Veridise. Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

EVM Bridge

Veridise Inc.

September 7, 2023

► **Prepared For:**

Webb
https://webb.tools/

► **Prepared By:**

Jon Stephens
Shankara Pailoor
Ian Neal

► **Contact Us:** contact@veridise.com

► **Version History:**

Aug. 14, 2023 Initial Draft

© 2023 Veridise Inc. All Rights Reserved.

Contents

 © 2023 Veridise Inc.

1 Executive Summary

From Jul. 24, 2023 to Aug. 13, 2023, Webb engaged Veridise to review the security of their EVM Bridge. The review covered the the bridge's ZK circuits written in Circom and the on-chain contracts to interact with the bridge. Veridise conducted the assessment over 9 person-weeks, with 3 engineers reviewing code over 3 weeks at commit `848d073`. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers as well as extensive manual auditing. It should also be noted that all of the fixes were merged into the repository's main branch at commit `eeb4fc7`.

Description. The Webb EVM Bridge project implements a bridge that can be used to send funds across blockchains. To do so, each blockchain must have an anchor, token wrapper and bridge. The token wrapper essentially acts as a vault for supported funds. It allows users to wrap tokens for transfer and to later unwrap them on the destination chain to receive the underlying tokens back. The anchor then tracks the funds within the bridge by maintaining commitments of managed funds, which are cryptographic notes that indicates a user owns a certain amount of funds on a certain anchor. To do so, it maintains a merkle tree of the commitments users have created on the current chain and the roots of merkle trees on all adjacent chains. To prevent commitments from being spent multiple times, it also tracks the nullifiers of commitments that have been spent. Finally, the bridge manages the chain's infrastructure. It allows a committee to elect a governor who performs administrative tasks on the chain, such as updating the roots of anchors on adjacent chains.

To interact with the anchors and send funds between blockchains, users wrap their funds and then interact with Webb's ZK circuits which are forked from Tornado Nova. These circuits operate on external wrapped funds and internal commitments. They allow users spend commitments and external funds on the current chain to move funds to connected chains and withdraw funds to the current chain. Importantly, they also enforce that no funds are lost or created. Once a proof has been created, it can be provided to the anchor for execution.

Code assessment. The Webb developers provided the source code of the EVM Bridge contracts for review. To facilitate the Veridise auditors' understanding of the code, the EVM Bridge developers provided documentation on the overall architecture and design of the project. The source code also contained some documentation in the form of READMEs and in-line documentation on functions and storage variables. The source code contained a test suite, which the Veridise auditors noted cover key user-flows and edge cases.

Summary of issues detected. The audit uncovered 35 issues, 3 of which are assessed to be of high or critical severity by the Veridise auditors. Specifically, V-WBT-VUL-001 identified a function that could be used to steal funds, V-WBT-VUL-002 identified the potential for commitments to be overwritten and V-WBT-VUL-003 identified a replay attack that could allow someone to vote against a user's wishes. The Veridise auditors also identified several

© 2023 Veridise Inc.

medium-severity issues, including methods to bypass the bridge's rate limiting (V-WBT-VUL-008), methods to bypass the membership check of valid roots (V-WBT-VUL-006), and methods of manipulating the protocol's voting procedure (V-WBT-VUL-007, V-WBT-VUL-009).

Recommendations. After auditing the protocol, the auditors had a few suggestions to improve the EVM Bridge. During the audit, a significant amount of duplicated and dead code was identified. As an example, the `SetMembership` circuit provides a subset of the behavior in `ForceSetMembershipIfEnabled`, but `SetMembership` is not used at all. Similarly, several different contracts define similar functions such as `parseChainIdFromResourceId` which is not used in some of those contracts. We would recommend reducing the amount of duplication to improve the maintainability of the codebase.

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
EVM Bridge	`848d073`	Solidity and Circom	EVM-Compliant Chains

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Jul. 24 - Aug. 13, 2023	Manual & Tools	3	9 person-weeks

Table 2.3: Vulnerability Summary.

Name	Number	Resolved
Critical-Severity Issues	1	1
High-Severity Issues	2	1
Medium-Severity Issues	8	8
Low-Severity Issues	5	5
Warning-Severity Issues	14	14
Informational-Severity Issues	5	5
TOTAL	35	34

Table 2.4: Category Breakdown.

Name	Number
Data Validation	10
Logic Error	7
Usability Issue	4
Gas Optimization	3
Maintainability	3
Compatibility Issue	2
Replay Attack	1
Frontrunning	1
Locked Funds	1
Transaction Ordering	1
Denial of Service	1
Authorization	1

© 2023 Veridise Inc.

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of EVM Bridge's smart contracts and ZK circuits as outlined in the scope below. In our audit, we sought to answer the following questions:

- Can a user withdraw funds they do not own from the bridge?
- Can a user create or destroy funds?
- Is it possible for commitments to be lost or user funds to be locked in the bridge?
- Can commitments be spent multiple times?
- Can a user bypass the bridge's rate-limiting protections?
- Can a user interact with an unsupported chain?
- Is the bridge protected against token price fluctuations?
- Can a user cast a vote for another user without their permission?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved a combination of human experts and automated program analysis & testing tools. In particular, we conducted our audit with the aid of the following techniques:

- *Static analysis.* To identify potential common vulnerabilities, we leveraged our custom smart contract analysis tool Vanguard. This tool is designed to find instances of common smart contract and zero-knowledge circuit vulnerabilities, such as reentrancy, uninitialized variables and underconstrained signals.
- *Automated verification.* To verify safety properties of the zero-knowledge circuits, we leveraged our custom automated verification tool Picus. This tool is designed to prove or find violations of determinism, which is an important safety property for zero-knowledge circuits.

Scope. The scope of this audit is limited to the following directories in the repository located at `https://github.com/webb-tools/protocol-solidity`:

- `/circuits/merkle-tree` (excl. `batchMerkleTreeUpdate.circom`, `merkleForest.circom`)
- `/circuits/set`
- `/circuits/vanchor` (excl. `transactionForest.circom`)
- `/packages/contracts/contracts/handlers`
- `/packages/contracts/contracts/hashers`
- `/packages/contracts/contracts/interfaces`
- `/packages/contracts/contracts/libs`
- `/packages/contracts/contracts/structs`
- `/packages/contracts/contracts/handlers`

© 2023 Veridise Inc.

- `/packages/contracts/contracts/tokens` (excl. `AaveTokenWrapper.sol`)
- `/packages/contracts/contracts/trees` (excl. `LinkableIncrementalBinaryTree.sol`, `MerkleForest.sol`)
- `/packages/contracts/contracts/utils`
- `/packages/contracts/contracts/vanchors` (excl. `VAnchorForest.sol`)
- `/packages/contracts/contracts/verifiers`

Methodology. Veridise auditors inspected the provided tests, and read the EVM Bridge documentation. They then began a manual audit of the code assisted by both static analyzers and automated verification. During the audit, the Veridise auditors regularly met with the EVM Bridge developers to ask questions about the code and discuss newly discovered issues.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

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

In this case, we judge the likelihood of a vulnerability as follows in 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

In addition, we judge the impact of a vulnerability as follows in 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, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-WBT-VUL-001	Public Function Allows Theft from VAnchor	Critical	Fixed
V-WBT-VUL-002	InsertTwo Functions may Overwrite Leaves	High	Fixed
V-WBT-VUL-003	Replay Attack on Voting Signatures	High	Partially Fixed
V-WBT-VUL-004	Unchecked assumptions in hash11	Medium	Fixed
V-WBT-VUL-005	Missing SNARK_SCALAR_FIELD Checks	Medium	Fixed
V-WBT-VUL-006	Incorrect Initialization in Membership Circuits	Medium	Fixed
V-WBT-VUL-007	Batch Voting May Change Period Mid-Vote	Medium	Fixed
V-WBT-VUL-008	RateLimitedVAnchor Limit Bypass	Medium	Fixed
V-WBT-VUL-009	Voting Results May Change on Vote Order	Medium	Intended Behavior
V-WBT-VUL-010	Potential Problems with Transfer	Medium	Fixed
V-WBT-VUL-011	Token Wrapper Bundling	Medium	Acknowledged
V-WBT-VUL-012	Native Token Wrapping may Bypass Limit	Low	Fixed
V-WBT-VUL-013	Missing zeroLeaf Check	Low	Fixed
V-WBT-VUL-014	Multiple Voting Periods per Transaction	Low	Fixed
V-WBT-VUL-015	Voting Disabled by Default	Low	Intended Behavior
V-WBT-VUL-016	Validate Calldata Has Correct Size	Low	Fixed
V-WBT-VUL-017	Missing Address Checks in VAnchorVerifier	Warning	Fixed
V-WBT-VUL-018	Missing Address Checks in TxProofVerifier	Warning	Fixed
V-WBT-VUL-019	Invalid Argument Reverts with Invalid Proof	Warning	Fixed
V-WBT-VUL-020	Unchecked Bytes Length in decodeRoots	Warning	Fixed
V-WBT-VUL-021	Anyone May Deploy Contract via Deployer	Warning	Intended Behavior
V-WBT-VUL-022	External Functions Should Be Public	Warning	Fixed
V-WBT-VUL-023	Ambiguous Batch Voting Tallying Logic	Warning	Intended Behavior
V-WBT-VUL-024	Missing Address Check in LinkableAnchor	Warning	Fixed
V-WBT-VUL-025	nthZero May Return Invalid Leafs	Warning	Fixed
V-WBT-VUL-026	Create2 Compatibility Problems	Warning	Acknowledged
V-WBT-VUL-027	Hardcoded Address Compatibility Problems	Warning	Acknowledged
V-WBT-VUL-028	No Depth to Voter Merkle Tree	Warning	Fixed
V-WBT-VUL-029	Use Constant Instead of Magic Number	Warning	Fixed
V-WBT-VUL-030	Tree Height Not Validated	Warning	Fixed
V-WBT-VUL-031	Unnecessary Getter Function	Info	Fixed
V-WBT-VUL-032	Incorrect Documentation of RawMerkleTree	Info	Fixed
V-WBT-VUL-033	Deployed Circuits Are Labeled as Test	Info	Fixed
V-WBT-VUL-034	Missing Bounds on MAX_EXT_AMOUNT	Info	Fixed
V-WBT-VUL-035	Multiple Requires Can Be Simplified to One	Info	Fixed

© 2023 Veridise Inc.

4.1 Detailed Description of Issues

4.1.1 V-WBT-VUL-001: Public Function Allows Theft from VAnchor

Severity	Critical	**Commit**	848d073
Type	Logic Error	**Status**	Fixed
File(s)	`VAnchorBase.sol`		
Location(s)	`_withdrawAndUnwrap`		

The `VAnchorBase` abstract contract appears to provide common internal behaviors common to anchors. Two of the functions, `_withdrawAndUnwrap` and `_executeWrapping` are public but we believe were intended to be internal. In particular, the `_withdrawAndUnwrap` function shown below provides functionality to unwrap bridged tokens and then send the underlying tokens to the recipient.

```
function _withdrawAndUnwrap(address _fromTokenAddress, address _toTokenAddress,
    address _recipient, uint256 _minusExtAmount
) public payable {
    // We first withdraw the assets and send them to ‘this‘ contract address.
    // This ensure that when we unwrap the assets, ‘this‘ contract has the
    // assets to unwrap into.
    _processWithdraw(_fromTokenAddress, payable(address(this)), _minusExtAmount);

    ITokenWrapper(_fromTokenAddress).unwrapAndSendTo(
        _toTokenAddress, _minusExtAmount, _recipient);
}
```

Snippet 4.1: Public function that could allow funds to be stolen

Since the wrapped token may have been bridged to this chain, however, the user making the request may not possess the wrapped tokens on this chain. Instead, it is sufficient to prove that they deposited tokens on another chain so that they may be withdrawn on this chain. As such, the `_processWithdraw` function will mint new wrapped tokens (if necessary) for the user, assuming a deposit has been made.

```
function _processWithdraw(address _token, address _recipient, uint256 _minusExtAmount
) internal virtual {
    uint balance = IERC20(_token).balanceOf(address(this));
    if (balance >= _minusExtAmount) {
        // transfer tokens when balance exists
        IERC20(_token).safeTransfer(_recipient, _minusExtAmount);
    } else {
        // mint tokens when not enough balance exists
        IMintableERC20(_token).mint(_recipient, _minusExtAmount);
    }
}
```

Snippet 4.2: The definition of the `_processWithdraw` function

Impact If a user calls `_withdrawAndUnwrap` they do not need to deposit funds first. Instead, a malicious user can simply call `_withdrawAndUnwrap` to directly remove funds. As such, any user can easily drain the bridge of all deposited funds.

Recommendation Make both `_withdrawAndUnwrap` and `_executeWrapping` internal.

© 2023 Veridise Inc.

4.1.2 V-WBT-VUL-002: InsertTwo Functions may Overwrite Leaves

Severity	High	**Commit**	848d073
Type	Data Validation	**Status**	Fixed
File(s)	`MerkleTreeWithHistory.sol, LinkableIncrementalBinaryTree.sol`		
Location(s)	`_insertTwo,_insertTwo`		

It can be more efficient to insert multiple leaves into a tree rather than insert leaves one by one, as a single interior node of the tree could be modified to perform the insertion. As an optimization, the developers implemented an "insert two" operation that would insert two leaves into the tree. It is performed by modifying the the interior node directly above the first empty leaf in the tree as shown below.

```
// Disclaimer: using this function assumes both leaves are siblings.
function _insertTwo(uint256 _leaf1, uint256 _leaf2) internal override returns (uint32
     index) {
    uint32 _nextIndex = nextIndex;
    require(
        _nextIndex != uint32(2) ** levels,
        "Merkle tree is full. No more leaves can be added"
    );
    uint32 currentIndex = _nextIndex / 2;
    uint256 currentLevelHash = hashLeftRight(_leaf1, _leaf2);
    uint256 left;
    uint256 right;
    for (uint32 i = 1; i < levels; i++) {
        if (currentIndex % 2 == 0) {
            left = currentLevelHash;
            right = uint256(hasher.zeros(i));
            filledSubtrees[i] = currentLevelHash;
        } else {
            left = filledSubtrees[i];
            right = currentLevelHash;
        }
        currentLevelHash = hashLeftRight(left, right);
        currentIndex /= 2;
    }

    uint32 newRootIndex = (currentRootIndex + 1) % ROOT_HISTORY_SIZE;
    currentRootIndex = newRootIndex;
    nextIndex = _nextIndex + 2;
    roots[newRootIndex] = Root(currentLevelHash, nextIndex);
    return _nextIndex;
}
```

Snippet 4.3: Function used to insert two entries in the tree

Note that the `_insertTwo` function has a disclaimer stating that the function assumes both leaves are siblings. This, however, is not enforced by the function and does not appear to be checked by users of the function.

Impact If `_insertTwo` is called when the size of the tree is odd, the newest leaf in the tree will be overwritten due to this unchecked assumption.

Recommendation To prevent potential programming mistakes, check that `nextIndex` is even.

© 2023 Veridise Inc.

4.1.3 V-WBT-VUL-003: Replay Attack on Voting Signatures

Severity	High	**Commit**	848d073
Type	Replay Attack	**Status**	Partially Fixed
File(s)	`Governable.sol`		
Location(s)	`voteInFavorForceSetGovernorWithSig`		

The `Governable` contract allows votes to be cast via signature so that users don't have to directly interact with the contract itself. To do so, the user must sign the flattened data contained in the `Vote` struct, which is shown below.

```
struct Vote {
    uint32 leafIndex;
    address proposedGovernor;
    bytes32[] siblingPathNodes;
}
```

Snippet 4.4: Definition of the `Vote` struct that will be signed by users

It should be noted, however, a user can vote multiple times in the same voting period and it appears that often-times the merkle tree is not updated between voting periods. As signatures are not blacklisted when used, it is therefore the case that the same signature can be re-used by others within the same voting period and likely across voting periods as well.

```
function voteInFavorForceSetGovernorWithSig(
    Vote[] memory votes, bytes[] memory sigs
) external isValidTimeToVote {
    require(votes.length == sigs.length, "Governable: Invalid number of votes and
    signatures");
    for (uint i = 0; i < votes.length; i++) {
        ...
        // Recover the address from the signature
        address proposerAddress = recover(abi.encode(votes[i]), sigs[i]);
        // Check merkle proof is valid
        if (
            _isValidMerkleProof(votes[i].siblingPathNodes, proposerAddress, votes[i].
    leafIndex)
        ) {
            _processVote(votes[i], proposerAddress);
        }
    }
}
```

Snippet 4.5: Location where votes are processed by validating signatures

Impact Once a user votes with a signature, that signature will be exposed to other users. This can allow a malicious user to manipulate a vote, potential against a user's will, to elect the governor of their choice (so long as the user has voted for them in the past).

Recommendation Add a nonce to the vote struct and ensure that a nonce can only be used once.

Developer Response The Veridise engineers noticed that the replay attack was still possible within a voting period. We expect, however, that each user will cast 1 vote per voting period as users will be expected to cast their vote specifically on a governor that will preserve the liveness of the bridge.

© 2023 Veridise Inc.

4.1.4 V-WBT-VUL-004: Unchecked Assumptions in hash11

Severity	Medium	Commit	848d073
Type	Data Validation	Status	Fixed
File(s)	PoseidonHasher.sol		
Location(s)	hash11		

The `hash11` function accepts a dynamic `uint256[]` array. If the array is under 11 elements long, the function performs the hash as if the unprovided elements are zero. However, there is no check that the array is not greater than 11 elements long, leading to a possible array-out-of-bounds error when the `input11` copy is being filled (as `input11` is only 11 elements):

```
for (uint256 i = 0; i < array.length; i++) {
        input11[i] = array[i];
}
```

Snippet 4.6: If `array.length > 11`, an array-out-of-bounds error will occur

Impact This can cause unexpected reversions due to an out-of-bounds error. Additionally, the implicit zero-padding of short arrays may cause the computed hash to differ from what the user expects.

Recommendation If the function is not anticipated to be used (as it currently appears to be unreferenced), the function could be removed. Otherwise, the following fixes could be implemented:

- Add explicit checks to ensure the size of the input array is less than or equal to 11 elements.
- Change the input argument to a static array of size 11 elements (similar to the `hash1`, `hash3`, `hash4`, and `hash5` functions).

The function could also be changed to use array slices instead of copy subarrays for improved efficiency.

4.1.5 V-WBT-VUL-005: Missing Checks Against SNARK_SCALAR_FIELD

Severity	Medium	**Commit**	848d073
Type	Data Validation	**Status**	Fixed
File(s)	`PoseidonHasher.sol`		
Location(s)	`hash1`, `hash3`, `hash4`, `hash5`, `hash11`		

All hash functions in `PoseidonHasher` (with the exception of `hashLeftRight`) omit a check that `leaf < SNARK_SCALAR_FIELD`, so the provided inputs may be larger than the `SNARK_SCALAR_FIELD` constant.

Impact Out-of-range input values may result in erroneous hash computations.

Recommendation Add appropriate checks against `SNARK_SCALAR_FIELD` at all locations that compute a Poseidon hash. Alternatively, a wrapper around Poseidon hashing could be constructed to perform this check so that all locations that use Poseidon hashing do not have to include this check separately.

© 2023 Veridise Inc.

4.1.6 V-WBT-VUL-006: Incorrect Initialization in Membership Circuits

Severity	Medium	Commit	848d073
Type	Logic Error	Status	Fixed
File(s)	`membership.circom`, `membership_if_enabled.circom`		
Location(s)	`ForceSetMembershipIfEnabled`, `SetMembership`		

The templates `SetMembership` and `ForceSetMembershipIfEnabled` try to check if an element e is in a set S by generating a constraint of the form $\prod_{s \in S} s - e = 0$. They do does this by iterating over elements s of the set and building the product as shown below:

```
template SetMembership(length) {
  signal input element;
  signal input set[length];

  signal diffs[length];
  signal product[length + 1];
  product[0] <== element;

  for (var i = 0; i < length; i++) {
    diffs[i] <== set[i] - element;
    product[i + 1] <== product[i] * diffs[i];
  }

  product[length] === 0;
}
```

The issue is that $product[0]$ is initialized to $element$ which results in the constraint $element * \prod_i set[i] - element = 0$ which can be trivially satisfied when $element$ is 0.

Impact `ForceSetMembershipIfEnabled` is used in the circuit `ManyMerkleProof` to check whether a computed Merkle root belongs to a set of known Merkle roots. A malicious user that knows how to compute a Merkle root equal to 0 could pass along the hashes and generate an invalid proof.

Recommendation We recommend that $product[0]$ is initialized to 1.

4.1.7 V-WBT-VUL-007: Batch Voting May Change Voting Period in Middle of Vote

Severity	Medium	Commit	848d0
Type	Logic Error	Status	Fixed
File(s)	`Governable.sol`		
Location(s)	`voteInFavorForceSetGovernorWithSig`		

The external function `voteInFavorForceSetGovernorWithSig` tallies votes for ownership of the contract within a voting period. It takes as input a list of votes, tallies the vote, and elects a new governor if the number of votes for a candidate is above simple majority. However, the logic of the implementation is incorrect and can result in a change in voting period in the middle of the vote. As a result, votes in one period could count for the next period.

To see why, here is the implementation of `voteInFavorForceSetGovernorWithSig`:

```
require(votes.length == sigs.length, "Governable: Invalid number of votes
    and signatures");
        for (uint i = 0; i < votes.length; i++) {
            require(
                votes[i].proposedGovernor != address(0x0),
                "Governable: Proposed governor cannot be the zero address"
            );
            // Recover the address from the signature
            address proposerAddress = recover(abi.encode(votes[i]), sigs[i
    ]);

            // Check merkle proof is valid
            if (
                _isValidMerkleProof(votes[i].siblingPathNodes,
    proposerAddress, votes[i].leafIndex)
            ) {
                _processVote(votes[i], proposerAddress);
            }
        }
```

Note that the function calls `_processVote` in each iteration and whose implementation is shown below:

```
// If the proposer has already voted, remove their previous vote
        if (alreadyVoted[currentVotingPeriod][voter] != address(0x0)) {
            address previousVote = alreadyVoted[currentVotingPeriod][voter
    ];
            numOfVotesForGovernor[currentVotingPeriod][previousVote] -= 1;
        }

        alreadyVoted[currentVotingPeriod][voter] = vote.proposedGovernor;
        numOfVotesForGovernor[currentVotingPeriod][vote.proposedGovernor]
    += 1;
        _tryResolveVote(vote.proposedGovernor);
```

Observe that the votes are tallied by the `currentVotingPeriod`. After processing a vote, `_processVote` calls `_tryResolveVote` to determine if a new leader is elected, and if a new leader is elected, it will transfer ownership by calling `_transferOwnership` shown below:

 © 2023 Veridise Inc.

```
require(newOwner != address(0), "Governable: New governor is the zero
    address");
address previousGovernor = governor;
governor = newOwner;
lastGovernorUpdateTime = block.timestamp;
currentVotingPeriod++;
emit GovernanceOwnershipTransferred(previousGovernor, newOwner,
    lastGovernorUpdateTime);
```

The key issue is that, while `_transferOwnership` increments the voting period, `voteInFavorForceSetGovernorWithSi` does not check whether the ownership has changed. As a result, subsequent votes after ownership transfer will get tallied for the next voting period.

Impact One attack scenario is the following:

1. Voter A knows they will have their privileges stripped by the next round so they submit multiple votes (offchain) hoping one appears at the beginning and another appears at the end of the list.
2. When processing the votes, the ownership is transferred somewhere in the middle of the list before one of the A's votes.
3. `voteInFavorForceSetGovernorWithSig` adds their votes to the next round and their vote counts for the next period even though their privileges are revoked.

Recommendation One way to simplify the logic is to separate the ownership transfer logic from the vote processing logic. In particular, it makes sense to transfer ownership only after tallying all the votes.

4.1.8 V-WBT-VUL-008: RateLimitedVAnchor allows larger withdrawals if no withdrawals occur for several days

Severity	Medium	**Commit**	848d073
Type	Logic Error	**Status**	Fixed
File(s)	`RateLimitedVAnchor.sol`		
Location(s)	transact		

The `transact` function limits the amount of withdraws by tracking `currentDailyWithdrawal` and limiting that value to `DAILY_WITHDRAWAL_LIMIT` within one day of `startTime`. Once a full day has passed, the `currentDailyWithdrawl` will be reset and `startTime` will be incremented by 1 day.

```
if (block.timestamp < startTime + 1 days) {
    currentDailyWithdrawal = (_externalData.extAmount < 0)
      ? currentDailyWithdrawal + uint256(-_externalData.extAmount)
      : currentDailyWithdrawal;
} else {
    // If we are in a new day, reset the currentDailyWithdrawal and set the new
    startTime
    currentDailyWithdrawal = (_externalData.extAmount < 0)
        ? uint256(-_externalData.extAmount)
        : 0;
    startTime = startTime + 1 days;
}
```

Snippet 4.7: `startTime` update logic from `transact`

However, if there aren't any withdrawals for multiple days, one user could end up withdrawing more than the daily limit, since `startTime` is only incremented one day at a time. For example, if `block.timestamp = startTime + 7 days`, the first transact call will increment `startTime` by one, making `block.timestamp = startTime + 6 days` and resetting the `currentDailyWithdrawal` again.

Impact More than `DAILY_WITHDRAWL_LIMIT` can be withdrawn from the contract, defeating the rate limit set by the handler if no withdrawals happen for multiple days.

Recommendation Change the logic when resetting the `currentDailyWithdrawal` to set `startTime = block.timestamp`, which will make the `currentDailyWithdrawal` count for the next 24 hours after the transaction that reset the limit.

© 2023 Veridise Inc.

4.1.9 V-WBT-VUL-009: Voting Results May Change On Vote Order

Severity	Medium	**Commit**	848d073
Type	Frontrunning	**Status**	Intended Behavior
File(s)	`Governable.sol`		
Location(s)	`voteInFavorForceSetGovernor`, `voteInFavorForceSetGovernorWithSig`		

Since `_processVote` immediately changes the governor to a proposed governor once `voterCount / 2 + 1` votes are received, the new governor can be manipulated in multiple ways:

1. `voteInFavorForceSetGovernor` transactions can be front-run to hand the governorship to a minority candidate. For example, if `voterCount` is 5 (requiring 3 votes) and there are 5 votes for proposed governor Foo in the mempool, frontrunning 3 votes for Bar will result in Bar winning and the 5 votes for Foo being rejected (as the period has changed and `isValidTimeToVote` will fail).
2. `voteInFavorForceSetGovernorWithSig` can be manipulated depending on how the arguments are sorted. For example, if the `votes` argument contains [Foo, Bar, Foo, Bar, Foo, Foo, Foo, Foo] and 3 votes are required to win, the original sorting of the list will result in Foo winning, but sorting the list as [Foo, Foo, Foo, Foo, Foo, Bar, Bar, Bar] will result in Bar winning (as the period can currently change multiple times during this function per Issue V-WBT-VUL-007).

Impact The result of the governor election may be maliciously manipulated either via frontrunning or by the submitter of the batch voting process.

Recommendation The recommendation presented in Issue V-WBT-VUL-007 will resolve the issue for `voteInFavorForceSetGovernorWithSig`. To additionally fix the frontrunning issue in `voteInFavorForceSetGovernor`, a mechanism to close all voting and then process all tallied votes in a separate block should be introduced should be introduced so that all submitted votes will be considered when changing the governor.

Developer Response The `voterCount` is also assumed to be the maximum number of voters in the system. Therefore, if `voterCount / 2` votes are in favor of a new governor, we can safely assume that there is no other governor that can get more votes.

4.1.10 V-WBT-VUL-010: Potential Problems with transfer

Severity	Medium	Commit	848d073
Type	Locked Funds	Status	Fixed
File(s)	`Treasury.sol, TokenWrapper.sol`		
Location(s)	`rescueTokens,_wrapForAndSendTo,_unwrapAndSendTo`		

Solidity has multiple functions that can be used to send funds to other users. The `transfer` function is commonly considered the safest as it strictly limits the amount of gas the callee may to prevent external calls (and therefore reentrancy) and will revert if the fallback function reverts. Recently, several popular user wallets have implemented more complicated fallback functions which consume more gas than `transfer` provides.

```
function _unwrapAndSendTo(
    address sender,
    address tokenAddress,
    uint256 amount,
    address recipient
) internal {
    ...

    if (tokenAddress == address(0)) {
        // transfer native liquidity from the token wrapper to the sender
        payable(recipient).transfer(amount);
    }

  ...
}
```

Snippet 4.8: Function where `transfer` is called to send funds to users

Impact In some cases, the use of `transfer` has cause funds to become locked in the contracts for users who make use of wallets with expensive fallback functions.

Recommendation While we believe that `transfer` often improves the security of contract implementations, it has recently been causing usability issues for end-users. We therefore recommend using `transfer` when the recipient is known and low-level calls when sending funds to end-users.

 © 2023 Veridise Inc.

4.1.11 V-WBT-VUL-011: Different Tokens may use Single Token Wrapper

Severity	Medium	Commit	848d073
Type	Logic Error	Status	Acknowledged
File(s)	FungibleTokenWrapper.sol		
Location(s)	add		

The webb protocol wraps tokens that may be bridged to other chains so that the wrapped token may be minted to indicate that funds are owed to a user. The protocol additionally allows the governor to specify multiple token implementations that may wrapped with a single wrapped token. Conceptually this can allow tokens with the same theoretical value, such as stablecoins, to be wrapped with a single wrapped token.

```
function add(
    address _tokenAddress,
    uint32 _nonce
) external override onlyHandler onlyInitialized onlyIncrementingByOne(_nonce) {
    require(!valid[_tokenAddress], "FungibleTokenWrapper: Token should not be valid")
    ;
    tokens.push(_tokenAddress);

    if (!historicallyValid[_tokenAddress]) {
        historicalTokens.push(_tokenAddress);
        historicallyValid[_tokenAddress] = true;
    }
    valid[_tokenAddress] = true;
    emit TokenAdded(_tokenAddress);
}
```

Snippet 4.9: Function to add valid tokens to wrap/unwrap

Impact Given value fluctuations even in stablecoins, bundling multiple tokens is risky. In events where the price of one underlying token fluctuates (such as with stablecoin depegs), this could allow users to exchange less valuable tokens for more valuable tokens.

Recommendation Only allow a single token to be wrapped with a `WrappedToken`.

Developer Response The risk of adding multiple tokens is managed through the governance system. The bridge is designed such that all tokens supported by the bridge should have the same inherent value (i.e. one bridge for ETH and a separate bridge for USD stablecoins). If this is the case, there should be very little risk in wrapping multiple tokens with the same wrapper.

4.1.12 V-WBT-VUL-012: Native Token Wrapping may Bypass Limit

Severity	Low	**Commit**	848d073
Type	Logic Error	**Status**	Fixed
File(s)	`TokenWrapper.sol`		
Location(s)	`isValidWrapping`		

The Webb developers limit the number of funds that may be owned by the protocol by restricting the number of wrapped tokens that may be created. To do so, the `isValidWrapping` modifier calls `_isValidAmount` to check that the total supply is less than the limit after minting. As shown in the code below, however, the limit will not be checked for native tokens since `_amount` is always zero in the case where native tokens are used.

```
modifier isValidWrapping(
    address _tokenAddress,
    address _feeRecipient,
    uint256 _amount
) {
    if (_tokenAddress == address(0)) {
        require(_amount == 0, "TokenWrapper: Invalid amount provided for native
    wrapping");
        ...
    } else {
        ...
    }

    require(_feeRecipient != address(0), "TokenWrapper: Fee Recipient cannot be zero
    address");
    require(_isValidAmount(_amount), "TokenWrapper: Invalid token amount");
    _;
}
```

Snippet 4.10: The implementation of the `isValidWrapping` modifier

Recommendation If native tokens are used, pass `msg.value` into `_isValidAmount`.

© 2023 Veridise Inc.

4.1.13 V-WBT-VUL-013: Missing zeroLeaf check

Severity	Low	**Commit**	848d073
Type	Data Validation	**Status**	Fixed
File(s)	`transaction.circom`, `transactionForest.circom`		
Location(s)	`Transaction`, `TransactionForest`		

The `Transaction` and `TransactionForest` templates takes a `zeroLeaf` parameter, but do not verify if the given transaction uses this `zeroLeaf` parameter.

Impact A transaction may fail to verify if the wrong zero leaf has been used, and the user may be unaware of why this error occurred if `zeroLeaf` is not explicitly checked.

Recommendation Add validation that `zeroLeaf` is consistent with the other parameters and inputs to `Transaction` and `TransactionForest`.

4.1.14 V-WBT-VUL-014: Multiple Voting Periods in One Transaction If Session Length Is 0

Severity	Low	**Commit**	848d073
Type	Transaction Ordering	**Status**	Fixed
File(s)	`Governable.sol`		
Location(s)	`isValidTimeToVote`		

`isValidTimeToVote` checks if another vote for a new governor can occur by seeing if enough time has elapsed since the previous vote.

```
require(
    block.timestamp >=
        lastGovernorUpdateTime +
            ((sessionLengthMultiplier * averageSessionLengthInMillisecs) / 1000),
    "Governable: Invalid time for vote"
);
```

Snippet 4.11: Logic checking the valid voting time

However, if `averageSessionLengthInMillisecs` is set to 0 (which is not prevented), `lastGovernorUpdateTime` may be equal to `block.timestamp` and another vote would still be allowed. This means that the governor could change multiple times during the same period.

Impact This leads to the voting period being changed multiple times during a single transaction. Furthermore, votes that would normally be rejected (due to the governor being changed recently) are instead accepted and used to change the governor.

Recommendation Either change the logic of `isValidTimeToVote` to ensure that the block timestamp must change between votes using a greater-than (rather than greater-than-or-equals) relation:

```
block.timestamp >
    lastGovernorUpdateTime +
        ((sessionLengthMultiplier * averageSessionLengthInMillisecs) / 1000)
```

Or, ensure that `averageSessionLengthInMillisecs` must be non-zero.

 © 2023 Veridise Inc.

4.1.15 V-WBT-VUL-015: Voting disabled due to default value of averageSessionLengthInMillisecs

Severity	Low	**Commit**	848d073
Type	Denial of Service	**Status**	Intended Behavior
File(s)	`Governable.sol`		
Location(s)	`averageSessionLengthInMillisecs`		

The default value of `averageSessionLengthInMillisecs` is `2 ** 64 - 1` (i.e., uint64 max), which effectively disables voting (as `isValidTimeToVote` blocks all voting until this time has elapsed). `averageSessionLengthInMillisecs` can only be changed if `transferOwnershipWithSignature` is called (which must be signed by the governor).

Impact If the governor becomes unresponsive or malicious before the initialization of `averageSessionLengthInMillisecs` to a reasonable value, the contract becomes unrecoverable.

Recommendation Allow `averageSessionLengthInMillisecs` to be set in the constructor so voting can occur without governor intervention.

Developer Response This behavior is intended.

4.1.16 V-WBT-VUL-016: Validate Calldata Has Correct Size

Severity	Low	**Commit**	848d073
Type	Data Validation	**Status**	Fixed
File(s)	`AnchorHandler.sol`, `TokenWrapperHandler.sol`, `TreasuryHandler.sol`, `SignatureBridge.sol`		
Location(s)	`executeProposal`, `_handleSetResource`		

The bridge allows the governor to execute various proposals and set resources by signing and encoding a message as calldata. Assuming the signature is valid, the message will be decoded to execute some function, many of which are security-critical. When decoding and executing the action, the length of the calldata is not validated as shown below.

```
function executeProposal(bytes32 resourceID, bytes calldata data)
    external override onlyBridge {
    ...

    if (functionSig == bytes4(keccak256("setHandler(address,uint32)"))) {
        uint32 nonce = uint32(bytes4(arguments[0:4]));
        address newHandler = address(bytes20(arguments[4:24]));
        treasury.setHandler(newHandler, nonce);
    } else if (
        functionSig == bytes4(keccak256("rescueTokens(address,address,uint256,uint32)
    "))
    ) {
        uint32 nonce = uint32(bytes4(arguments[0:4]));
        address tokenAddress = address(bytes20(arguments[4:24]));
        address payable to = payable(address(bytes20(arguments[24:44])));
        uint256 amountToRescue = uint256(bytes32(arguments[44:76]));
        treasury.rescueTokens(tokenAddress, to, amountToRescue, nonce);
    } else {
        revert("TreasuryHandler: Invalid function sig");
    }
}
```

Snippet 4.12: Location where calldata is accessed without validating the size

Impact Since solidity silently allows calldata to be read past the length by returning 0, this can result in configuration errors.

Recommendation Validate the that the calldata is of appropriate length.

© 2023 Veridise Inc.

4.1.17 V-WBT-VUL-017: Missing Address Checks in VAnchorVerifier

Severity	Warning	**Commit**	848d073
Type	Usability Issue	**Status**	Fixed
File(s)	`VAnchorVerifier.sol`		
Location(s)	`constructor`		

The constructor to `VAnchorVerifier` accepts four contract types as arguments. An invalid address could be passed into the constructor accidentally, which would cause `verifyProof` to revert when accessing the invalid addresses.

Impact If an incorrect or invalid address is provided, it could cause the contract to become non-functional, requiring the contract to be redeployed.

Recommendation Several possible fixes could be implemented:

- Add checks to ensure constructor arguments are not `address(0)` (though invalid addresses could still be provided).
- Add setter functions for the `IVAnchorVerifier*` member variables of the contract that can only be invoked by the contract owner and/or authorized users.

4.1.18 V-WBT-VUL-018: Missing Address Checks in TxProofVerifier

Severity	Warning	**Commit**	848d073
Type	Usability Issue	**Status**	Fixed
File(s)	`TxProofVerifier.sol`		
Location(s)	`constructor`		

The constructor to `TxProofVerifier` accepts a `IAnchorVerifier` type as an arguments. An invalid address could be passed into the constructor accidentally, which would cause `verifyProof` to revert when accessing the invalid `IAnchorVerifier` address.

Impact If an incorrect or invalid address is provided, it could cause the contract to become non-functional, requiring the contract to be redeployed.

Recommendation Several possible fixes could be implemented:

- Add checks to ensure the constructor argument are not `address(0)` (though an invalid addresse could still be provided).
- Add setter functions for the `verifier` member variable of the contract that can only be invoked by the contract owner and/or authorized users.
- Check that `_verifier.code` is not empty in the constructor.

© 2023 Veridise Inc.

4.1.19 V-WBT-VUL-019: Invalid Argument Reverts with Invalid Proof

Severity	Warning	**Commit**	848d073
Type	Usability Issue	**Status**	Fixed
File(s)	`VAnchorVerifier.sol`		
Location(s)	`verifyProof`		

In `verifyProof`, if `maxEdges != 1 && maxEdges != 7`, `verifyProof` will always return false. This will result in callers (e.g., `TxProofVerifier.verify`) assuming that the proof is invalid and therefore reverting the transaction, even though the proof may be valid and the arguments were accidentally misconfigured.

Impact This can cause confusion to callers about the nature of the failure, making debugging the true issue (i.e., invalid arguments) more difficult.

Recommendation If `maxEdges != 1 && maxEdges != 7`, revert in `verifyProof` with an error that indicates that the `maxEdges` argument is invalid.

4.1.20 V-WBT-VUL-020: Unchecked Bytes Length in decodeRoots

Severity	Warning	**Commit**	848d073
Type	Data Validation	**Status**	Fixed
File(s)	`LinkableIncrementalBinaryTree.sol`, `LinkableAnchor.sol`		
Location(s)	`decodeRoots`		

The function `decodeRoots` takes as input a parameter `roots` which is a collection of bytes representing Merkle roots and converts them into array of `bytes32` of length `self.maxEdges + 1`. It does so by iterating over `roots` in 32 byte chunks and converting each chunk. However, the loop bound is `i <= self.maxEdges` which is independent of the size of `roots` and the function does not check whether `roots` is sufficiently large to iterate. This is seen in the following code snippet:

```
function decodeRoots(
        LinkableIncrementalTreeData storage self,
        bytes calldata roots
    ) internal view returns (bytes32[] memory) {
        bytes32[] memory decodedRoots = new bytes32[](self.maxEdges + 1);
        for (uint256 i = 0; i <= self.maxEdges; i++) {
            decodedRoots[i] = bytes32(roots[(32 * i):(32 * (i + 1))]);
        }

        return decodedRoots;
    }
```

If the length of `roots` is less than `32 * (maxEdges + 1)` then this function will revert with an index out of bounds error.

Impact This function is currently not used anywhere else but may end up unnecessarily wasting gas when `roots` is insufficiently large.

Recommendation We recommend adding a check at the beginning of the function to ensure that `roots` is sufficiently large.

© 2023 Veridise Inc.

4.1.21 V-WBT-VUL-021: Anyone May Deploy Contract via Deployer

Severity	Warning	**Commit**	848d073
Type	Authorization	**Status**	Intended Behavior
File(s)	`DeterministicDeployFactory.sol`		
Location(s)	`deploy`, `deployFungibleToken`, `deployVAnchor`		

The `DeterministicDeployFactory` allows anyone to deploy a contract via this one as shown below. While this does not present an inherent problem on the blockchain, it could cause problems off-chain if monitoring for contracts deployed via this contract.

```
function deploy(bytes memory bytecode, uint _salt) external returns (
    address) {
    address addr;
    assembly {
        addr := create2(0, add(bytecode, 0x20), mload(bytecode), _salt)
        if iszero(extcodesize(addr)) {
            revert(0, 0)
        }
    }

    emit Deploy(addr);
    return addr;
}
```

Impact If an off-chain component is used to track or trust addresses deployed via this contract, it may be possible for the off-chain component to trust malicious contracts.

Recommendation If it is important that only certain entities perform deployments via this contract, consider adding access controls to the deploy functions.

Developer Response This behavior is intended, as we aren't performing any monitoring of deployments using this contract.

4.1.22 V-WBT-VUL-022: External Function Should Be Public

Severity	Warning	**Commit**	848d073
Type	Gas Optimization	**Status**	Fixed
File(s)	`DeterministicDeployFactory.sol`, `IMerkleSystem.sol`, `ChainIdWithType.sol`		
Location(s)	`deploy`, `getZeroHash`, `isKnownRoot`, `isCorrectExecutionChain`		

The `DeterministicDeployFactory` provides a function to deploy a contract given some bytecode. This function is currently marked as external despite being used internally in the contract.

```
function deploy(bytes memory bytecode, uint _salt) external returns (
    address) {
    address addr;
    assembly {
        addr := create2(0, add(bytecode, 0x20), mload(bytecode), _salt)
        if iszero(extcodesize(addr)) {
            revert(0, 0)
        }
    }

    emit Deploy(addr);
    return addr;
}
```

Similarly, `IMerkleSystem` declares several external functions including `getZeroHash` and `isKnownRoot`, and `ChainIdWithType` similarly declares `isCorrectExecutionChain` as external. These functions are used by inherited contracts, though, and should be public rather than external.

Impact Calling a public function within a contract is more gas-efficient than calling an external function.

Recommendation Make these functions public rather than external.

© 2023 Veridise Inc.

4.1.23 V-WBT-VUL-023: Batch Voting Tallying Logic Ambiguous due to Ownership Transfer in the Middle

Severity	Warning	Commit	848d073
Type	Logic Error	Status	Intended Behavior
File(s)	`Governable.sol`		
Location(s)	`voteInFavorForceSetGovernorWithSig`		

The function `voteInFavorForceSetGovernorWithSig` tallies a list of votes and elects a new governor if the number of votes for a candidate exceeds simple majority. The function process the list of votes one by one and after processing each vote will check if a candidate has more than the simple majority. In that case, it will transfer ownership in the middle of the vote but continue processing the rest. One potential issue is due to the fact that there could be duplicate votes in the list. The policy of the contract seems to handle this by only considering a voter's last vote in the list. This is seen in the function `_processVote` shown below:

```
// If the proposer has already voted, remove their previous vote
if (alreadyVoted[currentVotingPeriod][voter] != address(0x0)) {
    address previousVote = alreadyVoted[currentVotingPeriod][voter];
    numOfVotesForGovernor[currentVotingPeriod][previousVote] -= 1;
}

alreadyVoted[currentVotingPeriod][voter] = vote.proposedGovernor;
numOfVotesForGovernor[currentVotingPeriod][vote.proposedGovernor] += 1;
_tryResolveVote(vote.proposedGovernor);
```

If a voter already voted during the voting period, then the function will remove their previous vote and consider the current one being processed. However, after processing each vote, `_processVote` calls `_tryResolveVote` to determine if ownership should be transferred and performs the transfer if it should. However, by transferring ownership in the middle, this violates the policy of the contract to only consider the last vote. Consider the following scenario:

There are two voters A, B and the list of votes are:

A - vote for B

B - vote for B

A - vote for A

and suppose the governor is A. In that case, the contract will elect a new governor after processing B's vote; however, according the policy it seems the vote should be split with 1 vote for A and 1 for B.

Impact The main impact is an incorrect processing of the votes and someone could be elected who shouldn't.

Recommendation The easiest fix would be to elect a governor after processing all the votes in the list.

Developer Response The `voterCount` is also assumed to be the maximum number of voters in the system. Therefore, if `voterCount / 2` votes are in favor of a new governor, we can safely assume that there is no other governor that can get more votes.

© 2023 Veridise Inc.

4.1.24 V-WBT-VUL-024: Missing Address Check in LinkableAnchor

Severity	Warning	**Commit**	848d073
Type	Usability Issue	**Status**	Fixed
File(s)	`LinkableAnchor.sol`		
Location(s)	`constructor`		

The contract's constructor does not validate whether the handler is the 0 address as shown below:

```
/// @notice The LinkableAnchor constructor
/// @param _handler The address of the ‘AnchorHandler‘ contract
/// @param _outerTreeHeight The height of outer-most merkle tree
/// @param _maxEdges The maximum # of edges this linkable tree connects to
constructor(address _handler, uint32 _outerTreeHeight, uint8 _maxEdges) {
    handler = _handler;
    outerLevels = _outerTreeHeight;
    maxEdges = _maxEdges;
}
```

This can cause the contract to revert when accessing an invalid handler.

Impact If an incorrect or invalid address is provided, it could cause the contract to become non-functional, requiring the contract to be redeployed.

Recommendation Add a `require` asserting that `_handler` is not equal to 0.

4.1.25 V-WBT-VUL-025: nthZero returns invalid leafs for out-of-bounds batchLevels

Severity	Warning	**Commit**	848d073
Type	Data Validation	**Status**	Fixed
File(s)	`batchMerkleTreeUpdate.circom`		
Location(s)	`nthZero`		

`nthZero` supports zero leafs for `0 <= n <= 15` and returns 0 for higher levels. If a `BatchTreeUpdate` is erroneously created with `batchLevels > 15` (and therefore `zeroBatchLeaf = 0`), the update will fail to verify the updates.

Impact This can result in failing computation without notifying the user of the true source of the error, as the verification will fail without telling the user that a batch size outsize of the range is unsupported.

Recommendation Rather than returning 0, add an assertion that `n` must be between 0 and 15 so the error is more clearly communicated to the user.

© 2023 Veridise Inc.

4.1.26 V-WBT-VUL-026: Create2 Compatibility Problems

Severity	Warning	**Commit**	848d073
Type	Compatibility Issue	**Status**	Acknowledged
File(s)	`DeterministicDeployFactory.sol`		
Location(s)	`deploy`		

The `DeterministicDeployFactory` allows users to pass in `bytecode` that will be deployed via the `create2` operation. On most chains, this will work as intended, however zkSync requires that any bytecode passed into `deploy` can be determined statically (i.e. the bytecode must be fixed).

```
function deploy(bytes memory bytecode, uint _salt) external returns (address) {
    address addr;
    assembly {
        addr := create2(0, add(bytecode, 0x20), mload(bytecode), _salt)
        if iszero(extcodesize(addr)) {
            revert(0, 0)
        }
    }
    ...
}
```

Snippet 4.13: Function for deploying new contracts that uses the `create2` operation

Impact This contract will be unable to be deployed on zkSync.

Recommendation If possible, ensure the `bytecode` can be determined statically if this will be deployed on zkSync.

Developer Response We have noted this and won't deploy the contract on zkSync.

4.1.27 V-WBT-VUL-027: Hardcoded Address Compatibility Problems

Severity	Warning	**Commit**	848d073
Type	Compatibility Issue	**Status**	Acknowledged
File(s)	`SanctionFilter.sol`		
Location(s)	`SANCTIONS_CONTRACT`		

To prevent sanctioned users from interacting with the bridge, Webb makes use of the Chainalysis sanction oracle. While this oracle appears to be present on most of the chains Webb wants to deploy on, it is absent from a few, such as zkSync.

```
contract SanctionFilter {
    address constant SANCTIONS_CONTRACT = 0x40C57923924B5c5c5455c48D93317139ADDaC8fb;
    ...
}
```

Snippet 4.14: Contract to prevent interactions with sanctioned addresses that uses a hardcoded address

Impact On chains without the Chainalysis oracle, calling the contract will likely cause transactions to revert, breaking the anchor.

Recommendation Consider making the address configurable or validate before deployment that the contract exists on the deployment chain.

Developer Response We have noted this and will consult with the Chainanalysis documentation to determine if there is a sanction list contract deployed on the chain before we use this contract.

 © 2023 Veridise Inc.

4.1.28 V-WBT-VUL-028: No Depth to Voter Merkle Tree

Severity	Warning	Commit	848d073
Type	Data Validation	Status	Fixed
File(s)	Governable.sol		
Location(s)	_isValidMerkleProof		

The `Governable` contract allows the governor to define the set of voters by setting a merkle tree root, where the content of the merkle tree is the hashes of voter addresses. No depth is required though, instead users will implicitly provide the depth of the tree when casting votes as shown below.

```
function _isValidMerkleProof(
    bytes32[] memory siblingPathNodes,
    address leaf,
    uint32 leafIndex
) internal view returns (bool) {
    bytes32 leafHash = keccak256(abi.encodePacked(leaf));
    bytes32 currNodeHash = leafHash;
    uint32 nodeIndex = leafIndex;
    for (uint8 i = 0; i < siblingPathNodes.length; i++) {
        if (nodeIndex % 2 == 0) {
            currNodeHash = keccak256(abi.encodePacked(currNodeHash, siblingPathNodes[
    i]));
        } else {
            currNodeHash = keccak256(abi.encodePacked(siblingPathNodes[i],
    currNodeHash));
        }
        nodeIndex = nodeIndex / 2;
    }
    return voterMerkleRoot == currNodeHash;
}
```

Snippet 4.15: Location where a merkle proof is validated without considering the depth of the voter tree

Impact The voting mechanism is open to a second pre-image attack. While these attacks are computationally difficult with hashes of this size, allowing users to construct arbitrary-sized increases their chances of success. As an example, consider the case where a user happens to own the address that hashes to the root. They will always be able to vote by providing an empty siblings array.

Recommendation Require that the governor also provide the depth of the tree and perhaps the zero value used to fill empty leaves.

4.1.29 V-WBT-VUL-029: Use Constant Instead of Magic Number

Severity	Warning	**Commit**	848d073
Type	Maintainability	**Status**	Fixed
File(s)	`TokenWrapper.sol`		
Location(s)	`getFeeFromAmount`, `getAmountToWrap`		

In the `TokenWrapper` function, `10000` is used to represent 100% when computing the fee percentage. This constant is used multiple times throughout the file, which could cause mistakes when changing the contract in the future.

```
function getAmountToWrap(uint256 _deposit) public view override returns (uint256) {
    return (_deposit * 10000) / (10000 - feePercentage);
}
```

Snippet 4.16: Location where repeated numbers are used rather than constants

Impact If this value is updated in the future, it is possible that this could lead to mistakes if all of the values are not updated consistently.

Recommendation Consider using a constant or immutable variable rather than hardcoding the number.

© 2023 Veridise Inc.

4.1.30 V-WBT-VUL-030: Tree Height not Validated

Severity	Warning	**Commit**	848d073
Type	Data Validation	**Status**	Fixed
File(s)	`LinkableAnchors.sol`		
Location(s)	`constructor`		

The `LinkableAnchor` contract stores the height of the outer-most Merkle Trees in a storage variable called `outerLevels` and this value is set in the constructor as shown below:

```
/// @notice The LinkableAnchor constructor
/// @param _handler The address of the 'AnchorHandler' contract
/// @param _outerTreeHeight The height of outer-most merkle tree
/// @param _maxEdges The maximum # of edges this linkable tree connects to
constructor(address _handler, uint32 _outerTreeHeight, uint8 _maxEdges) {
    handler = _handler;
    outerLevels = _outerTreeHeight;
    maxEdges = _maxEdges;
}
```

The contract only supports trees whose heights are between 1 and 32 (levels 0–31) which means the contract could be instantiated with an `outerLevels` value that is not supported. This will result in many functions, namely `getLatestNeighborEdges`, `getLatestNeighborRoots`, `isValidRoots` being unusable as they call `this.getZeroHash(outerLevels-1)` which reverts when `outerLevels - 1` is not in $[0, 31]$.

Impact Many functions in the contract will become unusable if `outerLevels` is incorrectly instantiated.

Recommendation Validate during the constructor that `outerLevels` is within the supported range.

4.1.31 V-WBT-VUL-031: Unnecessary getter function

Severity	Info	**Commit**	848d073
Type	Gas Optimization	**Status**	Fixed
File(s)	`ProposalNonceTracker.sol`		
Location(s)	`getProposalNonce()`		

`getProposalNonce()` is a getter function for the `proposalNonce` variable. However, since `proposalNonce` is a public `uint32`, it already has a getter function generated by the compiler for external callers (`proposalNonce()`).

Impact `getProposalNonce()` increases the size of the contract's code unnecessarily.

Recommendation Remove the `getProposalNonce()` function and replace invocations with `proposalNonce()`.

© 2023 Veridise Inc.

4.1.32 V-WBT-VUL-032: Incorrect documentation of RawMerkleTree

Severity	Info	**Commit**	848d073
Type	Maintainability	**Status**	Fixed
File(s)	`merkleTree.circom`		
Location(s)	`RawMerkleTree`		

The documentation for `RawMerkleTree` implies that it takes a merkle root as an input and verifies that the given root matches the provided leaf and path through the tree. However, the `RawMerkleTree` only computes the root given the leaf and path and relies on the user to perform any given-root verification.

```
1 // Verifies that merkle proof is correct for given merkle root and a leaf
```

Snippet 4.17: The comment in `RawMerkleTree`

This particular comment also appears verbatim in 2 other circom files.

Impact The incorrect documentation can make the code more difficult to understand and maintain.

Recommendation Ensure the documentation is up-to-date for the current version of the implementation.

4.1.33 V-WBT-VUL-033: Deployed circuits are labeled as test

Severity	Info	**Commit**	848d073
Type	Maintainability	**Status**	Fixed
File(s)	`circuits/test/*`		
Location(s)	Directory structure		

The circuits under the `circuits/test` directory are not test circuits, but rather the circuits used for witness generation and the generation of the verifiers in the `contracts/contracts/verifiers` directory. The directory structure gives the impression that these circuits will not be deployed.

Impact This directory structure leads to confusion about how the zero-knowledge proof component of the project is organized and functions.

Recommendation Rename the directory that these circuits are housed in.

© 2023 Veridise Inc.

4.1.34 V-WBT-VUL-034: Missing bounds on MAX_EXT_AMOUNT

Severity	Info	Commit	848d073
Type	Data Validation	Status	Fixed
File(s)	VAnchorBase.sol		
Location(s)	_configureMinimumWithdrawalLimit, _configureMaximumDepositLimit		

`_configureMinimumWithdrawalLimit` and `_configureMaximumDepositLimit` can be set for an amount higher that `MAX_EXT_AMOUNT`, even though `calculatePublicAmount` requires transaction amounts to be between `-MAX_EXT_AMOUNT` and `MAX_EXT_AMOUNT`.

Impact These missing checks can result in uncaught misconfiguration errors that may cause valid transactions to be rejected. For example, if `minimumWithdrawalAmount > MAX_EXT_AMOUNT`, no withdrawal transactions can be processed.

Recommendation Add bounds checks to these setter functions.

4.1.35 V-WBT-VUL-035: Multiple Requires can be simplified to one

Severity	Info	**Commit**	848d073
Type	Gas Optimization	**Status**	Fixed
File(s)	`ProposalNonceTracker.sol`, `Governable.sol`		
Location(s)	`manyIncrementingByOne`, `onlyIncrementingByOne`, `transferOwnershipWithSignature`		

The functions `manyIncrementingByOne`, `onlyIncrementingByOne`, and `transferOwnershipWithSignature` perform two `require` checks to validate the `nonce` when only one is necessary. In particular, all of these functions enforce that for two values `X` and `Y` that `X == Y + 1` by first requiring `Y < X` and then requiring `X <= Y+1`. These two requires could be simplified to `require(X == Y + 1)`.

Impact These additional requires waste gas and make the code more difficult to read.

Recommendation We recommend replacing the two `requires` with a single one of the form `require(X == Y + 1)`.

© 2023 Veridise Inc.

Picus Results 5

We used Picus, our automated verifier for ZK circuits, to check the determinacy of sub-circuits in EVM Bridge. The sub-circuits in scope were those that had at least one output signal. The following table shows the sub-circuits that Picus verified along with the parameters used to instantiate them. In particular, we instantiated all sub-circuits in scope with parameters used in top-level circuits.

Table 5.1: Circuits Verified.

Circuit	Deterministic	Parameters
HashLeftRight	✓	-
DualMux	✓	-
RawMerkleTree	✓	levels $\in \{2, 4, 8\}$
MerkleTree	✓	levels $\in \{2, 4, 8\}$
TreeLayer	✓	height $\in \{2, 4\}$
GetMerkleRoot	✓	levels $\in \{2, 4\}$
KeyPair	✓	-
Signature	✓	-

© 2023 Veridise Inc.