391bd644bcd1ea7b42d`

This commit refactors bridge tests, mostly to avoid repeated similar code blocks.

Commit `bab85328364885fffab88025299314025d1ec66e`

This commit changes the Solidity bridge and vault contracts, so that the vault is now responsible for wrapping ETH into wETH. This removes the need for the bridge contract to know the address of wETH, as now it only works with native ETH instead.

Commit `e887a8e36f0f02355f82e34846d45232febe9241`

This commit updates the configuration of cargo-deny to allow issue `RUSTSEC-2024-0336`, which affects rustls and could be abused by a malicious TLS client to cause a DoS condition. To our knowledge, this issue does not affect bridge nodes.

The issue advisory is available at https://github.com/rustls/rustls/security/advisories/GHSA-6g7w-8wpp-frhj.

Commit `0c85b040ab3c7f42c6598a117702f15f37f2e836`

This commit updates the EVM test runner to fix path clashes that would occur when running multiple tests concurrently.

Commit `9152fae8fe0e09cac9a5524a056525b296c290d7`

This commit adds a new `eth_contracts_start_block_fallback` key to the bridge node configuration. This setting complements the existing `eth_contracts_start_block_override`. The two settings differ in that the "override" setting always applies, even if a block number can be retrieved from storage. This means that if set, the "override" setting will cause all events (starting from the specified block) to be processed when the node starts.

The "fallback" setting does not take precedence, and will only be used if a starting block number cannot be retrieved from storage.

Commit `0a2ed18c139001d22d23651442041d546a22a741`

This commit fixes the name of the test module contained in `bridge_tests.move`, which was incorrectly named `sui` instead of `bridge`. It also modifies the rust unit tests harness to allow running the Move tests with `cargo test` (including in the CI).

Commit `9f4e635fdbd3c3569c889c0ad852eaa6fd09a023`

This commit refactors `bridge.move` functions related to the final step of bridging a token (claiming), making them clearer. It also emits an event if the claim is rejected due to the rate limit being triggered.

Commit `fa91a50653e836f95b3815ff962389b74eaf1be3`

This commit increases the performance of the testsuite, mainly by performing setup operations in parallel. This is achieved by introducing a new `BridgeTestCluster` struct which is responsible for storing the state of the "virtual" cluster of bridge committee nodes involved in the test, and a corresponding `BridgeTestClusterBuilder` which is responsible of initializing `BridgeTestCluster`, performing setup actions in parallel (using async tokio tasks) where possible.

Commit `aff1e6b0e76b694b276ca9290d60c971d2a985b8`

This commit adds support for several Move events to the bridge rust crates (specifically, it adds support for `TokenTransferApproved`, `TokenTransferClaimed`, `TokenTransferAlreadyApproved` and

`TokenTransferAlreadyClaimed`).

It also adds an explicit check in the bridge `action_executor.rs`, ensuring that non-reverted transactions also emit a `TokenTransferClaimed` or `TokenTransferAlreadyClaimed` event, as an additional guarantee that the Move bridge module processed the transfer correctly.

Commit `8e7d74545946d9c7bc711d8bac864d87755a04ec`

This commit renames `bridge.move::TokenBridgeEvent` to `TokenDepositedEvent`, and removes the superfluous `message_type` field. The change also required changing the rust codebase accordingly.

5. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

5.1. Intro

The bridge is implemented by several EVM contracts.

SuiBridge

This is the main contract, exposing the core functionality that allows to initiate an outgoing asset transfer and to finalize an incoming asset transfer.

It inherits from CommitteeUpgradeable and is (according to the deploy scripts as reviewed) the exclusive owner of instances of a BridgeVault and BridgeLimiter instance.

BridgeVault

This contract has the sole purpose of keeping custody ERC-20 assets on behalf of the bridge. Its functions are external but guarded by `onlyOwner` modifiers, which allows only the bridge contract to call them. The functionality of the contract is trivial and not directly reachable. For this reason, we do not provide an explicit threat modeling for BridgeVault.

BridgeLimiter

This contract keeps track of the amount of assets flowing out of the bridge contract and provides public functions that can be used to check and enforce the rate limit. It also provides functions that allow the SuiBrige contract to submit an outgoing transfer for record keeping and a public function that can be used to update the rate-limit thresholds (requiring a set of signatures from bridge committee members with enough weight).

BridgeConfig

This contract stores several bridge-configuration parameters:

- Current chain ID
- Mapping between supported ERC-20 token ID and their Ethereum addresses
- Prices used by the BridgeLimiter contract
- Mapping storing supported destination/source chains

All the configuration parameters can be updated by providing a message signed by a subset of the bridge committee members with sufficient (>50%) total weight.

CommitteeUpgradeable

This abstract contract implements upgradeability and is used via inheritance by SuiBridge, BridgeCommittee, BridgeConfig, and BridgeLimiter.

Upgrading requires submitting a message signed by a subset of the bridge committee members with sufficient (>50%) total weight.

BridgeCommittee

This contract keeps track of the set of committee members and provides functions to verify a message is signed properly by a big enough set of the committee (depending on the message type).

It also provides a function that allows to put on and remove from the blocklist one or more members, provided a message signed by a set of committee members with sufficient weight (>50%).

5.2. Contract: SuiBridge

Function: `initialize(address _committee, address _, address _limiter, address _wETH)`

This function can be used only once to initialize the contract.

Inputs

- `_committee`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Address of the bridge committee contract.
- `_vault`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Address of the bridge vault contract.
- `_limiter`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Address of the bridge limiter contract.
- `_wETH`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Address of the WETH contract.

Branches and code coverage (including function calls)

Intended branches

- ☑ Initializes the bridge.

Negative behavior

- ☐ Reverts if the contract is already initialized.

Function: `transferBridgedTokensWithSignatures(bytes[] memory signatures, BridgeUtils.Message memory message)`

This function allows the caller to finish a token transfer by providing a bridge message and enough signatures from bridge committee members.

Inputs

- `signatures`
  - **Control**: Arbitrary.
  - **Constraints**: Must be valid aggregate signatures for `message`.
  - **Impact**: Signatures authorizing the `message`.
- `message`
  - **Control**: Arbitrary.
  - **Constraints**:
    - `messageType` must be `TOKEN_TRANSFER`.
    - Source `chainID` must be allowed.
    - Must not be a replay (`nonce` not already processed).
    - Payload must be a valid `TokenTransferPayload` — destination `chainID` must match, `tokenID` must be supported, and source and destination addresses' length must be valid.
  - **Impact**: Message describing details of the transfer (e.g., asset, amount, source, destination).

Branches and code coverage (including function calls)

Intended branches

- ☑ Validates incoming message signature and payload, transfers tokens from the vault, and marks the message nonce as processed.

Negative behavior

- ☑ Reverts if the signatures are invalid (insufficient signatures' weight).
- ☑ Reverts if the signatures are invalid (signature from an unknown key) (tested indirectly).
- ☑ Reverts if the signatures are invalid (repeated signature) (tested indirectly).
- ☑ Reverts if the signatures are invalid (blocklisted signer) (tested indirectly).
- ☑ Reverts if the nonce was already processed.
- ☑ Reverts if the message type is invalid.

- [x] Reverts if the message source chain is invalid.
- [ ] Reverts if the message payload is invalid (wrong destination `chainID`).
- [ ] Reverts if the message payload is invalid (wrong `tokenID`).
- [ ] Reverts if the message payload is invalid (wrong source-address length).
- [ ] Reverts if the message payload is invalid (wrong destination-address length).
- [x] Reverts if the rate limit was reached.
- [ ] Reverts if the contract is paused.

Function: `executeEmergencyOpWithSignatures(bytes[] memory signatures, BridgeUtils.Message memory message)`

This function allows the caller to pause or unpause the bridge contract, requiring an aggregate signature with enough weight.

Inputs

- `signatures`
  - **Control**: Arbitrary.
  - **Constraints**: Must be valid aggregate signatures for `message`.
  - **Impact**: Signatures authorizing the `message`.
- `message`
  - **Control**: Arbitrary.
  - **Constraints**:
    - `messageType` must be `EMERGENCY_OP`.
    - `nonce` must match expected nonce.
    - Destination `chainID` must match current chain.
    - Payload must be a valid `EmergencyOpPayload`.
  - **Impact**: Message specifying whether the bridge should be paused or unpaused.

Branches and code coverage (including function calls)

Intended branches

- [x] Validates incoming message signature and payload and pauses/unpauses the contract.

Negative behavior

- [x] Reverts if the signatures are invalid (insufficient signatures weight).
- [x] Reverts if the signatures are invalid (signature from an unknown key) (tested indirectly).
- [x] Reverts if the signatures are invalid (repeated signature) (tested indirectly).
- [x] Reverts if the signatures are invalid (blocklisted signer) (tested indirectly).

- ☑ Reverts if the nonce is invalid.
- ☑ Reverts if the message type is invalid.
- ☐ Reverts if the message payload is invalid (wrong destination `chainID`).
- ☐ Reverts if the message payload is invalid (incorrect length).
- ☑ Reverts if the message payload is invalid (incorrect opcode).

Function: `bridgeERC20(uint8 tokenID, uint256 amount, bytes memory recipientAddress, uint8 destinationChainID)`

This function allows the caller to initiate an ERC-20 token transfer.

Inputs

- `tokenID`
    - **Control**: Arbitrary.
    - **Constraints**: Must correspond to a supported asset.
    - **Impact**: Determines the asset to bridge.
- `amount`
    - **Control**: Arbitrary.
    - **Constraints**: User must own at least this amount, and bridge must have sufficient allowance.
    - **Impact**: Amount to bridge.
- `recipientAddress`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Address of the recipient on destination chain.
- `destinationChainID`
    - **Control**: Arbitrary.
    - **Constraints**: Must be a valid destination `chainID`.
    - **Impact**: ID of the destination chain.

Branches and code coverage (including function calls)

Intended branches

- ☑ Validates arguments, transfers tokens from the user to the bridge vault, emits deposit event, and increases the stored nonce for token transfers.

Negative behavior

- ☑ Reverts if the `tokenID` is invalid.
- ☑ Reverts if the bridge has insufficient allowance.

- ☐ Reverts if the contract is paused.
- ☐ Reverts if the destination chain is invalid.

Function: `bridgeETH(bytes memory recipientAddress, uint8 destinationChainID)`

This function allows the caller to initiate a native ETH transfer.

Inputs

- `recipientAddress`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the recipient on destination chain.
- `destinationChainID`
  - **Control**: Arbitrary.
  - **Constraints**: Must be a valid destination `chainID`.
  - **Impact**: ID of the destination chain.

Branches and code coverage (including function calls)

Intended branches

- ☑ Validates arguments, wraps ETH to WETH and transfers them to the bridge vault, emits deposit event, and increases the stored nonce for token transfers.

Negative behavior

- ☐ Reverts if the contract is paused.
- ☐ Reverts if the destination chain is invalid.

5.3. Introduction

The Sui portion of the group consists of eight modules.

bridge

This is the entry point to all bridge functionality. It keeps tracks of all messages received, the nonces for respective chains, and the committee of voters on the bridge, and it manages access to the treasury and rate limiter. The bridge module has functionality to do the following:

- Post a new message that is either a token transfer or a system-management message
- Start a bridge transaction, sending tokens to another chain

- Approve a posted message by submitting committee signatures
- Claim tokens either directly to the caller or to the destination account of the cross-chain transfer
- Execute the various system messages
- Add more tokens to be supported by the bridge

chain_ids

A module used for storing the hardcoded chain IDs of some supported chains as well as routes. It supports some simple queries on this information, such as querying the list of valid routes if a particular route is valid and if a chain ID is part of the hardcoded supported chain IDs.

committee

This is the committee manager, which handles tracking the members of the committee and the blocklist. This module supports the following:

- Updating the blocklist
- Adding a registration request to the list
- Creating a new committee based on the registrations available at the time
- Checking the signatures to see if enough voting power has signed the request

crypto

This is a module containing a single utility function that converts an ECDSA public key to an Ethereum address.

limiter

This module tracks and helps enforce a rate limit on value outflows from the bridge. For each route, the module keeps a ring buffer of value outflows. The buffer tracks the amount of value outflowed for every hour for the past 24 hours. Records that become too old are dequeued from the front of the ring buffer. The ring buffer is used to maintain a sum of the amount of value leaving the bridge on a certain route within the last day. This sum is finally used to check if a given transaction will exceed the rate limit. The limits and asset prices are configurable through a dedicated function.

message

This module contains the serialization and deserialization for bridge messages. The module also contains hardcoded constants for distinguishing a pause and an unpause message as well as a function hardcoding the voting power required to execute a given type of message.

message_types

This short module contains hardcoded message type codes, and getters for them.

treasury

The treasury module keeps tracks of treasuries, one for each coin, as well as the approximate price for the purposes of the rate limiter. The treasury keeps track of all the supported tokens, and supports minting, burning, updating the price of tokens, and adding support for new tokens. Note that it is not possible to remove support for a token.

5.4. The Security Model

The available methods to interact with the bridge are the following:

```
public fun committee_registration(
    self: &mut Bridge,
    system_state: &mut SuiSystemState,
    bridge_pubkey_bytes: vector<u8>,
    http_rest_url: vector<u8>,
    ctx: &TxContext
)
public fun register_foreign_token<T>(self: &mut Bridge, tc: TreasuryCap<T>,
    uc: UpgradeCap, metadata: &CoinMetadata<T>)
public fun send_token<T>(
    self: &mut Bridge,
    target_chain: u8,
    target_address: vector<u8>,
    token: Coin<T>,
    ctx: &mut TxContext
)
public fun approve_bridge_message(
    self: &mut Bridge,
    message: BridgeMessage,
    signatures: vector<vector<u8>>,
)
public fun claim_token<T>(self: &mut Bridge, clock: &Clock, source_chain: u8,
    bridge_seq_num: u64, ctx: &mut TxContext): Coin<T>
public fun claim_and_transfer_token<T>(
    self: &mut Bridge,
    clock: &Clock,
    source_chain: u8,
    bridge_seq_num: u64,
    ctx: &mut TxContext
)
public fun execute_system_message(
    self: &mut Bridge,
    message: BridgeMessage,
    signatures: vector<vector<u8>>,
)
```

Other methods are available on the main bridge module and other modules, but they are read-only or test-only.

The committee registration can be called by anyone with any set of arguments, but the target validator is the sender of the transaction. This cannot be forged.

Registering new tokens for the bridge requires that the caller hav the `UpgradeCap` for the bridge.

For sending tokens, the target chain is checked to be in a whitelist of supported chains. If the coin passed in is not supported, it will fail to be burned. More importantly, the coin is passed in by the caller. The target address can't be checked here, but since any coins that are lost due to an invalid target address belongs to the caller, it is the same as a burn.

In all branches of `approve_bridge_message`, there is a call that verifies the unmodified message data with the list of signatures passed in.

```
inner.committee.verify_signatures(message, signatures);
```

In particular, the signature checker verifies the following:

- Each signature is that of a committee member
- No signatures are that of the same member
- The voting power threshold for the message is reached

The token claim functions both check that:

- The rate limit has not been reached
- The sequence number for which the claim is being made has not already been claimed
- The claim is properly signed by the committee

The `claim_token` function function ensures that only the recipient can claim the coins by checking that the sender of the transaction is the recipient before returning the claimed coins. This is secure - but it can potentially be a problem if the user signs a malicious transaction. Refer to finding 3.5. ↗ for a more in depth discussion.

The `claim_and_transfer_token` function always transfers to the recipient on chain rather than the caller.

`execute_system_message` always checks signatures on the passed in message before executing it. Replay is not possible here, since the sequence number of the message is checked against the number on chain.

6. Assessment Results

At the time of our assessment, the reviewed code was neither deployed to the Ethereum Mainnet nor Sui.

During our assessment on the scoped SuiBridge V1 contracts, we discovered five findings. No critical issues were found. Two findings were of medium impact, two were of low impact, and the remaining finding was informational in nature.

6.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.