Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	Token Bridge
Timeline	2024-10-01 through 2024-10-01
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	README.md
Source Code	• Fantom-foundation/Bridge #558465d
Auditors	• Jonathan Mevs Auditing Engineer • Julio Aguilar Auditing Engineer • Mustafa Hasan Senior Auditing Engineer

Documentation quality	High	
Test quality	High	
Total Findings	9	Fixed: 7 Acknowledged: 2
High severity findings	1	Fixed: 1
Medium severity findings	3	Fixed: 2 Acknowledged: 1
Low severity findings	3	Fixed: 2 Acknowledged: 1
Undetermined severity findings	0	
Informational findings	2	Fixed: 2

Summary of Findings

In this audit, Quantstamp reviewed the smart contracts responsible for managing the bridging of ERC-20 Tokens to the Sonic Network from Ethereum, and back. This bridge includes an emergency withdrawal mechanism that enables users to withdraw their funds on Ethereum if either the bridge or the Sonic chain becomes non-functional for more than 14 days. If the bridge were to die, it could not be revived.

The bridge follows a classic design, in which tokens to be bridged are locked on the one chain, and minted on another. When withdrawing the assets from a network, they are burned, and unlocked from the deposit contract on the other chain, being transferred back to the depositor. Bridging relies on two core communication channels. One channel is untrusted, and used to communicate proofs between chains, done by the users. The other channel is trusted, and managed by the validators of the Sonic Network itself. These validators periodically update the state root of the other chain, i.e. the state root of Sonic Network is updated on the `StateOracle` contract on Ethereum. Before receiving assets on the destination chain, users must provide a proof, that shows their deposit on Ethereum is present in the trusted state root of that chain.

We assume the Sonic Network to operate as intended. Specifically, we trust that the validators responsible for communicating the state roots act according to network specifications, including slashing enforcements and minimum staking requirements. We also assume that contracts will be deployed with expected configurations, such as the `TokenDeposit` contract being deployed with the `UpdateManager` as the owner.

We discovered a high severity vulnerability that has the potential to halt the bridge operations if both the `updateVerifier` address and `newValidators` are being update in a single call to `UpdateManager.update()`. One medium severity issues reported highlight potential issues that can occur in the case of the bridge becoming inoperative. The other medium severity issue highlights the front-running risk associated with the `update()` function. Beyond that, we highlight lower severity issues, and include Auditor Suggestions for further improvement of the codebase.

We thank the Sonic team for their collaboration throughout the audit. The team provided thorough documentation which proved helpful. The test suite is thorough and has good coverage.

Fix Review Update

The Sonic Bridge team provided fixes to the issues presented in the original report or acknowledged them with sufficient reasoning. During the fix review, we identified a medium severity issue (SONIC-9) that could result in the loss of user funds to the bridge if the `exitAdministrator` were never assigned and the bridge were to go offline. This issue has since been addressed.

There were some new features included during the fix review. Users should be aware that the Sonic Bridge team now has the power to unregister tokens that are supported by the bridge and they require trust to honor all funds moving through this bridge.

There is now more robust management for the situation where the number of validators participating in the bridge may decrease to a point below what is needed to reach a quorum and therefore be unable to verify updates to the state oracles. Previously, in order for a state oracle update to be verified, validators had to always exceed a 2/3 quorum of staking weight. If there wasn't an update within 2 weeks, the bridge would be dead forever. Now, instead of holding the quorum constant, it is implemented as a piecewise function, where over the period of 200 days, the quorum needed changes depending on how long it has been since the last state update.

Specifically, if the bridge hasn't been updated after five days, the quorum required will decrease from 66.67% to 55% at a constant rate over two days. If there still hasn't been an update after the 7th day, the quorum will remain constant at 55% until 182 days have passed. Then, there will be a constant rate decrease from 55% to 25% quorum on days 182-189, after which the 25% quorum will remain from day 189-200, after which the bridge will be completely dead.

We emphasize here that this is only in place for emergency situations such as technical errors. There is no intent or plan to ever let the quorum reach 25% by the Sonic Bridge team, but they make the option possible after long periods of no state update. In the case of such emergency, we note that users may have to wait for funds to be received for extended periods of time, 200 days in the absolute worst scenario. Again, this is a failsafe and not part of the intended normal functioning of the bridge.

We again thank the Sonic team for their commitment to security and collaboration throughout this audit.

ID	DESCRIPTION	SEVERITY	STATUS
SONIC-1	Missing Support for Updating Both the `updateVerifier` and the `newValidators`	High	Fixed
SONIC-2	Funds Could Be Stolen While the Bridge or Sonic Chain Are Dead	Medium	Fixed
SONIC-3	Front-running Can Lead to Stolen Validator Rewards	Medium	Acknowledged
SONIC-4	Missing Support for Automatic Rebasing Tokens	Low	Acknowledged
SONIC-5	Signatures Do Not Follow the EIP-191 Standard	Low	Fixed
SONIC-6	Ignored return value	Low	Fixed
SONIC-7	Usage of `selfdestruct` Can Deny Servicing of `UpdateManager.payFastLane()`	Informational	Fixed
SONIC-8	Multiplication After Division	Informational	Fixed
SONIC-9	Stuck Funds if no `exitAdministrator` Assigned	Medium	Fixed

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

> **Disclaimer**
> Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
   1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

All the Solidity files in `contracts/` were in scope for this audit.

Files Included

Repo: https://github.com/Fantom-foundation/Bridge/tree/main(558465d) Files: contracts/

Files Excluded

Repo: https://github.com/Fantom-foundation/Bridge/tree/main(558465d) Files: Everything else

Operational Considerations

- The SFC contract used in `SFCUpdateVerifier` is out of scope. We trust it to do the following, in accordance with interfaces and expected behaviors:
  - `getValidatorPubKey()`, `getValidator()`
  - `pubKeyAddressToValidatorId()`
  - `totalActiveStake()`
- We assume that `tokenPairs` will be accurately maintained between chains. That is, users will always be able to bridge back and forth.
- We assume that all tokens minted to a user on Sonic will ONLY be minted from the `Bridge` contract. This way, there is always a 1:1 relationship between the coins in existence on Ethereum, and those on Sonic.
- The `DirectExitAdministrator` assumes that the `balance` mapping in the `MintedERC20` token is stored in the first storage slot (index 0). This assumption is based on the current OpenZeppelin implementation of the ERC20 standard. For future updates or redeployments, ensure that this assumption remains valid, or adjust the slot index accordingly if the storage layout changes.
- `TokenPairs.REGISTER_ROLE` has the power to unregister tokens, making them unsupported by the bridge. Trust is required in the Sonic Bridge team to honor all funds in the bridge.

Key Actors And Their Capabilities

`ActivityMonitor` Contract Owner

Key Responsibilities:

- Mark activity for validators to maintain a state on Sonic that detects participation of a validator in a given time range.

Trust Assumptions:

- The Activity Monitor Contract Owner will only invoke `markActivity()` honestly and as expected.

Functions:

- `ActivityMonitor.markActivity()`
- `Ownable.renounceOwnership()`
- `Ownable.transferOwnership()`

`Bridge` Contract Owner

Key Responsibilities:

- Manage the `deposit` and `proofVerifier` addresses on the `Bridge` contract, the Sonic Network portion of the Bridge.
- After the token deposit address as been assigned, we assume the owner to be the `UpdateManager` contract.

Trust Assumptions:

- All addresses will be accurately assigned.

Functions:

- `Bridge.setTokenDepositAddress()`
- `Bridge.setProofVerifier()`
- `Ownable.renounceOwnership()`
- `Ownable.transferOwnership()`

`ExitAdministrator` Roles

Key Responsibilities:

- The `DEFAULT_ADMIN_ROLE` of the `ExitAdministrator` is responsible for assigning the token deposit address, as well as other roles.
- The `IMPLEMENTATION_ROLE` is responsible for invoking `withdrawWhileDead()` to support Ethereum withdraws in the case of a dead bridge. We assume this to be the `DirectExitAdministrator` contract.

Trust Assumptions:

- All addresses will be accurately assigned.
- `withdrawWhileDead()` will only be invoked by the `IMPLEMENTATION_ROLE` as needed.

Functions:

`DEFAULT_ADMIN_ROLE`

- `AccessControl.grantRole()`
- `AccessControl.renounceRole()`
- `AccessControl.revokeRole()`
- `ExitAdministrator.setTokenDeposit()`

`IMPLEMENTATION_ROLE`

- `ExitAdministrator.withdrawWhileDead()`

`MintedERC20` `minter` address

Key Responsibilities:

- Mint tokens to a user on Sonic. We assume this address to be assigned to the `Bridge` contract on Sonic.

Trust Assumptions:

- Mints will only take place after a complete and accurate bridging process.

Functions:

- `MintedERC20.mint()`

`StateOracle` Contract Owner

Key Responsibilities:

- Update the state of the other chain, on a given chain. We assume this owner to be assigned to the `UpdateManager` contract.

Trust Assumptions:

- Updates will only done in accordance with the overall Sonic network and its validators.

Functions:

- `StateOracle.update()`
- `Ownable.renounceOwnership()`
- `Ownable.transferOwnership()`

`TokenDeposit` Contract Owner

Key Responsibilities:

- Reassign the `proofVerifier` address as needed. We assume the owner to be the `UpdateManager` contract.

Trust Assumptions:

- All addresses will be accurately assigned.

Functions:

- `TokenDeposit.setProofVerifier()`
- `Ownable.renounceOwnership()`
- `Ownable.transferOwnership()`

`TokenPairs` Roles

Key Responsibilities:

- The `DEFAULT_ADMIN_ROLE` of the `TokenPairs` contract is responsible for managing the `REGISTER_ROLE` of this contract.
- The `REGISTER_ROLE` is responsible for registering pairs of token addresses between Ethereum and Sonic.

Trust Assumptions:

- All addresses will be accurately assigned.

Functions:

`DEFAULT_ADMIN_ROLE`

- `AccessControl.grantRole()`
- `AccessControl.renounceRole()`
- `AccessControl.revokeRole()`

`REGISTER_ROLE`

- `TokenPairs.register()`

`UpdateManager` Roles

Key Responsibilities:

- The `DEFAULT_ADMIN_ROLE` of the `UpdateManager` contract is responsible for managing the roles in this contract.
- The `HEARTBEAT_ROLE` is responsible for assigning the heartbeat of the updates done to the bridge state between chains.
- The `FEE_SETTER_ROLE` is responsible for setting the fast lane fee.
- The `MONITOR_SETTER_ROLE` is responsible for setting the activity monitor in this contract.

Trust Assumptions:

- All addresses will be accurately assigned.
- Heartbeats and fees will be assigned in accordance with network guidelines.

Functions:

`DEFAULT_ADMIN_ROLE`

- `AccessControl.grantRole()`
- `AccessControl.renounceRole()`
- `AccessControl.revokeRole()`

`HEARTBEAT_ROLE`

- `UpdateManager.setHeartbeat()`
- `UpdateManager.setFastLanePadding()`

`FEE_SETTER_ROLE`

- `UpdateManager.setFasLaneFee()`

`MONITOR_SETTER_ROLE`

- `UpdateManager.setActivityMonitor()`

`TokenDeposit` Contract Owner

Key Responsibilities:

- Reassign the `proofVerifier` address as needed. We assume the owner to be the `UpdateManager` contract.

Trust Assumptions:

- All addresses will be accurately assigned.

Functions:

- `TokenDeposit.setProofVerifier()`
- `Ownable.renounceOwnership()`
- `Ownable.transferOwnership()`

`ValidatorRegistry` Contract Owner

Key Responsibilities:

- The owner of the `ValidatorRegistry` contract is responsible for setting the validators used to sign off on state updates for the bridge. We assume the owner to be the `UpdateManager` contract.

Trust Assumptions:

- We assume that the `setValidators()` function will only be invoked after the required number of Sonic Network validators have signed the state change.

Functions:

- `ValidatorsRegistry.setValidators()`
- `Ownable.renounceOwnership()`
- `Ownable.transferOwnership()`

Findings

SONIC-1 Missing Support for Updating Both the `updateVerifier` and the `newValidators`

High · Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `53ccf01fb301ed4a082d87b129bff0269cc95d3`. The order has been corrected to allow for both of these states to be updated in tandem.

File(s) affected: `UpdateManager.sol`

Description: If `UpdateManager.update()` were to be invoked with both a list of new validators, and a new update verifier contract, the bridge can potentially become inoperative. This is due to the order in which the conditionals are executed inside this function. The first condition invokes `updateVerifier.setValidators()` if there is a list of `newValidators` specified during the call. However, the third condition, will update `updateVerifier = IUpdateVerifier(_updateVerifier)` in the case where there is a non-zero `_updateVerifier` address specified during the call. With the reassignment of the `updateVerifier` address occuring after `newValidators` are updated in the previous `updateVerifier` address, this new `updateVerifier` address does not have the proper set of new validators assigned in its state.

Further, assuming that the `IUpdateVerifier(ValidatorsRegistry)` contract was deployed with the `UpdateManager` contract assigned as the owner, the bridge would be totally non-operational because there would be no way to invoke `setValidators()` on this new `updateVerifier` contract, because there are no validators defined in its state.

Recommendation: The order of the update should reassign all addresses before invoking any functions on it. Therefore, in the `update()` function, the first and third conditionals should be swapped, so that if there is a new `updateVerifier` address being assigned in conjunction with a new set of validators being defined, the new set of validators will be reflected in the new `updateVerifier` contract.

SONIC-2 Funds Could Be Stolen While the Bridge or Sonic Chain Are Dead

Medium · Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `5dcf7016b6644d5e8f48eb32201fd6ba4e058a80`.
> The client provided the following explanation:
>
> `Admin access replaced by updating the ExitAdministrator by validators update.`

File(s) affected: `ExitAdministrator.sol`

Description: The `DirectExitAdministrator` contract validates the user's proof of minted balance on Sonic and tracks users who have withdrawn their funds from the bridge. In contrast, the `ExitAdministrator` neither performs these validations nor tracks withdrawals. It merely calls the `TokenDeposit` contract, which trusts the `ExitAdministrator` without further verification. If the admin of the `ExitAdministrator` is compromised, the `IMPLEMENTATION_ROLE` could be granted to another address, potentially allowing them to steal all the funds from the bridge.

Furthermore, the additional modularity provided by the `ExitAdministrator` seems redundant. Since it does not offer any meaningful functionality or validation and directly interacts with the `TokenDeposit` contract, its responsibilities can be handled by the `DirectExitAdministrator` after verifying the user's proof.

Recommendation: We recommend removing the `ExitAdministrator` contract and relying solely on the `DirectExitAdministrator` to manage fund withdrawals and validations.

SONIC-3 Front-running Can Lead to Stolen Validator Rewards

Medium · Acknowledged

> **Update**
> Marked as "Acknowledged" by the client.
> The client provided the following explanation:
>
> `This is intended, the reward is intended for anybody who send the update. The reward should cover the update gas fee, should not be set much higher than it.`

File(s) affected: `UpdateManager.sol`

Description: The `update()` function is callable by anyone, allowing an attacker that monitors the transaction pool to submit a transaction calling the function with correct parameters and a higher gas price. As the function transfers the contract balance to the function caller, the attacker may be able to claim a reward intended for validators.

Recommendation: Restrict the function to only be called by validator nodes, or make clear of this inherent risk to users.

SONIC-4 Missing Support for Automatic Rebasing Tokens

Low · Acknowledged

> **Update**
> Marked as "Acknowledged" by the client.
> The client provided the following explanation:
>
> `Needs to be taken into account when adding tokens into TokenPairs registry.`

File(s) affected: `Bridge.sol`, `TokenDeposit.sol`

Description: When depositing tokens on Ethereum to bridge to Sonic Network the `TokenDeposit` contract accounts for any fee-on-transfer tokens by calculating the true balance of tokens received by the contract before beginning the bridging process for a user. However, there is inadequate support for automatic rebasing tokens, such as `stETH`. Users bridging automatically rebasing tokens will eventually have their rebasing rewards stuck in the `TokenDeposit` contract.

Exploit Scenario:

1. Alice deposits `100 stETH` in the `TokenDeposit` contract. These funds remain locked in the contract until burned on Sonic and claimed on Ethereum.
2. Alice claims her `100 stETH` on Sonic Network.
3. Over time, on Ethereum, Alice's balance of `stETH` that is custodied by the `TokenDeposit` contract increases to `105 stETH`
4. `Bridge.withdraw()` only allows her to withdraw `100 stETH`, as that is the only amount of `stETH` she has on Sonic Network.
5. `TokenDeposit.claim()` will only support the `100 stETH` she deposited on Sonic Network, leaving the accrued `5 stETH` of rewards, stuck in the contract.

Recommendation: The simplest solution would be to not support automatically rebasing tokens in the bridging process, and to adequately document this to end users. Supporting automatic rebasing would introduce significant complexities in the code as there would need to be a yield tracking mechanism. We recommend considering internally whether these tokens will be supported, and adjusting the design accordingly.

SONIC-5 Signatures Do Not Follow the EIP-191 Standard

Low · Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `92f9538921bf22adf6b84cd92b1b1678a3f65c7f`; `pr-61`.
>
> This issue has been addressed by adhering to Version `0x00` of EIP-191. `0x19`, `0x00`, and the address of the `UpdateManager` contract are prepended to the update message data (e.g. `blockNum`, `stateRoot` etc), prior to signature recovery and check.

File(s) affected: `ValidatorsRegistry.sol`, `SFCUpdateVerifier.sol`

Description: The `verifyUpdate()` function calls `ECDSA.recover(messageHash, signatures[i])` without first applying `MessageHashUtils.toEthSignedMessageHash()` to the `messageHash`, which is compliant with the EIP-191 standard for signed data. Additionally, this method is recommended as the safer approach in the `ECDSA` file.

```
* IMPORTANT: `hash` _must_ be the result of a hash operation for the
* verification to be secure: it is possible to craft signatures that
* recover to arbitrary addresses for non-hashed data. A safe way to ensure
* this is by receiving a hash of the original message (which may otherwise
* be too long), and then calling {MessageHashUtils-toEthSignedMessageHash} on it.
```

Recommendation: We recommend following with the standard and calling `toEthSignedMessageHash()` on `messageHash`.

SONIC-6 Ignored return value

Low · Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `28272a71d9602b1e3cc18bb2a7f629b1a33dce7f`.

File(s) affected: `Bridge.sol`

Description: The `claim()` function calls `mint()` on the minted token address, which returns a boolean indicating whether the minting succeeded. However, the returned value isn't checked, which may lead to unexpected consequences in case the minting fails.

Recommendation: Require that the returned value is `true`.

SONIC-7 Usage of `selfdestruct` Can Deny Servicing of `UpdateManager.payFastLane()`

Informational · Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `3ddf2e26297b00493437c51e99f0a129850522f3`; review2 branch.
>
> Now, there is a flag toggling whether there was a fee deposited directly through `payFastLane()`. If ETH were to enter this contract in another way, it would not block the invocation of this function.

File(s) affected: `UpdateManager.sol`

Description: The following require statement is included to ensure that once a fast lane reward is deposited, there isn't another one deposited until the first is consumed.

`require(address(this).balance == msg.value, "Fast lane busy");`

This is found in `payFastLane()` a `payable` function, which is intended to be the only entry point for ETH in the contract. However, if a savvy user wanted to disrupt calls to this function, they can do so by forcing ETH into the balance of this contract. Although, there is no real incentive for anyone to do this, as they will be wasting their own money, and ultimately contributing to the fast lane reward, for free. Further, after the first call to `update()` this deposit would be consumed.

Recommendation: We note this here to make the team aware that it is possible to get funds into this contract outside of `payFastLane()`. To totally avoid this, the contract could include a flag that toggles whether there is a pending reward to be claimed that is only turned on when `payFastLane()` is invoked. That way, any ETH that enters the contract outside of this function will not block its invocation.

SONIC-8 Multiplication After Division

Informational · Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `533a396922cf0f4de98566d4501d9ed1b8f5bca6`.

File(s) affected: `SFCUpdateVerifier.sol`, `ValidatorsRegistry.sol`

Description: The `verifyUpdate()` function, found in both `SFCUpdateVerifier` and `ValidatorsRegistry`, contains the following line of code:

`require(weight > totalWeight() / 3 * 2, "Insufficient signatures weight");`

As Solidity only supports integer division, the result of `totalWeight / 3` will be floored prior to being multiplied by 2. There can be slight inaccuracies in this calculation is a 2 / 3 validator weight is trying to be achieved.

Recommendation: To avoid loss of precision, always multiply before dividing: `require(weight > totalWeight() * 2 / 3, "Insufficient signatures weight");`

SONIC-9 Stuck Funds if no `exitAdministrator` Assigned

Medium · Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `3ddf2e26297b00493437c51e99f0a129850522f3`; `pr-61`.
>
> The team explained that the absence of an `exitAdministrator` indicates that the Sonic chain is operating independently of Ethereum. If the bridge were to die and this address were to remain unassigned, then on-dead recovery withdrawals would be disabled on Ethereum. In this case, Sonic chain may proceed as an independent chain, in which case locked funds on Ethereum will be indicative of the value owed to the user on Sonic chain. We emphasize here that such behavior is intended only to occur in emergency situations, where the bridge is considered dead, after 200 days.
>
> The issue described here has been fixed by separating offline time and dead state. Now, the bridge is only considered online if the state has been updated within the last 24 hours.

File(s) affected: `TokenDeposit.sol`

Description: `TokenDeposit.exitAdministrator` is intended to be a privileged address that can invoke `withdrawWhileDead()`. During the fix review the following conditional was included to `TokenDeposit.getDeadStatus()`:

```
if (exitAdministrator == address(0)) {
    return false; // The dying mechanism is disabled if no exitAdministrator is set.
}
```

While the comment included above appears that there is some intention behind having this function return false when there is no `exitAdministrator` set, doing so presents multiple scenarios for users to have their funds stuck in the bridge.

In the scenario where the bridge is non-operational due to insufficient validators participating to meet quorum at a given time, there would be no way for the exit administrator to be assigned, as that process requires the quorum of validators to sign a message indicating the assignment.

Further, this system has no requirement that the `exitAdministrator` be assigned on creation. In the case where this address is not assigned before the bridge becomes non-operational due to decreased validator participation, there will be no way to assign it and funds will be lost.

Exploit Scenario:

Funds stuck during deposit on Ethereum:

1. Alice invokes `TokenDeposit.deposit()` successfully, as the bridge is not considered dead as long as there is no `exitAdministrator` assigned.
2. Upon realizing that the bridge is non-operational, Alice invokes `TokenDeposit.cancelDepositWhileDead()` to retrieve her funds that were already deposited.
3. Alice is never able to successfully cancel the deposit as this require statement found in `cancelDepositWhileDead()` will always revert due to the absence of the exit administrator: `require(getDeadStatus(), "Bridge is not dead");`

Funds stuck during withdrawal on Sonic Network:

1. Alice invokes `Bridge.withdraw()` successfully, with the state changes represented on Sonic Network. This process does not consider whether the bridge is dead.
2. Before the Ethereum State Oracle can get updated with this withdrawal transaction, the bridge dies.
3. At this point, Alice should be able to withdraw her pending funds on Ethereum through `DirectExitAdministrator.withdraw()`. However, this call will always revert due to this require statement: `require(deposit.getDeadStatus(), "Bridge is not dead");`, and the fact that no validators will be able to assign this address.

Recommendation: We recommend invoking `provingContract.setExitAdministrator()` upon deployment of `UpdateManager`. This ensures that there will always be an exit administrator assigned in the system.

Auditor Suggestions

S1 Missing Input Validation

Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `47f00d4be54901865d3f77ee460d1dbe5ed16f61`.

Related Issue(s): SWC-123

Description: It is important to validate inputs, even if they only come from trusted addresses, to avoid human error. All of the contract's constructors should further verify addresses to ensure they are different from `address(0)`.

Recommendation: We recommend adding the relevant checks.

S2 Unlocked Pragma

Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `4d3f45f50e33277e6c4d20d7a017878d84b8141c`.

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.8.*`. The caret (`^`) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, we recommend to remove the caret to lock the file onto a specific Solidity version.

S3 Outdated Solidity Version

Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `4d3f45f50e33277e6c4d20d7a017878d84b8141c`.

Description: As security standards develop, so does the Solidity language. In order to stay up to date with current practices, it's important to use a recent version of Solidity and recent conventions.

Recommendation: Use the latest version of Solidity, 0.8.27, as suggested by documentation here.

S4 Gas Savings

Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `2f469c299f22171a4ae86b57170c2444ea019c89`.
> The client provided the following explanation:
>
> `Variables marked immutable, replacing require() with custom errors skipped as a gas optimization not worth the effort.`

Description: Gas savings can be included in these contracts by preferring custom errors over `require()` statements. Additionally, the following variables in the following contracts can be marked immutable:

- `UpdateManager`
  - `stateOracle`
  - `provingContract`
- `Bridge`
  - `stateOracle`
  - `tokenPairs`
- `DirectExitAdministrator`
  - `deposit`
- `MessageBusVerifier`
  - `updateVerifier`
- `MintedERC20`
  - `dec`
  - `minter`
- `SFCUpdateVerifier`
  - `sfc`
- `StateOracle`
  - `chainId`
- `TokenDeposit`
  - `bridge`
  - `exitAdministrator`
  - `stateOracle`
  - `tokenPairs`

S5 Missing Event Emission

Acknowledged

> **Update**
> Marked as "Acknowledged" by the client.
> The client provided the following explanation:
>
> `When the validator set will be connected to the network SFC, the weights can change very often, adding the event could be expensive with no significant benefit for our monitoring.`

File(s) affected: `ValidatorsRegistry.sol`

Description: `ValidatorsRegistry.setValidators()` should emit an event to better support off-chain monitoring of the bridge.

Recommendation: Emit the recommended event.

S6 Variable Shadowing

Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `bf9f05d926d733ec822939aa263d4ca6b8ecf2b`.

Description: The following places in the code shadow variables and functions that are inherited:

- `ActivityMonitor.constructor()`
  - contains parameter `owner` which shadows `Ownable.owner`
- `Bridge.constructor()`
  - `_owner` shadows vairable in `Ownable`
- `MintedERC20.constructor()`
  - `name` and `symbol` shadow variables in `ERC20`.
- `StateOracle.constructor()`
  - `owner` shadows the function in `Ownable`
- `TokenDeposit.constructor()`
  - `owner` shadows the function in `Ownable`
- `ValidatorsRegistry.constructor()`
  - `owner` shadows the function in `Ownable`

Recommendation: Consider renaming the variables that are responsible for shadowing.

S7 Unused Code

Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `a9345d13ca40b12dcb19633a8f2f46aee105d0f0`.
> The client provided the following explanation:
>
> `TokenPairs.mintedToOriginal kept as it allows to ensure that multiple tokens aren't mapped to a single minted token.`

File(s) affected: `TokenPairs.sol`, `AcitivityMonitor.sol`, `TokenDeposit.sol`

Description: The `TokenPairs.mintedToOriginal` mapping is not used by any other smart contracts. Similarly, in the `TokenDeposit` contract, the `tokenWithdrawn` mapping is also unused. Additionally, the `ActivityMonitor` is solely used to update its internal mapping and is not referenced by other contracts.

Recommendation: We recommend removing the unused code or add documentation stating their external use.

S8 Ownership Can Be Renounced

Acknowledged

> **Update**
> Marked as "Acknowledged" by the client.
> The client provided the following explanation:
>
> `All ownerships are hold by contract UpdateManager which never calls renounceOwnership.`

Description: If the owner renounces their ownership, all ownable contracts will be left without an owner. Consequently, any function guarded by the `onlyOwner` modifier will no longer be able to be executed.

Recommendation: Confirm that this is the intended behavior. If not, override and disable the `renounceOwnership()` function in the affected contracts. For extra security, consider using a two-step process when transferring the ownership of the contract (e.g. `Ownable2Step` from OpenZeppelin).

S9 Critical Role Transfer Not Following Two-Step Pattern

Acknowledged

> **Update**
> Marked as "Acknowledged" by the client.
> The client provided the following explanation:
>
> `All ownerships are hold by contract UpdateManager, no changes of ownership are possible.`

Description: The owner of the contracts can call `transferOwnership()` to transfer the ownership to a new address. If an uncontrollable address is accidentally provided as the new owner address then the contract will no longer have an active owner, and functions with the `onlyOwner` modifier can no longer be executed.

Recommendation: Consider using OpenZeppelin's `Ownable2Step` contract to adopt a two-step ownership pattern in which the new owner must accept their position before the transfer is complete.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.10.0

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Automated Analysis

Slither

We filtered the issues highlighted by Slither and consolidated the relevant issues in this report.

Test Suite Results

All tests are passing and the test suite is comprehensive.

Update:

The tests have been modified to addresses the changes introduced during the fix review. All tests are passing and the test suite remains comprehensive.

```
Bridge
  ✔ Should deploy correctly (46ms)
  ✔ Should succeed and return deposit slot index
  ✔ Should succeed and return hash
  ✔ Should succeed and return withdraw id
  Set token deposit address
    ✔ Should succeed and set token deposit address (77ms)
    ✔ Should revert when deposit address already set (80ms)
  Set proof verifier
    ✔ Should succeed and set proof verifier (78ms)
    ✔ Should revert when not owner
  Claim
    ✔ Should succeed and claim (103ms)
    ✔ Should revert when already claimed (72ms)
    ✔ Should revert when proof is invalid
    ✔ Should revert when no minted token exists (773ms)
  Withdraw
    ✔ Should succeed and withdraw (156ms)
    ✔ Should revert when withdrawal id was already used (48ms)
    ✔ Should revert when no amount given
    ✔ Should revert when no minted token exists
    ✔ Should revert when the bridge is offline (68ms)

MPTProofVerifier
  VerifyProof
    ✔ Should succeed and verify valid proof
    ✔ Should revert when valid proof for a different state root given (95ms)
    ✔ Should revert when invalid value for valid proof given (783ms)
    ✔ Should revert when no-account proof of account non-existence given (114ms)

MessageBus
  ✔ Should deploy correctly
  Vote
    ✔ Should revert when not a validator
    ✔ Should succeed and vote

MintedERC20
  ✔ Should deploy correctly
  Mint
    ✔ Should revert when not minter
    ✔ Should succeed and mint

SFCUpdateVerifier
  ✔ Is possible to iterate the validators list (44ms)
  ✔ Is possible to iterate after deactivating a validator (69ms)
  ✔ Is not possible to insert a validator with invalid pubkey

StateOracle
  ✔ Should deploy correctly
  Update
    ✔ Should revert when not owner
    ✔ Should revert when replaying older state
    ✔ Should succeed and update state

TokenDeposit
  ✔ Should deploy correctly
  ✔ Should succeed and return withdrawal slot index
  ✔ Should succeed and return hash
  ✔ Should succeed and return balance slot index
  ✔ Should succeed and return claim slot index
  ✔ Should succeed and return deposit id
  Set proof verifier
    ✔ Should succeed and set proof verifier
    ✔ Should revert when not owner
  Deposit
    ✔ Should revert when not supported token given
    ✔ Should succeed and deposit tokens (44ms)
    ✔ Should revert when depositing the same id twice
    ✔ Should revert when the bridge is dead
    ✔ Should revert when dead bridge goes alive (94ms)
    ✔ Should revert when no tokens transferred (44ms)
    ✔ Should revert when token transfer returns false (45ms)
    ✔ Should revert when token transfer does not revert on failure (43ms)
    ✔ Should succeed when token with transfer fee given (67ms)
    ✔ Should succeed when token with missing returns given (42ms)
  Claim
    ✔ Should revert when invalid proof given (48ms)
    ✔ Should succeed and claim tokens (64ms)
    ✔ Should revert when claiming the same id twice (69ms)
    ✔ Should revert when token transfer returns false (225ms)
    ✔ Should revert when token transfer does not revert on failure (212ms)
    ✔ Should succeed when token with transfer fee given (68ms)
    ✔ Should succeed when token with missing returns given (71ms)
  Withdraw while dead
    ✔ Should revert when bridge is alive
    ✔ Should revert when not white-listed token given
    ✔ Should revert with invalid proof
    ✔ Should succeed and withdraw tokens (103ms)
    ✔ Should revert when withdrawing the same token twice (270ms)
    ✔ Should succeed when bridge goes live (95ms)
    ✔ Should revert when token transfer returns false (275ms)
    ✔ Should revert when token transfer does not revert on failure (217ms)
    ✔ Should succeed when token with transfer fee given (227ms)
    ✔ Should succeed and withdraw token with missing returns (85ms)
  Cancel deposit while dead
    ✔ Should revert when no deposit to cancel
    ✔ Should revert when bridge is alive
    ✔ Should revert with invalid proof
    ✔ Should succeed and cancel deposit (351ms)
    ✔ Should revert when cancelling twice (93ms)
    ✔ Should succeed when bridge goes live (206ms)
    ✔ Should revert when token with transfer fee given (98ms)
    ✔ Should revert when token missing returns given (113ms)

TokenPairs
  ✔ Should deploy correctly
  Register
    ✔ Should revert when not register role
    ✔ Should revert when original address is zero
    ✔ Should revert when minted address is zero
    ✔ Should succeed and register tokens (53ms)
    ✔ Should revert when original address already registered
    ✔ Should revert when minted address already registered
  Unregister
    ✔ Should revert when not register role
    ✔ Should revert when mapping does not exist
    ✔ Should unregister tokens (48ms)

UpdateManager
  ✔ Should deploy correctly
  ✔ Should succeed and return chain id
  ✔ Should succeed and return last block number
  ✔ Should succeed and return proof verifier address
  ✔ Should succeed and return validators count
  ✔ Should succeed and return validator
  ✔ Should succeed and return validator's weight
  ✔ Should succeed and return validators' total weight
  Set heartbeat
    ✔ Should revert when heartbeat role not granted
    ✔ Should succeed and set heartbeat
    ✔ Should revert when heartbeat is not greater than zero
    ✔ Should revert when heartbeat is not greater than fast lane padding
  Set fast lane padding
    ✔ Should revert when heartbeat role not granted
    ✔ Should succeed and set fast lane padding
    ✔ Should revert when fast lane padding is greater than heartbeat
  Set fast lane fee
    ✔ Should revert when fee setter role not granted
    ✔ Should succeed and set fast lane fee
  Set activity monitor
    ✔ Should revert when monitor setter role not granted
    ✔ Should succeed and set activity monitor
  Update
    ✔ Should succeed and update state
    ✔ Should succeed and update validators (48ms)
    ✔ Should succeed and update proof verifier (43ms)
    ✔ Should succeed and update update verifier (51ms)
    ✔ Should succeed and update exit administrator (64ms)
  Pay fast lane
    ✔ Should revert when insufficient fee sent
    ✔ Should revert when block number is too low (39ms)
    ✔ Should revert when block number is too high (after regular update) (42ms)
    ✔ Should revert when block number is too high (after fast lane update) (52ms)
    ✔ Should succeed and pay fast lane (after regular update) (69ms)
    ✔ Should succeed and pay fast lane twice in the same heartbeat period (when updated in between) (56ms)
    ✔ Should revert when fast lane request already pending (79ms)
    ✔ Should succeed and complete multiple fast lane updates (1925ms)

ValidatorsRegistry
  ✔ Should deploy correctly
  Set validator
    ✔ Should revert when not owner
    ✔ Should succeed and add validator
    ✔ Should succeed and update validator
    ✔ Should succeed and add multiple validators (64ms)
    ✔ Should succeed and allow to remove validator (47ms)
  Verify update
    ✔ Should revert when invalid signer present (118ms)
    ✔ Should revert when multiple signatures from single signer present (66ms)
    ✔ Should revert when not enough signatures present
    ✔ Should succeed and verify update (40ms)
    Quorum decreasing
      ✔ succeed with 60% of quorum after 7 days of state oracle inactivity
      ✔ keeps Quorum at 2/3 if there was no update
      ✔ keeps Quorum at 2/3 for first 5 days
      ✔ keeps Quorum between 2/3 and 55% in between 5th and 7th day (166ms)
      ✔ keeps Quorum at 55% between 7th day and 6 months later (573ms)
      ✔ keeps Quorum between 55% and 33% in between 6 months from the last update and 7 days later
      ✔ keeps Quorum at 33% after 6 months and 7 days after the last update


137 passing (28s)
```

Code Coverage

Code coverage is generally high, although coverage can be improved for the `SFCUpdateVerifier` contract.

Update Code coverage for the new tests remains high.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	91.63	78.57	94.29	91.07	
ActivityMonitor.sol	100	50	100	100	
Bridge.sol	96	76.67	90	96.88	107
DirectExitAdministrator.sol	100	87.5	100	100	
MPTProofVerifier.sol	92.86	60	100	93.33	44
MessageBus.sol	100	75	100	100	
MintedERC20.sol	100	75	100	100	
SFCUpdateVerifier.sol	48.15	33.33	62.5	48.48	... 72,73,76,78
StateOracle.sol	100	83.33	100	100	
TokenDeposit.sol	97.78	80	100	98.18	125
TokenPairs.sol	100	93.75	100	100	
UpdateManager.sol	100	86.96	100	100	
ValidatorsRegistry.sol	96	81.25	100	90.77	... 3,54,55,106
contracts/interfaces/	100	100	100	100	
IActivityMonitor.sol	100	100	100	100	
IMintedBurnableERC20.sol	100	100	100	100	
IProofVerifier.sol	100	100	100	100	
IProvingContract.sol	100	100	100	100	
ISFC.sol	100	100	100	100	
IStateOracle.sol	100	100	100	100	
ITokenDeposit.sol	100	100	100	100	
ITokenPairs.sol	100	100	100	100	
IUpdateVerifier.sol	100	100	100	100	
contracts/tests/	100	50	100	100	
SFCMock.sol	100	50	100	100	
contracts/tests/erc20/	66.67	100	66.67	66.67	
MissingReturns.sol	100	100	100	100	
NoRevert.sol	100	100	100	100	
ReturnsFalse.sol	100	100	100	100	
Standard.sol	66.67	100	66.67	66.67	16
TransferFee.sol	100	100	100	100	
All files	91.87	76.79	93.75	91.77	

Changelog

- 2024-10-01 - Initial report
- 2024-10-17 - Final Report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication or other making available of the report to you by Quantstamp.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites&asp; owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or or any open source or third-party software, code, libraries, materials, or information to, to, called by, referenced by or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

© 2025 – Quantstamp, Inc.