CERTIK

Security Assessment

Axie Infinity - Audit

Jun 21st, 2022

Table of Contents

Summary

This report has been prepared for Ronin Network to discover issues and vulnerabilities in the source code of the Axie Infinity - Audit project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	Axie Infinity - Audit
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/axieinfinity/ronin-smart-contracts-v2
Commit	• abe18fe7c333657297fa29409025dbb54852d204 • 90dad8afb431c6dc4f3d1a6aaffd0f12f72c825c

Audit Summary

Delivery Date	Jun 21, 2022 UTC
Audit Methodology	Static Analysis, Manual Review

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Mitigated	Partially Resolved	Resolved
● Critical	0	0	0	0	0	0	0
● Major	2	0	0	2	0	0	0
● Medium	5	0	0	2	0	2	1
● Minor	4	0	0	2	0	1	1
● Optimization	1	0	0	0	0	0	1
● Informational	7	0	0	5	0	0	2
● Discussion	0	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
GAC	common/GovernanceAdmin.sol	bf659cbce26ad6bb67096eef08ce52e88f81d3bc37e20e962c68a00307f99ed1
RVC	common/RoninValidator.sol	4e01a0c67b012ab2a2a188eaca29b4ad9564fa1ff47fd0d9ce5d57d6b506822b
GGC	extensions/governance/GatewayGovernance.sol	32ca3070eeea8c5a8605e74340567f231c44cc207b79cee4b3c3ab4279fdec3b
GPG	extensions/governance/GlobalProposalGovernance.sol	db88232a64d418b8815068cd2225cbe1a427c8209c4e0c23b5d71b3e86914c0a
GCK	extensions/governance/Governance.sol	018503aeb6f544f81cbe17f5411212891999eea5771ced2934f79ff74996125b
PGC	extensions/governance/ProposalGovernance.sol	12bd0a2c0214a6018b4354f8843b92b05891e0c44c6260c450edfd76b02ce0e8
GVC	extensions/GatewayV2.sol	ad15c1d1e9af2d7d44fd3f79a2490201bf0fca4bb5e574a762553ec3a964cb6b
HPA	extensions/HasProxyAdmin.sol	24aef138712d0d2f8d18da3ac5fd2b873d10ce60eb845ffdcb0768fb7b452580
MWC	extensions/MinimumWithdrawal.sol	cb70c81c0e18125d236bf4a31352f7092abaa5b47d5f46e22c25f52bd1593fc8
TUP	extensions/TransparentUpgradeableProxyV2.sol	56f02515eac98350739f670a0a9a28f3974431d198f55db0bc637cb793a0c127
WLC	extensions/WithdrawalLimitation.sol	67c340d0b6f6ba83c2a5f320884674d6a3e0bb4eeaf4029310d1fe38583aece6
IER	interfaces/IERC20Mintable.sol	4795937cb211a75c6c525b06508e7f57d73e7bbc24d6b4e36cb3d26b2c19aea5
IEC	interfaces/IERC721Mintable.sol	a93c33101084deef5fca264a4dff73f05cce8ca33519648d2128596b62946214
IQC	interfaces/IQuorum.sol	5e12f2f1134550dfe70bc1f2503ff11fb9181c6b874f29bc262393e01c5daa12
IWE	interfaces/IWETH.sol	688a73efabe2972c17647f4daba15e1e55d59aa9a5d267cf7c1f2aca26dddfda

ID	File	SHA256 Checksum
IWV	interfaces/IWeightedValidator.sol	a6553f833882c27e2c71ac1e3925185c891eefc3b458de947f89eabdf054aa3f
MTC	interfaces/MappedTokenConsumer.sol	1beb2fdc968753fce0e0878b230bbc2db818c71d3b99cd56b4224f8ef2f5f4f3
SCC	interfaces/SignatureConsumer.sol	f9f8a78e55b9de1c5627e5be695e004c7bc29a3e387358e5a25d430550791052
BCK	library/Ballot.sol	ebaac64bd83794d8051c5e3067c04320a24055e14dd5454a17dba7cb117ad23b
GPC	library/GlobalProposal.sol	40e1dc63905c17c856174961a9c433915aebf8cae41eb1b2491b3be2ca27d980
PCK	library/Proposal.sol	20983d4eb425c6a75f50df57faa2f48cd7c253c7960d7a271db5048cf287e228
TCK	library/Token.sol	18c9118afe457001db1288016bf862af194bd44b27be0babe0a6bb01e82b6704
TCP	library/Transfer.sol	86ba568b7e2d0c28b57e319423db1291fa5409a0408e755978f78fd6ebdceb53
IMG	mainchain/IMainchainGatewayV2.sol	41e04dbcfd5032a3d4db1f71c12b3360d03ba94dc0400eae4df3b2d55196cbe9
MGV	mainchain/MainchainGatewayV2.sol	ca514918cfa5cf5170476610685d349c8b93fdadba51d9ec9133456cd02642ef
BMC	migration/BridgeMigration.sol	0dd7da666b3242839f0d53d4ec143bac7531669bd3920ed648425554f9be792f
MER	mocks/MockERC721.sol	8ab73c3fe72a92f2bdd016f3a5c51de87db04e0da7d8fd82391bc1a2628654d2
MGC	mocks/MockGatewayV2.sol	49f5985faaab611c67c0e5e40231ba3dcbb8604191c5ea93499e2d064a20d44e
IRG	ronin/IRoninGatewayV2.sol	6e1474b6b1084326bc85cd028679e95b408db36a1199cf3766451d428807d371
RGV	ronin/RoninGatewayV2.sol	c73070b6c018fd92167b02b2b4c38c7226b13c1d9469ecfb88e43ef01e00f05b
WLK	extensions/WithdrawalLimitation.sol	78f9a9a781cd296df0cf163d4b38209c8dec8418a3ecb25d72a7b32413d5762c
MGK	mainchain/MainchainGatewayV2.sol	0c1e636e34db47fe49758878680cd974292b0c65bf1493c8dbdd67d8113b49e1

Review Notes

Overview

Ronin Network has created a set of contracts that allow bridging assets and governance proposals between Ronin Network and other EVM blockchains.

External Dependencies

The scope of the audit treats third-party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets.

Ronin Network relies on :

- A Frontend server : So users can ask for the bridge of their assets;
- Bridge relayers : Servers to relay proposals on other chains;
- Ronin Validators : Who also validate deposits and withdrawals;
- Potentially, some other servers : Who listen events, and trigger actions upon event reception;
- Ronin Ethereum Sidechain ecosystem.

Those elements are critical to Ronin Bridge's functioning and security, and need to be audited.

Some other smart contracts dependancies exist:

- `ECDSA`, `AccessControlEnumerable`, `IQuorum`, `IWeightedValidator` for the `GovernanceAdmin` contract;
- `Initializable`, `Strings`, `StorageSlot` for the `RoninValidator` contract;
- `Strings` for the `GatewayGovernance` contract;
- `Strings`, `SignatureConsumer` for the `Governance` contract;
- `Pausable`, `IQuorum`, `IWeightedValidator` for the `GatewayV2` contract;
- `StorageSlot` for the `HasProxyAdmin` contract;
- `TransparentUpgradeableProxy` for the `TransparentUpgradeableProxyV2` contract;
- `ECDSA` for the `GlobalProposal` contract;
- `Address` for the `Proposal` contract;
- `IERC20`, `IERC721`, `Strings`, `IWETH` for the `Token` contract;
- `ECDSA`, `IERC20`, `Strings` for the `Transfer` contract;
- `AccessControlEnumerable`, `Initializable` for the `MainchainGatewayV2` contract;
- `Ownable`, `IERC20` for the `BridgeMigration` contract;
- `AccessControlEnumerable`, `Initializable`, `IERC20Mintable`, `IERC721Mintable` for the `RoninGatewayV2` contract.

We assume these vulnerable actors and implement proper logic to collaborate with the current project.

Privileged Roles

The following roles are adopted to enforce the access control:

- Role `_owner` is adopted to update configurations of the contract `BridgeMigration`,
- Role `RELAYER_ROLE` is adopted to update configurations of the contract `GovernanceAdmin`,
- Role `DEFAULT_ADMIN_ROLE` is adopted to update configurations of the contract `GovernanceAdmin`,
- Role `onlyGovernor` is adopted to update configurations of the contract `GovernanceAdmin`,
- Role `onlySelfCall` is adopted to update configurations of the contract `GovernanceAdmin`,
- Role `onlyAdmin` is adopted to update configurations of the contract `RoninValidator`,
- Role `onlyAdmin` is adopted to update configurations of the contract `GatewayV2`,
- Role `onlyAdmin` is adopted to update configurations of the contract `MinimumWithdrawal`,
- Role `ifAdmin` is adopted to update configurations of the contract `TransparentUpgradeableProxyV2`,
- Role `onlyAdmin` is adopted to update configurations of the contract `WithdrawalLimitation`,
- Role `onlyAdmin` is adopted to update configurations of the contract `MainchainGatewayV2`,
- Role `WITHDRAWAL_UNLOCKER_ROLE` is adopted to update configurations of the contract `MainchainGatewayV2`,
- Role `onlyAdmin` is adopted to update configurations of the contract `MainchainGatewayV2`,
- Role `WITHDRAWAL_MIGRATOR` is adopted to update configurations of the contract `RoninGatewayV2`.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of `Timelock` contract.

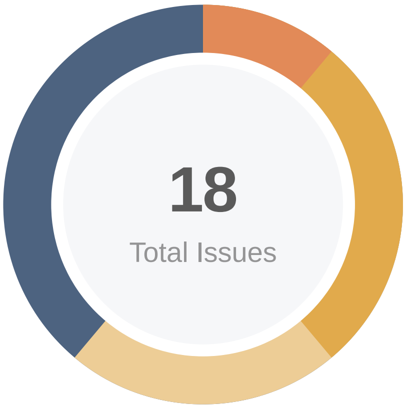
Project Goals

The engagement was scoped to provide a security assessment of the Ronin Network bridge. Specifically, we sought to verify the following non-exhaustive list of potential attack vectors:

- C6.1: Verify that bridge requires all necessary values to be included in the message and signed: chain ids, receiver, amount, nonce.
- C6.2: Verify that used signatures are invalidated to protect bridge from replay attacks.
- C6.3: Verify that message hash generation algorithm is resistant to collision attacks.
- C6.4: Verify that bridge includes source and destination chains identifiers in the signed message and correctly verifies them.
- C6.5: Verify that bridge does not allow spoofing chain identifiers.

- C6.6: Verify that bridge uses a nonce parameter to allow the same operation (the same sender, receiver and amount) to be executed multiple times.
- C6.7: Verify signed message cannot be used in a different context (use domain separator from EIP-712).

Findings



🟥 **Critical**	**0** (0.00%)
🟧 **Major**	**2** (11.11%)
🟨 **Medium**	**5** (27.78%)
🟨 **Minor**	**4** (22.22%)
🟦 **Informational**	**7** (38.89%)
🟩 **Discussion**	**0** (0.00%)

ID	Title	Category	Severity	Status
GLOBAL-01	Centralization Related Risks	**Centralization / Privilege**	● **Major**	Acknowledged
GLOBAL-02	External Dependencies	Logical Issue	● Medium	Acknowledged
GLOBAL-03	No Storage Gap In Logical Contracts	Logical Issue	● Medium	Partially Resolved
GLOBAL-04	No Delay In Governance Tasks	Logical Issue	● Major	Acknowledged
GLOBAL-05	Unlocked Pragma	Language Specific	● Informational	Acknowledged
CKP-01	Potential Lack Of Liquidity	Logical Issue	● Medium	Partially Resolved
CKP-02	`_minimumVoteWeight()` Can Be Set To A Low Value	Logical Issue	● Minor	Partially Resolved
CKP-03	Incompatibility With Deflationary Tokens	Volatile Code	● Minor	Acknowledged
GAC-01	Relayers Can Execute Any Proposal In A Certain Condition	Logical Issue	● Medium	Resolved
GAC-02	No Check That Address Is An Actual Contract	Logical Issue	● Minor	Resolved
GCK-01	Inconsistency With Comments	Logical Issue	● Informational	Resolved
MGV-01	Validators Could Be Too Powerful	Logical Issue	● Medium	Acknowledged
MGV-02	Using Of Default Value	Logical Issue	● Informational	Acknowledged

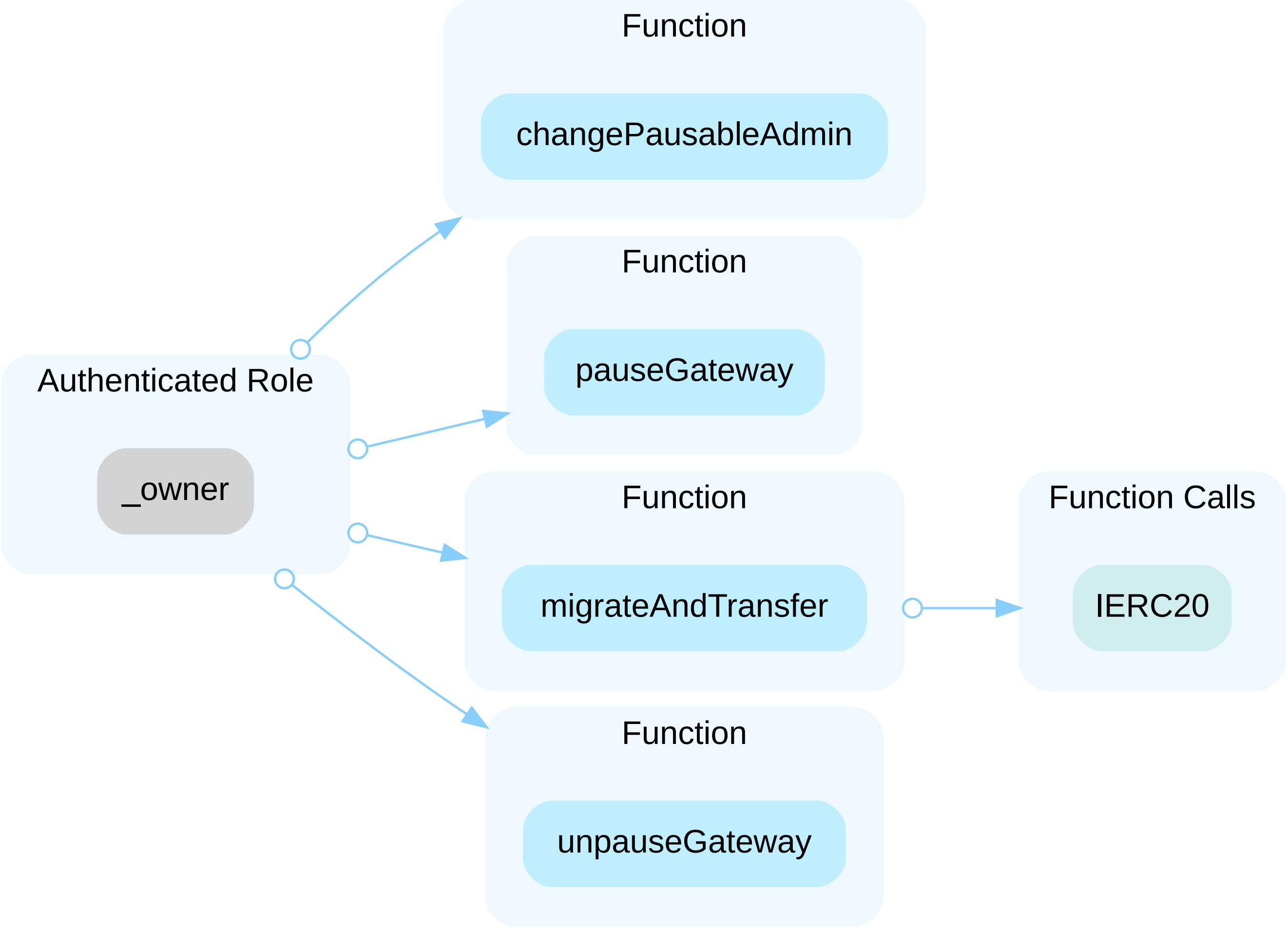
ID	Title	Category	Severity	Status
TCK-01	Potential Re-entrancy On `handleAssetTransfer()`	Logical Issue	● Informational	ⓘ Acknowledged
TCK-02	Completion Of If-else Branch	Volatile Code	● Informational	✓ Resolved
TUP-01	Design Violation	Inconsistency	● Informational	ⓘ Acknowledged
WLC-01	Inappropriate Upper Limits For Fees	Logical Issue	● Minor	ⓘ Acknowledged
WLK-01	Questions About Tiers Model	Inconsistency	● Informational	ⓘ Acknowledged

GLOBAL-01 | Centralization Related Risks

Category	Severity	Location	Status
Centralization / Privilege	● **Major**		ⓘ Acknowledged

Description

In the contract `BridgeMigration` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and call the `migrateAndTransfer()` function to steal funds.



In the contract `GovernanceAdmin` the role `RELAYER_ROLE` has authority over the functions below:

- `relayProposal()` : Relay a proposal and votes on another chain;
- `relayGlobalProposal()` : Relay a "Global" proposal and votes on another chain.

Any compromise to the `RELAYER_ROLE` account may allow the hacker to take advantage of this authority and attempt to relay false proposals on the impacted chain.

In the contract `GovernanceAdmin` the role `DEFAULT_ADMIN_ROLE` has a high level of authority over the contract and can add/modify roles (variable `_roleSetter`)

Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and take over important roles of the contract.

In the contract `GovernanceAdmin` the role `onlyGovernor` has authority over the functions below:

- `propose()` : Propose a Proposal;
- `proposeGlobal()` : Propose a "Global" Proposal;
- `proposeProposalStructAndCastVotes()` : Propose a Proposal and cast votes;
- `proposeGlobalProposalStructAndCastVotes()` : Propose a "Global" Proposal and cast votes.

Any compromise to the `onlyGovernor` account may allow the hacker to take advantage of this authority and create fake proposals. The attacker would however need the votes from the validator.

In the contract `GovernanceAdmin` the role `onlySelfCall` has authority over the functions below:

- `changeProxyAdmin()` : Change the administrator of the proxy contract;
- `setValidatorContract()` : Change the address of the Validator contract;
- `setGatewayContract()` : Change the address of the Gateway contract.

This access control is particular, since it corresponds to the contract calling itself. If an attacker can create proposals and cast them, he could potentially trigger the functions above and take control over the whole contract, since he could modify the `ProxyAdmin`, the `Validator` contract, and the `Gateway` contract.

In the contract `RoninValidator` the role `onlyAdmin` has authority over the functions below:

- `addValidators()` : Add Ronin validators;
- `updateValidators()` : Update Ronin validators;
- `removeValidators()` : Remove Ronin validators;
- `setThreshold()` : Configure num/denum threshold.

Any compromise to the `onlyAdmin` account may allow the hacker to take advantage of this authority and add his own validators, which could later be used to attempt to vote on proposals.

In the contract `GatewayV2` the role `onlyAdmin` has authority over the functions below:

- `setThreshold()` : Configure num/denum threshold;
- `pause()/unpause()` : Pause/Unpause the contract;
- `setValidatorContract()` : Change the address of the Validator contract.

Any compromise to the `onlyAdmin` account may allow the hacker to take advantage of this authority and add his own validators (by modifying the `Validator` contract), which could later be used to attempt to vote on proposals.

In the contract `MinimumWithdrawal` the role `onlyAdmin` has authority over the functions below:

- `setMinimumThresholds()` : Sets the minimum thresholds to withdraw.

Any compromise to the `onlyAdmin` account may allow the hacker to take advantage of this authority and increase the minimum threshold to withdraw to bypass current limitations.

In the contract `TransparentUpgradeableProxyV2` the role `ifAdmin` has authority over the functions below:

- `functionDelegateCall()` : Proxy admin can call contract implementation.

Any compromise to the `onlyAdmin` account may allow the hacker to take advantage of this authority and attack the implementation contract with the role of the proxy Administrator.

In the contract `WithdrawalLimitation` the role `onlyAdmin` has authority over the functions below:

- `setFullSigsThresholds()` : Sets the thresholds for withdrawals that requires all validator signatures;
- `setLockedThresholds()` : Sets the amount thresholds to lock withdrawal;
- `setUnlockFeePercentages()` : Sets fee percentages to unlock withdrawal;
- `setDailyWithdrawalLimits()` : Sets daily limit amounts for the withdrawals.

Any compromise to the `onlyAdmin` account may allow the hacker to take advantage of this authority and modify withdrawals configurations.

In the contract `MainchainGatewayV2` the role `onlyAdmin` has authority over the functions below:

- `setWrappedNativeTokenContract()` : Modify the `wrappedNativeToken` state variable;
- `mapTokens()` : Maps current chain assets with Ronin assets;
- `mapTokensAndThresholds()` : Maps current chain assets with Ronin assets, and perform `setFullSigsThresholds()`, `setLockedThresholds()`, `setUnlockFeePercentages()`, `setDailyWithdrawalLimits()`.

Any compromise to the `onlyAdmin` account may allow the hacker to take advantage of this authority and cause a Denial Of Service by modifying the wrapped token or the tokens mappings.

In the contract `MainchainGatewayV2` the role `WITHDRAWAL_UNLOCKER_ROLE` has authority over the functions below:

- `unlockWithdrawal()` : Unlock withdrawals.

Any compromise to the `onlyAdmin` account may allow the hacker to take advantage of this authority and steal tokens by calling this function.

In the contract `MainchainGatewayV2` the role `onlyAdmin` has authority over the functions below:

- `mapTokens()` : Maps Ronin assets with other chain's assets.

Any compromise to the `onlyAdmin` account may allow the hacker to take advantage of this authority and cause a Denial Of Service by modifying the tokens mappings.

In the contract `RoninGatewayV2` the role `WITHDRAWAL_MIGRATOR` has authority over the functions below:

- `migrateWithdrawals()` : Migrate withdrawals;

Any compromise to the `WITHDRAWAL_MIGRATOR` account may allow the hacker to take advantage of this authority and steal tokens by calling this function.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND

- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[Ronin]:
The `BridgeMigration` is used only once to migrate the existing token in the old gateway to the new gateway (on Ethereum). Firstly we will deploy it then we ask the validators to provide us the necessary signatures to move the fund.

The `RELAYER_ROLE` in `GovernanceAdmin` can only relay proposal with enough governance signatures, so we think the risk is small.

GLOBAL-02 | External Dependencies

Category	Severity	Location	Status
Logical Issue	● Medium		ⓘ Acknowledged

Description

The Ronin bridge relies on external parties to function correctly.

For instance, for the bridge to work, some servers must exist, that will be in charge of capturing events, and triggering actions associated with private keys (cf Bridge Workers/Relayers).

In particular, Ronin Network relies on :

- A Frontend server : So users can ask for the bridge of their assets;
- Bridge relayers : Servers to relay proposals on other chains;
- Ronin Validators : Who also validate deposits and withdrawals;
- Potentially, some other servers : Who listen events, and trigger actions upon event reception;
- Ronin Ethereum Sidechain ecosystem.

Those elements are critical to Ronin Bridge's functioning and security, and need to be audited.

Some other smart contracts dependancies exist:

- `ECDSA`, `AccessControlEnumerable`, `IQuorum`, `IWeightedValidator` for the `GovernanceAdmin` contract;
- `Initializable`, `Strings`, `StorageSlot` for the `RoninValidator` contract;
- `Strings` for the `GatewayGovernance` contract;
- `Strings`, `SignatureConsumer` for the `Governance` contract;
- `Pausable`, `IQuorum`, `IWeightedValidator` for the `GatewayV2` contract;
- `StorageSlot` for the `HasProxyAdmin` contract;
- `TransparentUpgradeableProxy` for the `TransparentUpgradeableProxyV2` contract;
- `ECDSA` for the `GlobalProposal` contract;
- `Address` for the `Proposal` contract;
- `IERC20`, `IERC721`, `Strings`, `IWETH` for the `Token` contract;
- `ECDSA`, `IERC20`, `Strings` for the `Transfer` contract;
- `AccessControlEnumerable`, `Initializable` for the `MainchainGatewayV2` contract;
- `Ownable`, `IERC20` for the `BridgeMigration` contract;

- `AccessControlEnumerable`, `Initializable`, `IERC20Mintable`, `IERC721Mintable` for the `RoninGatewayV2` contract.

The above contract dependencies are considered secure in the context of the current audit.

Recommendation

It is recommended to audit third-party dependencies.
For the servers exposed on the Internet, it is recommended to perform a pentest :

- In `Black box` mode, to identify vulnerabilities that can be seen by an external attacker;
- In `Gray box` mode, to identify what a malicious user could do.

Alleviation

[Ronin]:

The team acknowledged this issue and decided not to change the current codebase.

GLOBAL-03 | No Storage Gap In Logical Contracts

Category	Severity	Location	Status
Logical Issue	● Medium		Partially Resolved

Description

Ronin has implemented proxyfiable contracts. Those contracts inherit from the following contracts (Interfaces are not mentionned) :

- `RoninValidator` : Inherits from `Initializable`, `HasProxyAdmin`.
- `RoninGatewayV2` : Inherits from `GatewayV2`, `GatewayGovernance`, `Initializable`, `MinimumWithdrawal`, `AccessControlEnumerable`;
- `MainchainGatewayV2` : Inherits from `WithdrawalLimitation`, `Initializable`, `AccessControlEnumerable`.

Some of those contracts do not implement a storage gap:

- `HasProxyAdmin`;
- `GatewayV2`;
- `MinimumWithdrawal`;
- `AccessControlEnumerable`;
- `WithdrawalLimitation`.

Because of this, if the logical contract is upgraded to a new version, and if variables are added in the dependencies, storage conflict could occur in the proxyfiable contracts, causing negative consequences over the functioning of the `VolumeWars` contract.

Recommendation

The logic contracts need to implement a storage gap, as per OpenZeppelin recommendation:

```
uint256[50] private ______gap;
```

For `AccessControlEnumerable`, an upgradeable version from OpenZeppelin is available.

Alleviation

[Ronin]:

The team partially resolved this issue by adding a storage gap in the contracts `GatewayV2`, `MinimumWithdrawal` and `WithdrawalLimitation` in the PR 23. For `HasProxyAdmin` and `AccessControlEnumerable` contracts, the team won't make any change for the current version.

GLOBAL-04 | No Delay In Governance Tasks

Category	Severity	Location	Status
Logical Issue	● Major		ⓘ Acknowledged

Description

According to the documentation, Governors are users, and those users will act by providing signatures when interacting with `GovernanceAdmin` contract.

Considering the users' behavior is unpredictable, it is recommended to introduce a certain time of delay when performing governance actions.

For example, in the case that the private keys of multiple governors are compromised, attackers could immediately perform the following actions to execute malicious proposals:

1. Create a Malicious Propocal (or Global Proposal),
2. Cast the Vote,
3. Execute a Malicious proposal.

This could have detrimental consequences over Ronin bridge.

Recommendation

It is recommended to introduce delays in Governance actions, so the bridge cannot be compromised in a matter of a very short period of time if Governance accounts were to be compromised. Also, it gives the time for the Ronin Network team to perform responses (e.g., pausing the main functionality) before executing malicious proposals.

Alleviation

[Ronin] :
Currently, we are asking the validators to store the governor account in a hardware wallets so it helps minimize the risk of getting compromised.

To fully mitigate this issue we will need to carefully design the strategy when the abnormal events happen, which would take too much time right now. We decided to leave it open for future upgrade of the system.

GLOBAL-05 | Unlocked Pragma

Category	Severity	Location	Status
Language Specific	● Informational		ⓘ Acknowledged

Description

Contracts should be deployed using the same compiler version/flags with which they have been tested. Locking the pragma (e.g. by not using ^ in pragma solidity 0.8.0) ensures that contracts do not accidentally get deployed using an older compiler version with unfixed bugs.

Reference: [SWC-103] https://swcregistry.io/docs/SWC-103

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.8.0` the contract should contain the following line:

```
pragma solidity 0.8.0;
```

Alleviation

[Ronin]:

The team acknowledged this issue and decided not to change the current codebase.

CKP-01 | Potential Lack Of Liquidity

Category	Severity	Location	Status
Logical Issue	● Medium	library/Token.sol (audit): 145~177; mainchain/MainchainGatewayV2.sol (audit): 148~151, 298; ronin/RoninGatewayV2.sol (audit): 325	Partially Resolved

Description

Both the `MainchainGatewayV2` and `RoninGatewayV2` contracts, upon Deposits and Withdrawals, use the `handleAssetTransfer()` to forward the funds to the final user.

The transfers might fail if there is not enough tokens in the contract. For instance, if there is not enough `_wrappedNativeToken` in the contract, the transaction will revert in the `transfer()` function:

```
function transfer(
  Info memory _info,
  address _to,
  address _token
) internal {
  bool _success;
  if (_info.erc == Standard.ERC20) {
    _success = tryTransferERC20(_token, _to, _info.quantity);
  } else if (_info.erc == Standard.ERC721) {
    _success = tryTransferERC721(_token, _to, _info.id);
  }

  if (!_success) {
    revert(
      string(
        abi.encodePacked(
          "Token: could not transfer ",
          toString(_info),
          " to ",
          Strings.toHexString(uint160(_to), 20),
          " token ",
          Strings.toHexString(uint160(_token), 20)
        )
      )
    );
  }
}
```

However, no event is emitted, so the Ronin network might not be alerted of this problem.

Additionally, if the `submitWithdrawal()` function on the Mainchain side reverted, there is no function on the Ronin Chain side to withdraw the locked funds in the Ronin Gateway contract. Therefore, the user might lose their funds forever.

Recommendation

The auditors would like to know how this edge case is dealt with by Ronin.

Alleviation

[Ronin]:
If there is not enough liquidity there is a bigger issue going on, and we will need to address it via governance process (e.g. Upgrade contracts, calling for signatures to withdraw the remaining tokens in the bridge).

CKP-02 | `_minimumVoteWeight()` Can Be Set To A Low Value

Category	Severity	Location	Status
Logical Issue	Minor	common/RoninValidator.sol (audit): 183~194; extensions/GatewayV2.sol (audit): 96~107	Partially Resolved

Description

When a withdrawal or a deposit operation is submitted, validators agree to validate an operation. For example, in `_submitWithdrawal()` function, when enough validators have validated the operation with their signatures, tokens are sent to users.

File `MainchainGatewayV2`

```
(...)
  _weight += _validatorContract.getValidatorWeight(_signer);
  if (_weight >= _minimumVoteWeight) {
   _passed = true;
   break;
  }
 }
 require(_passed, "MainchainGatewayV2: query for insufficient vote weight");
 withdrawalHash[_id] = _receiptHash;
}
(...)
  _recordWithdrawal(_tokenAddr, _quantity);
  _receipt.info.handleAssetTransfer(payable(_receipt.mainchain.addr), _tokenAddr, wrappedNativeToken);
  emit Withdrew(_receiptHash, _receipt);
```

This is intended in order to ensure that multiple validators vote on the same proposal, and one validator should usually not be able to pass a vote on his own.

The `_minimumVoteWeight` mentioned above is computed as follows:

```
(...)
 function _computeMinVoteWeight(
  Token.Standard _erc,
  address _token,
  uint256 _quantity,
  IWeightedValidator _validatorContract
 ) internal virtual returns (uint256 _weight, bool _locked) {
  uint256 _totalWeights = _validatorContract.totalWeights();
```

```
    _weight = _minimumVoteWeight(_totalWeights);
(...)
```

The `_weight` is computed as follows:

```
function _minimumVoteWeight(uint256 _totalWeight) internal view virtual returns (uint256) {
  return (_num * _totalWeight + _denom - 1) / _denom;
}
```

However, when `_num` and `_denom` are configured, the only restriction is :

```
  function _setThreshold(uint256 _numerator, uint256 _denominator)
    internal
    virtual
    returns (uint256 _previousNum, uint256 _previousDenom)
  {
    require(_numerator <= _denominator, "GatewayV2: invalid threshold");
```

`_denom` can be very large compared to `_num`. To take a concrete example, imagine that:

- 9 validators exist,
- Each validator has a weight of 100 (`_totalWeights` = 900),
- `_num` is 1,
- `_denom` is 1*10e18.

This kind of configuration would put `minimumVoteWeight()` to:

$$minimumVoteWeight() = (\_num * \_totalWeights + \_denom - 1)/\_denom$$
$$minimumVoteWeight() = (900 + 1 * 10e18)/(1 * 10e18)$$
$$minimumVoteWeight() = 1$$

This means that **any validator could validate any proposal**.

The value `1` has been validated with the following PoC:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract numDenom {
  uint256 public _num;
  uint256 public _denom;
  uint256 public _totalWeights;
  constructor(){
    _denom = 1 ether;
    _num=1;
    _totalWeights=900;
  }

  function _minimumVoteWeight() public view virtual returns (uint256) {
    return (_num * _totalWeights + _denom - 1) / _denom;
  }
}
```

Recommendation

It is recommended to add further validation upon `_denom` and `_num`to avoid any situation where a validator could pass a proposal by itself.

Alleviation

[Ronin]:

Any changes in the vote weight requirements will need to go through the voting process, so the risk is minimized.

CKP-03 | Incompatibility With Deflationary Tokens

Category	Severity	Location	Status
Volatile Code	● Minor	mainchain/MainchainGatewayV2.sol (audit): 326; ronin/RoninGatewayV2.sol (audit): 359	ⓘ Acknowledged

Description

When transferring standard ERC20 deflationary tokens, the input amount may not be equal to the received amount due to the charged transaction fee. As a result, an inconsistency in the amount will occur and the transaction may fail due to the validation checks.

For example, if a user deposit deflationary tokens (with a 10% transaction fee) into mainchain gateway contract, only 90 tokens actually arrive in the contract. However, the user can still withdraw 100 tokens (before fees) from the contract of the ronin side, which causes a lose 10 tokens in such a transaction.

Reference: https://thoreum-finance.medium.com/what-exploit-happened-today-for-gocerberus-and-garuda-also-for-lokum-ybear-piggy-caramelswap-3943ee23a39f

Recommendation

We advise the client to add necessary mitigation mechanisms to keep track of accurate balances if there is a need to support deflationary tokens.

Alleviation

[Ronin]:
The team acknowledged this issue and decided not to change the current codebase.

GAC-01 | Relayers Can Execute Any Proposal In A Certain Condition

Category	Severity	Location	Status
Logical Issue	● Medium	common/GovernanceAdmin.sol (audit)	✓ Resolved

Description

A Relayer can relay a proposal (creation of proposal and forward of signatures) on a specific chain. By calling `relayProposal()`, relayers can :

1. Create the proposal, coming from another chain;
2. Cast a vote for the proposal, by passing signed messages from validators; If the vote is passed and marked as executable, a `call()` will be performed.

The issue is that, if `_minimumForVoteWeight` is set to 0, relayers might be able to pass proposal with a fake signature, because `_totalForVoteWeight` (0) would be equal to `_minimumForVoteWeight` (0):

```
uint256 _minimumForVoteWeight = _getMinimumVoteWeight();
uint256 _totalForVoteWeight = _getWeights(_forVoteSigners);
    if (_totalForVoteWeight >= _minimumForVoteWeight) {
      _vote.status = VoteStatus.Approved;
(...)
  _proposal.execute();
```

To abuse this behavior, a malicious relayer could :

1. Create a malicious proposal, marking it as `executable`;
2. Sign a vote for this proposal with his own address;
3. Call `relayProposal()` to execute his proposal.

Recommendation

It is recommended to add a check that :

- `_totalForVoteWeight` is > 0;
- `_totalAgainstVoteWeight` is > 0;

This kind of check is already performed in the function `_castVotesBySignatures()` :

```
uint256 _weight = _getWeight(_signer);
if (_weight > 0) {
```

Alleviation

[Ronin]:

The team resolved this issue by adding the missing checks, in the PR 23.

GAC-02 | No Check That Address Is An Actual Contract

Category	Severity	Location	Status
Logical Issue	● Minor	common/GovernanceAdmin.sol (audit): 311~312, 322	✓ Resolved

Description

The `_setValidatorContract()` and `_setGatewayContract()` modify the value of contracts addresses, but do not validate if those addresses are valid contracts. Administrators could, by mistake, put an address not related to a contract.

Recommendation

It is recommended to perform checks to ensure that the modified variables correspond to contract. This could be done through the following check:

```
modifier isContract() {
    require((_isContract(msg.sender)), "only contracts are allowed");
    _;
}

function _isContract(address addr) internal view returns (bool) {
    uint256 size;
    assembly {
        size := extcodesize(addr)
    }
    return size > 0;
}
```

Alleviation

[Ronin]:

The team resolved this issue by adding a verification on the `code.length`, in the PR 23.

GCK-01 | Inconsistency With Comments

Category	Severity	Location	Status
Logical Issue	● Informational	extensions/governance/Governance.sol (audit): 252	Resolved

Description

The comment in the `Governance`contract, for the `_castVotesBySignatures()` function, states:

```
* @notice This method does not verify the proposal hash with the vote hash. Please consider checking it before.
```

When looking at the 4 functions calling `_castVotesBySignatures()`, 2 of them do not seem to perform the check:

- `_castGlobalProposalBySignatures()` : OK;
- `_castProposalBySignatures() : OK;
- `_proposeGlobalProposalStructAndCastVotes()` : **KO**;
- `_proposeProposalStructAndCastVotes()` : **KO**.

Recommendation

The auditors would like to know if there is a reason for this difference of behavior. If so, it might be opportune to modify the aforementioned comment.

Alleviation

[Ronin]:

The team acknowledged this is by design. The first two functions vote for existing proposals so they need to check the hash to make sure. The last two functions create a new proposal and cast the vote right away by the creator so you don't need to check the hash.

MGV-01 | Validators Could Be Too Powerful

Category	Severity	Location	Status
Logical Issue	● Medium	mainchain/MainchainGatewayV2.sol (audit): 122	ⓘ Acknowledged

Description

The function `submitWithdrawal()` verifies the signatures from validators. When all signatures are verified and when the threshold is met, assets will be transferred to the user specified in the `_receipt` parameter.

```
122    function submitWithdrawal(Transfer.Receipt calldata _receipt, Signature[] calldata _signatures)
123      external
124      virtual
125      whenNotPaused
126      returns (bool _locked)
127    {
128      return _submitWithdrawal(_receipt, _signatures);
129    }
```

The concern is, if the attacker exploited the private keys of the validators, the attacker can spoof the `receipt` and signatures, thus stealing the funds within the contract.

As the validator's logic is unknown, we propose a potential workaround to add a restriction on the caller of `submitWithdrawal()` and separate the caller with validators. The caller could be a server that calls `submitWithdrawal()` after having received the deposit events (`DepositRequested`).

In this way, by adding another layer of verification, even if the validators' private keys are compromised, the attacker cannot steal funds because the attacker needs to spoof a deposit event on the other chain.

Recommendation

The above proposal serves as a discussion purpose. We would also like to learn about how the Ronin network ensures the validators' private keys are safe.

Alleviation

[Ronin]:

The team agreed with this suggestion, and will work on it in a later stage.

MGV-02 | Using Of Default Value

Category	Severity	Location	Status
Logical Issue	● Informational	mainchain/MainchainGatewayV2.sol (audit): 397	ⓘ Acknowledged

Description

When request a deposit with a fallback function, the `info` varaible was filled with default values, meaning `info.erc` is `ERC20` and `info.id` is 0.

```
function _fallback() internal virtual whenNotPaused {
  if (msg.sender != address(wrappedNativeToken)) {
    Transfer.Request memory _request;
    _request.recipientAddr = msg.sender;
    _request.info.quantity = msg.value;
    _requestDepositFor(_request, _request.recipientAddr);
  }
}
```

Recommendation

Consider upgradeable feature of the project, we recommend explicitly assign values to those varaibles instead of using the default value.

Alleviation

[Ronin]:

The team acknowledged this issue and decided not to change the current codebase.

TCK-01 | Potential Re-entrancy On `handleAssetTransfer()`

Category	Severity	Location	Status
Logical Issue	● Informational	library/Token.sol (audit): 159~165	ⓘ Acknowledged

Description

In the `handleAssetTransfer()` function of the `Token` contract, if the `_token` is a `Token.Standard.ERC20`, the flow is as following to send the tokens:

```
} else if (_info.erc == Token.Standard.ERC20) {
uint256 _balance = IERC20(_token).balanceOf(address(this));

if (_balance < _info.quantity) {
  // bytes4(keccak256("mint(address,uint256)"))
  (_success, ) = _token.call(abi.encodeWithSelector(0x40c10f19, address(this),
_info.quantity - _balance));
  require(_success, "Token: ERC20 minting failed");
}

transfer(_info, _to, _token);
```

After analysis, it does not seem that a practical scenario is possible, in which Ronin Network funds would be at risk. The scenario below intends to describe where the issue lies.
In the hypothetical case that `_token` is a proxified and valuable ERC20 token controlled by an attacker, a re-entrancy could occur by abusing the `balanceOf()` function.

The flow is as following :

1. Attacker modifies the implementation of `_token` to modify the `balanceOf()` function, to call `handleAssetTransfer()`.
2. Attacker calls `handleAssetTransfer()`;
3. When the contract will call `IERC20(_token).balanceOf(address(this))`, the call will go to `handleAssetTransfer()`, performing the re-entrancy.

It is after the re-entrancy that the `transfer()` call is actually performed to send the tokens, making the attack possible.

Recommendation

It is recommended to apply OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the `handleAssetTransfer()` function, to prevent reentrancy attack.

Alleviation

[Ronin]:

The team acknowledged this issue and decided not to change the current codebase.

TCK-02 | Completion Of If-else Branch

Category	Severity	Location	Status
Volatile Code	● Informational	library/Token.sol (audit): 58~64, 117~121	✓ Resolved

Description

The `Token` library invokes the token transfers (via `transferFrom()` and `transfer()`). Those functions first check the token's type with an `if-else` branch. For example,

```
58      if (_info.erc == Standard.ERC20) {
59        (_success, _data) = _token.call(abi.encodeWithSelector(IERC20.transferFrom.selector, _from, _to, _info.quantity));
60        _success = _success && (_data.length == 0 || abi.decode(_data, (bool)));
61      } else if (_info.erc == Standard.ERC721) {
62        // bytes4(keccak256("transferFrom(address,address,uint256)"))
63        (_success, ) = _token.call(abi.encodeWithSelector(0x23b872dd, _from, _to, _info.id));
64      }
```

The above `if-else` branch is not completed, meaning it lacks an `else` branch to cover all the other situations. Since the current `Standard` enum only has two types, it will not cause any actual issue.

```
enum Standard {
  ERC20,
  ERC721
}
```

However, considering the upgradeable feature of the contract, if the library supports more types of tokens, it could lead to potential risk.

Recommendation

We recommend adding an else branch to cover all the possible situations. For example,

```
    if (_info.erc == Standard.ERC20) {
      ...
    } else if (_info.erc == Standard.ERC721) {
      ...
    } else {
      revert("Token: unsupported token standard");
    }
```

Alleviation

[Ronin]:

The team resolved this issue by adding a `else` branch in the PR 23.

TUP-01 | Design Violation

Category	Severity	Location	Status
Inconsistency	● Informational	extensions/TransparentUpgradeableProxyV2.sol (audit): 24~37	ⓘ Acknowledged

Description

The `TransparentUpgradeableProxy` is designed as follows:

- When `users` call the proxy, calls are forwarded to the implementation contract with `delegatecall`;
- When an `admin` calls the proxy, the call is executed on the proxy contract.

This design is meant to prevent `Proxy selector clashing` attacks.

The `TransparentUpgradeableProxyV2` contract implemented by Ronin violates this design, by allowing administrators to call the implementation contract, with the addition of the `functionDelegateCall()` function.

Recommendation

The auditors would like to understand the reason of this choice.

Alleviation

[Ronin]:
We use the **TransparentUpgradeableProxy** to mainly avoid selector clashing issues, which can cause unexpected behavior for the Bridge.

In the Ronin Bridge context, we set the **Governance Admin** contract (**GA**) as the **ProxyAdmin** of the Validator contract and the Gateway contract (which implements the **TransparentUpgradeableProxy** behind). These contracts only allow the **GA** contract to modify some critical states.
But the **TransparentUpgradeableProxy ProxyAdmin** is not allowed the to call any methods in the implementation contract; so we introduce the **TransparentUpgradeableProxyV2** that allows the **ProxyAdmin** to do it by explicitly calling the `functionDelegateCall` function.

Thanks to this function, the **GA** contract can call to Ronin Validator contract to retrieve governor addresses, and get/set thresholds despite it being the proxy admin.

WLC-01 | Inappropriate Upper Limits For Fees

Category	Severity	Location	Status
Logical Issue	● Minor	extensions/WithdrawalLimitation.sol (audit): 154	ⓘ Acknowledged

Description

The fee is calculated via the function `_computeFeePercentage()`:

```
220    function _computeFeePercentage(uint256 _amount, uint256 _percentage) internal view
virtual returns (uint256) {
221      return (_amount * _percentage) / _MAX_PERCENTAGE;
222    }
```

The percentage of the fee is set via function `_setUnlockFeePercentages()`. However, when setting the fee percentage, the fee percentage can be set as `_MAX_PERCENTAGE`, meaning all the transferred asset will be collected as fee.

```
151    function _setUnlockFeePercentages(address[] calldata _tokens, uint256[] calldata
_percentages) internal virtual {
152      require(_tokens.length == _percentages.length, "WithdrawalLimitation: invalid
array length");
153      for (uint256 _i; _i < _tokens.length; _i++) {
154        require(_percentages[_i] <= _MAX_PERCENTAGE, "WithdrawalLimitation: invalid
percentage");
155        unlockFeePercentages[_tokens[_i]] = _percentages[_i];
156      }
157      emit UnlockFeePercentagesUpdated(_tokens, _percentages);
158    }
```

Recommendation

It is recommended to set a more appropriate limit the fee when calling `_setUnlockFeePercentages()`.

Alleviation

[Ronin]:

The team acknowledged this issue and decided not to change the current codebase.

WLK-01 | Questions About Tiers Model

Category	Severity	Location	Status
Inconsistency	● Informational	extensions/WithdrawalLimitation.sol (PR21): 250~256	ⓘ Acknowledged

Description

The auditors do not see how the Tiers model is implemented through the code, especially :

- Tiers 2: All signatures from validators are required;
- Tiers 3: All signatures from validators are required, one additional human review to unlock the fund

The documentation states: "There will be another constraint on the number of token that can be withdraw in a day. We propose to cap the value at $50M. Since withdrawal of Tier 3 already requires human review, it will not be counted in daily withdrawal limit.".

However, within the `_setDailyWithdrawalLimits()` function, there is no validation that this limit cannot be pushed beyond 50M:

```
250   function _setDailyWithdrawalLimits(address[] calldata _tokens, uint256[] calldata _limits) internal virtual {
251     require(_tokens.length == _limits.length, "WithdrawalLimitation: invalid array length");
252     for (uint256 _i; _i < _tokens.length; _i++) {
253       dailyWithdrawalLimit[_tokens[_i]] = _limits[_i];
254     }
255     emit DailyWithdrawalLimitsUpdated(_tokens, _limits);
256   }
```

Recommendation

The auditors would like to have more information about how the Tiers model is implemented through the code.

Alleviation

[Ronin]:

The limit is not fixed yet, it is still an on-going discussion and can be changed via voting. Also the limit is just another layer of risk management. We don't know the perfect numbers for the limits yet so, we will need to roll it out and measure it.

Optimizations

ID	Title	Category	Severity	Status
BMC-01	Variables That Could Be Declared As Immutable	Gas Optimization	● Optimization	✓ Resolved

BMC-01 | Variables That Could Be Declared As Immutable

Category	Severity	Location	Status
Gas Optimization	● Optimization	migration/BridgeMigration.sol (audit): 32	✓ Resolved

Description

The linked variables `weth` assigned in the constructor can be declared as `immutable`. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.

Recommendation

It is recommended to declare these variables as immutable.

Alleviation

[Ronin]:

The team resolved this issue by setting the variables as `immutable` in the PR 22.

Appendix

SCSVSv2 Checks

CertiK used the SCSVSv2 referential to perform additional testing on Ronin bridge.

C6.1 - Verify that bridge requires all necessary values to be included in the message and signed: chain ids, receiver, amount, nonce.

In `MainchainGatewayV2`, the function `_submitWithdrawal()` uses a receipt:

```
struct Receipt {
  uint256 id; //nonce
  Kind kind;
  Token.Owner mainchain;
  Token.Owner ronin;
  Token.Info info;
}
```

With:

```
struct Info {
  Standard erc;
  // For ERC20:  the id must be 0 and the quantity is larger than 0.
  // For ERC721: the quantity must be 0.
  uint256 id;
  uint256 quantity; //Quantity
}

struct Owner {
  address addr; //Destination
  address tokenAddr;
  uint256 chainId; //chainId
}
```

In `RoninGatewayV2`, the function `_depositFor()` also uses a receipt:

```
struct Receipt {
  uint256 id; //nonce
  Kind kind;
  Token.Owner mainchain;
  Token.Owner ronin; //chainId
  Token.Info info;
}
```

With:

```
struct Info {
  Standard erc;
  // For ERC20:  the id must be 0 and the quantity is larger than 0.
  // For ERC721: the quantity must be 0.
  uint256 id;
  uint256 quantity; //Quantity
}

struct Owner {
  address addr; //Destination
  address tokenAddr;
  uint256 chainId; //chainId
}
```

Those elements appear to be in compliance with **C6.1**.

C6.2 - Verify that used signatures are invalidated to protect bridge from replay attacks.

Example _depositFor() - Ronin

What happens when tokens are bridged is that a proposal `ReceiptVote` (`depositVote[.chainId][_id]`) is created :

```
struct ReceiptVote {
  VoteStatus status; // Goes from Pending to Executed when funds are sent
  bytes32 finalHash;
  /// @dev Mapping from voter => receipt hash
  mapping(address => bytes32) receiptHash;
  /// @dev Mapping from receipt hash => vote weight
  mapping(bytes32 => uint256) weight;
}
```

Once a vote is passed, funds are sent and vote status is updated to `Executed`. It is not possible to replay a proposal because the vote will have been marked as executed:

```
require(_vote.status == VoteStatus.Pending, "Governance: the vote is finalized");
```

Example _submitWithdrawal() - Mainchain What happens when tokens are withdrawn is that a receipt digest is computed.

```
bytes32 _receiptDigest = Transfer.receiptDigest(_domainSeparator, _receipt.hash());
```

The function will check that the Ronin validators signed for this particular Digest:

```
_signer = ecrecover(_receiptDigest, _sig.v, _sig.r, _sig.s);
(...)
_weight += _validatorContract.getValidatorWeight(_signer);
```

As a consequence, it is not possible to forge fake requests because it would mean having access to Ronin Validators.

In addition, to avoid replay, a check is performed before processing withdrawal:

```
require(withdrawalHash[_id] == bytes32(0), "MainchainGatewayV2: query for processed
withdrawal");
```

If withdrawal is successful, the variable is updated:

```
withdrawalHash[_id] = _receiptHash;
```

Those elements appear to be in compliance with **C6.2**.

C6.3 - Verify that message hash generation algorithm is resistant to collision attacks.

The use of `keccak256()` function is OK as of today June 20th, 2022.

C6.4 - Verify that bridge includes source and destination chains identifiers in the signed message and correctly verifies them.

The verification is performed upon withdrawals:

```
  function _submitWithdrawal(Transfer.Receipt calldata _receipt, Signature[] memory
_signatures)
(...)
require(_receipt.mainchain.chainId == block.chainid, "MainchainGatewayV2: invalid chain
id");
```

The verification is also performed upon deposits:

```
  function _depositFor(Transfer.Receipt memory _receipt,address _validator,uint256
_weight,uint256 _minVoteWeight) internal {
(...)
    require(_receipt.ronin.chainId == block.chainid, "RoninGatewayV2: invalid chain id");
```

Those elements appear to be in compliance with **C6.4**.

C6.5 - Verify that bridge does not allow to spoof chain identifier.

Because of the verification performed previously in C6.4, it is not possible to spoof chain identifier.

C6.6 - Verify that bridge uses a nonce parameter to allow the same operation (the same sender, receiver and amount) to be executed multiple times.

A nonce is used for deposits (`depositCount`) :

```
    uint256 _depositId = depositCount++;
    Transfer.Receipt memory _receipt = _request.into_deposit_receipt(
      _requester,
      _depositId,
      _token.tokenAddr,
      roninChainId
    );
```

A nonce is used for withdrawals (`withdrawalCount`) :

```
    uint256 _withdrawalId = withdrawalCount++;
    Transfer.Receipt memory _receipt = _request.into_withdrawal_receipt(
      _requester,
      _withdrawalId,
      _mainchainTokenAddr,
      _chainId
    );
```

Those elements appear to be in compliance with **C6.6**.

C6.7 - Verify signed message cannot be used in a different context (use domain separator from EIP-712).

Because of the reasons mentioned in **C6.2**, contracts appear to be in compliance with **C6.6**. Also, MainchainGateway contract uses DOMAIN SEPARATOR from EIP-712.

Example - For withdrawals from Ronin to other chains, Domain separator is used

```
bytes32 _receiptDigest = Transfer.receiptDigest(_domainSeparator, _receiptHash);
```

This Domain Separator is unique for each `chainId` :

```
function _updateDomainSeparator() internal {    _domainSeparator = keccak256(
      abi.encode(
        keccak256("EIP712Domain(string name,string version,uint256 chainId,address
verifyingContract)"),
        keccak256("MainchainGatewayV2"),
        keccak256("2"),
        block.chainid,
        address(this)
      )
    );
  }
```

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK’s prior written consent in each instance.

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED “AS IS” AND

"AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.