verichains

SECURITY AUDIT OF

RONIN BRIDGE SMART CONTRACTS

Public Report

Jun 28, 2022

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on **May 17, 2022**. We would like to thank the Sky Mavis for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Ronin Bridge Smart Contracts. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified some vulnerable issues in the application, along with some recommendations. Sky Mavis team has resolved and updated smart contract code following our recommendations. Ronin Bridge Smart Contracts has passed with no Medium, High, or Critical severity issues.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Ronin Bridge Smart Contracts

The Ronin Bridge allows funds to be transferred from Ethereum to the Ronin Network and vice versa. The main features of the bridge are:

- Fast & seamless transactions with almost instant confirmation.
- Drastically reduced gas fees.
- The ability to withdraw Axie assets back to Ethereum Mainnet.
- Simplified on-boarding for new users, through a customized wallet solution.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of Ronin Bridge Smart Contracts. It was conducted on commit bbfe4508d2b9c9ecd3de5f9b5690b7c5b82f53c8 from git repository *https://github.com/axieinfinity/ronin-smart-contracts-v2*.

The latest version of the following files was made available in the course of the review:

SHA256 Sum	File
ad15c1d1e9af2d7d44fd3f79a2490201bf0fca4bb5e574a762553ec3a964cb6b	contracts/v0.8/extensions/GatewayV2.sol
0f40acf1dc9093a94ed61a1a1ed288f6c283fb8a4c4ca7e30ba204b058e12005	contracts/v0.8/extensions/governance/Governance.sol
1bb9630106a239d0ac9b9f6b2208e6f2110243d880ef9f4b59ce8730a0df4e09	contracts/v0.8/extensions/governance/GatewayGovernance.sol
2818fd8ddcebcfaa3f1e99b39d0d66e46847f81d8a044c3500b5ede21acf5495	contracts/v0.8/extensions/governance/GlobalProposalGovernance.sol
12bd0a2c0214a6018b4354f8843b92b05891e0c44c6260c450edfd76b02ce0e8	contracts/v0.8/extensions/governance/ProposalGovernance.sol
abb733ea41a7dc98f2bb173915b5c8d93b2c69e41c387104192f8e2223b36bc4	contracts/v0.8/extensions/WithdrawalLimitation.sol
24aef138712d0d2f8d18da3ac5fd2b873d10ce60eb845ffdcb0768fb7b452580	contracts/v0.8/extensions/HasProxyAdmin.sol
cb70c81c0e18125d236bf4a31352f7092abaa5b47d5f46e22c25f52bd1593fc8	contracts/v0.8/extensions/MinimumWithdrawal.sol

56f02515eac98350739f670a0a9a28f3974431d198f55db0bc637cb793a0c127	**contracts/v0.8/extensions/TransparentUpgradeableProxyV2.sol**
8ab73c3fe72a92f2bdd016f3a5c51de87db04e0da7d8fd82391bc1a2628654d2	**contracts/v0.8/mocks/MockERC721.sol**
49f5985faaab611c67c0e5e40231ba3dcbb8604191c5ea93499e2d064a20d44e	**contracts/v0.8/mocks/MockGatewayV2.sol**
a93c33101084deef5fca264a4dff73f05cce8ca33519648d2128596b62946214	**contracts/v0.8/interfaces/IERC721Mintable.sol**
be50351f320fb7b50630c2bd922c970de370166a0b004af4ff539582fa3295d9	**contracts/v0.8/interfaces/IWeightedValidator.sol**
4795937cb211a75c6c525b06508e7f57d73e7bbc24d6b4e36cb3d26b2c19aea5	**contracts/v0.8/interfaces/IERC20Mintable.sol**
688a73efabe2972c17647f4daba15e1e55d59aa9a5d267cf7c1f2aca26dddfda	**contracts/v0.8/interfaces/IWETH.sol**
f9f8a78e55b9de1c5627e5be695e004c7bc29a3e387358e5a25d430550791052	**contracts/v0.8/interfaces/SignatureConsumer.sol**
5e12f2f1134550dfe70bc1f2503ff11fb9181c6b874f29bc262393e01c5daa12	**contracts/v0.8/interfaces/IQuorum.sol**
c1abe887c2b7fadcc37d07924257aea9ef5fbd52f132a1b0c2ee108ec02de48d	**contracts/v0.8/common/RoninValidator.sol**
b20947552bd082d9cbac22194b63719ba055aa10aac977ee70adc8cf369bdc11	**contracts/v0.8/common/GovernanceAdmin.sol**
1a24068588f8e02cedb1ea4d0a97e54514c9e6fee08e024159a9dc458917cc90	**contracts/v0.8/mainchain/IMainchainGatewayV2.sol**
1de77980597b60998c7229cd95c513b574e43aac151c8a33e55ace818dc88f98	**contracts/v0.8/mainchain/MainchainGatewayV2.sol**
20983d4eb425c6a75f50df57faa2f48cd7c253c7960d7a271db5048cf287e228	**contracts/v0.8/library/Proposal.sol**
0315dc0386363aa73cf1c9fc8cb37952acc9fbb920748b9fcb08e668d51191ab	**contracts/v0.8/library/Token.sol**
86ba568b7e2d0c28b57e319423db1291fa5409a0408e755978f78fd6ebdceb53	**contracts/v0.8/library/Transfer.sol**
7fa90967172ab597e796ad4fbcff98c826890670af5f475cda02452c28d0584f	**contracts/v0.8/library/BridgeProposal.sol**

ebaac64bd83794d8051c5e3067c04320a24055e14dd5454a17dba7cb117ad23b	**contracts/v0.8/library/Ballot.sol**
f88b96867e7e49adfa751ba2c651b189b05db77d835c36673da5f8cfbffa2561	**contracts/v0.8/ronin/RoninGatewayV2.sol**
07a169a239fba383088cdd6c4eb2e76deb9c594ffea9b5c61c827420fe0ed3f9	**contracts/v0.8/ronin/IRoninGatewayV2.sol**
e27cf5a7108005fc14a0a8c8d6361510cb4b1b874bf17e5eb30b2b2426b81f2a	**test/v0.8/common/RoninValidator.test.ts**
d2cb6a8db49913636cd9d37e46788812d02a8351a7021915484e1b1482bf75ae	**test/v0.8/common/GovernanceAdmin.test.ts**
f0126aafc0dc40c85a3848f0e4548e52c716d45db600d9bf513f5271d1ce3ba2	**test/v0.8/mainchain/MainchainGatewayV2.test.ts**
0f39c39321a8b8f8c08c72002f10bcc94f7dacd92f4f49596bacbf307e554c8d	**test/v0.8/ronin/RoninGatewayV2.test.ts**

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The Ronin Bridge Smart Contracts was written in Solidity language, with the required version to be ^0.8.0. The source code was written based on OpenZeppelin's library.

2.1.1. Validators

Only validators can produce blocks on Ronin. Validators can also acknowledge deposit and withdrawal events to facilitate asset transfers between Ronin and other EVM-based chains.

2.1.2. Bridges

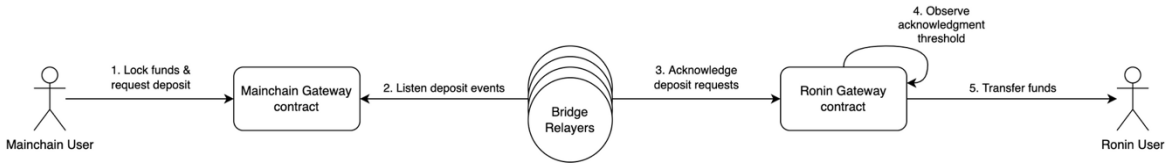
The bridge is designed to support multiple chains. When a deposit event happens on the mainchain, the Bridge component in each validator node will pick it up and relay it to Ronin by sending a corresponding transaction. For withdrawal and governance events, it will start from Ronin and then being relaid on other chains.

Depending on the action, these transactions will be relayed to GovernanceAdmin (changes in the validator list/updates to the consensus threshold) or RoninGatewayV2 (deposits, withdrawals).

2.1.3. Deposits

Users can deposit ETH, ERC20, and ERC721 (NFTs) by sending transactions to MainchainGatewayV2 and waiting for the deposit to be verified on Ronin. The validator will listen to the event on mainchain and then acknowledge the deposit on Ronin. The gateway should have a mapping between token contracts on Ethereum and on Ronin before the deposit can take place.

For deposit there is no restriction on how large a deposit can be.



2.1.4. Withdrawals

For withdrawal there are certain restrictions:

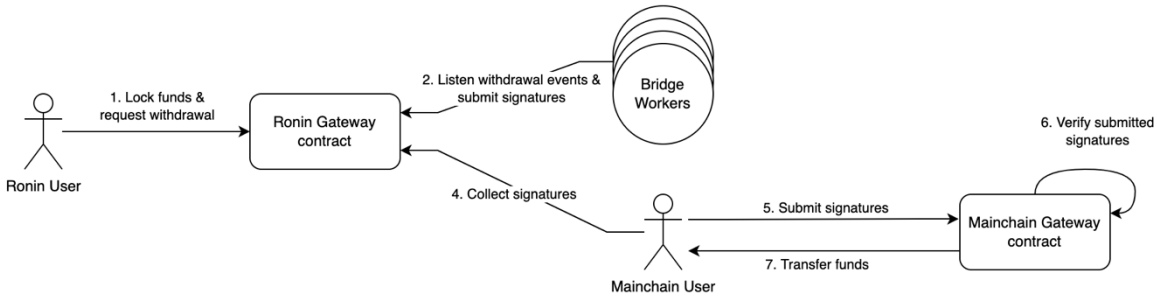
- Withdrawal tiers

There are 3 withdrawal tiers with different level of threshold required to withdraw. This is what was proposed initially by the dev team:

Tier	Withdrawal Value	Threshold
Tier 1	-	The normal withdrawal/deposit threshold
Tier 2	>= $1M	All signatures from validators are required
Tier 3	>= $10M	All signatures from validators are required, one additional human review to unlock the fund

- Daily withdrawal limit

There will be another constraint on the number of tokens that can be withdrawn in a day. They proposed to cap the value at $50M. Since withdrawal of Tier 3 already requires human review, it will not be counted in daily withdrawal limit.

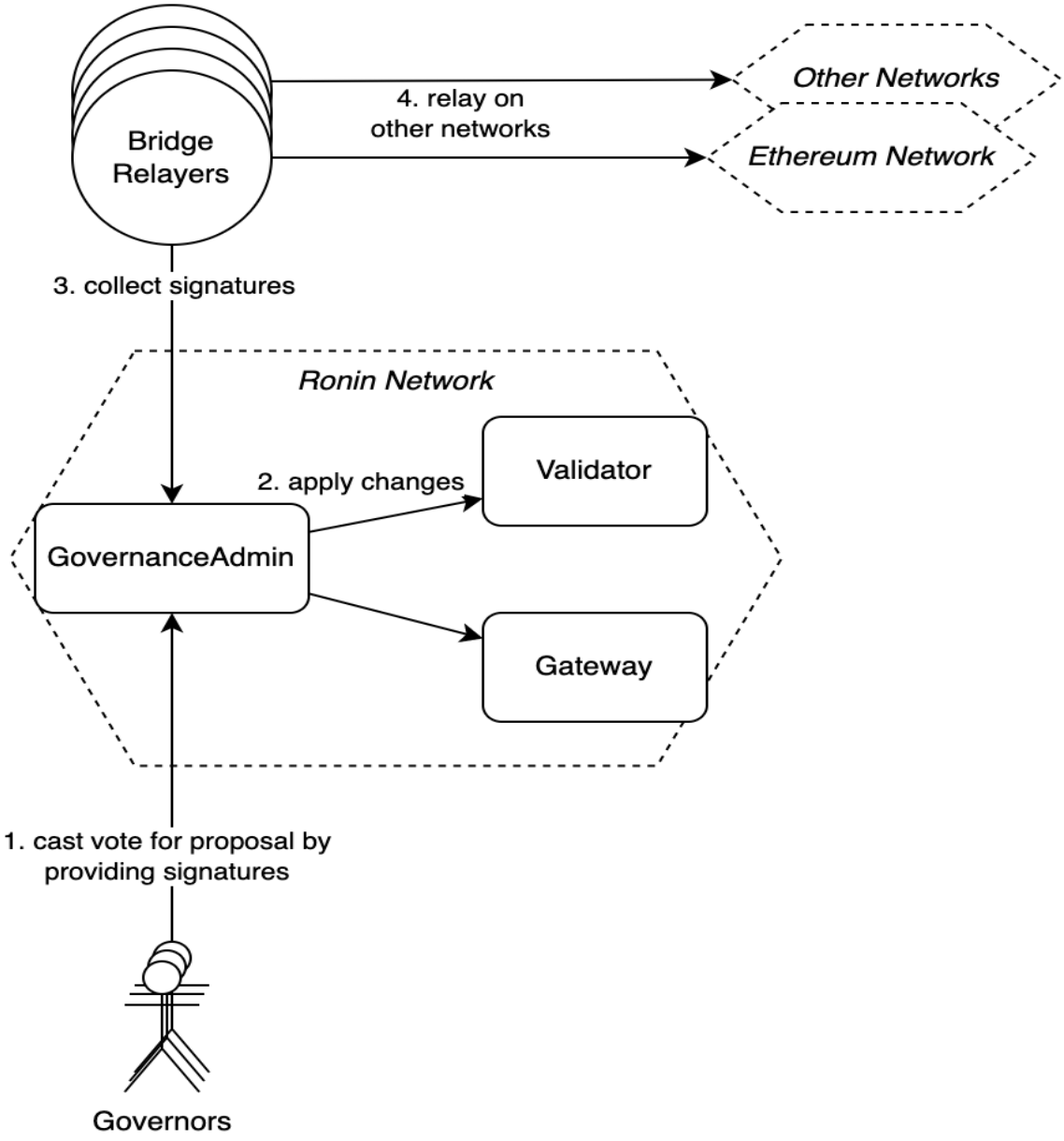


Normal withdrawal flow (tier 1 + tier 2). For tier 3, a separated human address will need to unlock the fund.

2.1.5. Governance

The GovernanceAdmin contract is mainly responsible for the governance process via a decentralized voting mechanism. Apart from the validator there is a list of governors, each validator has a corresponding governor. The governors can vote for changes such as: adding/removing validators, upgrading contracts, changing thresholds, etc. At any instance,

there will be at most one governance vote going on. The governor can be seen as a cold wallet while the validator can be considered as a hot wallet.



2.2. Architecture Security Review

In the section, we are going to summarize security assumptions of the architecture. **Security assumptions are trustworthy without verification or disputation**. We also analyze **current security controls and gaps** of the security assumptions. Finally, we offer **recommendations** to improve the architecture security.

2.2.1. Security Assumptions

The architecture is secure as long **as the number of malicious governance parties is unlikely to be greater than G**. G is the threshold in the GovernanceAdmin contract. The maximum impact is the total loss of the project fund.

The user's deposit is secure as long as **the number of malicious validators is not greater than N**. N is the threshold in the Ronin Gateway contract. The impacts are:

1. **Censorship**. The deposit could be locked if malicious validators refuse to acknowledge the assets.
2. **Counterfeit**. Attackers are able to mint counterfeit assets in Ronin. The attacks could be hard to reverse if the counterfeit assets are withdrawn to mainnet. Maximum impact depends on the withdrawal policies.

The user's withdrawal is secure as long as **the number of malicious validators is not greater than X**. X is the threshold in the Mainchain Gateway contract. The impacts are:

1. No >$50M **unattended withdrawn** per day.
2. >$50M withdrawal feasible if one human factor approves.
3. **Censorship**. No one is able to transfer assets if malicious validators refuse to cooperate.
4. **Counterfeit**. Attacker able to fake the withdrawal events to transfer assets. Maximum impact limited by the withdrawal rules.

In short there are 4 important security assumptions:

1. There is no risk of complete loss if the threshold of governance private keys are secured.
2. There is no risk of complete loss if the human private key is secured.
3. The maximum impact of leak validator private keys is $50M if and only if the community is able to detect and stop attacks in time.
4. Censorship is unlikely if the malicious validators are replaceable.

2.2.2. Security Controls and Gaps

The security assumptions only worked if the smart contracts were correctly implemented and proper out-of-chain security controls.

1. **There is no risk of complete loss if the threshold of governance private keys are secured.**
   We assume that all governance private keys should be protected by hardware wallets. We assume that the threshold of governance voting should justify the risk of complete loss.

2. **There is no risk of complete loss if the human private key is secured**
   We assume that the human private key should be protected in hardware wallets. While it is fairly low risk that all validators plus the private key are compromised; the role of the special person is not clearly defined.

3. **The maximum impact of leak validator private keys is $50M**
   We assume that the validator private keys should be protected by **Key Management Service to reduce the risk of leak keys**. In the worst cases, enough validators compromised could allow attackers to perform as many withdrawals as they want. Interesting that they could **choose to attack at the end of day when the counter is reseted**, then they could inflict a maximum **$100M** in a very short time. **We assume that the community is able to detect and respond quickly to limit the loss at $50M**.

4. **Censorship is unlikely**
   We assume that all governance private keys are available in case to vote out the malicious validators.

2.2.3. Security Recommendation

Below are our recommendations for a more secure architecture and design of Ronin bridge contract.

1. **Multi-tier security policies** for GovernanceAdmin for different tasks.

2. **Delay in governance changes** (24 hours for example) are also recommended to give the community a chance to react.

3. **The role of the special person** who manually approves >$50M transactions should be defined and have proper security controls around the role.

4. **Monitoring, monitoring and monitoring.** The ultimate goal of the architecture is to delay the attack and allow the community to detect and respond. If attackers perform withdrawals undetected, they are able to inflict more than $50M loss (up to complete loss of the funds)

Further risk control improvements could be considered

5. The current risk controls focus on delaying attackers (by hard cap) but unclear how to react against them. We assume in this case the Ronin validators use GovernanceAdmin to prevent further attacks. But the risk of slowness or absence of validators could cause more damage. **The incident response protocols** should be implemented by smart contracts which help quickly respond and limit loss.

6. **Using Zero Knowledge Proof** with the current risk control rules to improve the security assumptions rather than blindly trust the validators.

7. **Censorship security assumption** could be improved by giving users an emergency exit.

2.3. Code Review Findings

During the code audit process, the audit team identified some vulnerabilities in the given version of Ronin Bridge Smart Contracts. Sky Mavis team fixed the code, according to Verichains's draft reports, in commit abe18fe7c333657297fa29409025dbb54852d204.

Title	Severity	Status
WithdrawalLimitation.sol - lastDateSynced is not updated for each token	CRITICAL	Resolved
MainchainGatewayV2.sol - WITHDRAWAL_UNLOCKER_ROLE should not be granted to validators	MEDIUM	Resolved
RoninValidator.sol - Incorrect handling can lead to duplicated governor	LOW	Resolved
RoninGatewayV2.sol - bulkAcknowledgeMainchainWithdrew can be abused to prevent calling of requestWithdrawalSignatures	LOW	Resolved
DEFAULT_ADMIN_ROLE should be granted to the GovernanceAdmin contract	LOW	Acknowledged

Table 2. Findings

2.3.1. WithdrawalLimitation.sol - lastDateSynced is not updated for each token CRITICAL

When withdrawing tokens via the _submitWithdrawal function, the _recordWithdrawal function will be called to track the withdrawn tokens in a day. However, the lastDateSynced variable is used for all tokens, so when a user withdraws token X, the lastDateSynced will be updated to the current date. If another user withdraws token Y after that, the value of lastSyncedWithdrawal[Y] will not be reset, so that the _reachedWithdrawalLimit function may return true resulting in token Y that cannot be withdrawn.

```
function _recordWithdrawal(address _token, uint256 _quantity) internal virtual
{
    uint256 _currentDate = block.timestamp / 1 days;
    if (_currentDate > lastDateSynced) {
      lastDateSynced = _currentDate; // INCORRECT
      lastSyncedWithdrawal[_token] = _quantity;
    } else {
      lastSyncedWithdrawal[_token] += _quantity;
```

```
    }
}

function _reachedWithdrawalLimit(address _token, uint256 _quantity) internal view virtual
returns (bool) {
    if (_lockedWithdrawalRequest(_token, _quantity)) {
        return false;
    }

    uint256 _currentDate = block.timestamp / 1 days;
    if (_currentDate > lastDateSynced) {
        return dailyWithdrawalLimit[_token] <= _quantity;
    } else {
        return dailyWithdrawalLimit[_token] <= lastSyncedWithdrawal[_token] + _quantity; //
lastSyncedWithdrawal[_token] is not reset
    }
}
```

RECOMMENDATION

The lastDateSynced should be updated for each token, so we should convert this variable from type uint256 to mapping(address => uint256).

UPDATES

- *May 17, 2022*: This issue has been acknowledged and fixed by Sky Mavis team in pull request *https://github.com/axieinfinity/ronin-smart-contracts-v2/pull/7*.

2.3.2. MainchainGatewayV2.sol - WITHDRAWAL_UNLOCKER_ROLE should not be granted to validators MEDIUM

In the MainchainGatewayV2.test.ts testcase file, we notice that the WITHDRAWAL_UNLOCKER_ROLE is granted to all validators. This means any validator can call the unlockWithdrawal function.

```
validatorSet[network.name] = validators.map((v, i) => ({
    validator: v.address,
    governor: governors[i].address,
    weight: 1,
}));
accountSet['relayers'][network.name] = [relayer.address];
validatorThreshold[network.name] = { numerator: 1, denominator: validators.length };
gatewayThreshold[network.name] = { numerator: 1, denominator: validators.length };
mainchainMappedToken[network.name] = {
    mainchainTokens: [weth.address, erc721.address],
```

```
    roninTokens: [weth.address, erc721.address],
    fulSigThresholds: [10, 0], // tier-2 withdrawal
    lockedThresholds: [20, 0], // tier-3 withdrawal
    unlockFeePercentages: [100_000, 100_000], // 10%
    dailyWithdrawalLimits: [12, 0], // daily limit dont apply for tier-3 withdrawal
};
mainnetChainId[network.name] = [network.config.chainId!];
roninChainId[network.name] = 2020;
namedAddresses['weth'][network.name] = weth.address;
namedAddresses['roleSetter'][network.name] = deployer.address;
namedAddresses['governanceAdminOwner'][network.name] = deployer.address;
accountSet['withdrawalUnlockers'][network.name] = validators.map((v) => v.address);
// WITHDRAWAL_UNLOCKER_ROLE is granted for all validators
```

If a validator account is compromised, the attacker can observe all the tier-3 withdrawal transactions and call the unlockWithdrawal function to collect the withdrawal fee. In a worse situation, if all the validators are compromised, the attackers can easily use any validator account to call the unlockWithdrawal function and easily bypass the additional human review step. So, we can conclude that the security levels of tier-2 and tier-3 withdrawals are the same.

```
function unlockWithdrawal(Transfer.Receipt calldata _receipt) external
onlyRole(WITHDRAWAL_UNLOCKER_ROLE) {
    bytes32 _receiptHash = _receipt.hash();
    require(withdrawalHash[_receipt.id] == _receipt.hash(), "MainchainGatewayV2: invalid
receipt");
    require(withdrawalLocked[_receipt.id], "MainchainGatewayV2: query for approved
withdrawal");
    delete withdrawalLocked[_receipt.id];
    emit WithdrawalUnlocked(_receiptHash, _receipt);

    address _token = _receipt.mainchain.tokenAddr;
    if (_receipt.info.erc == Token.Standard.ERC20) {
      Token.Info memory _feeInfo = _receipt.info;
      _feeInfo.quantity = _computeFeePercentage(_receipt.info.quantity,
unlockFeePercentages[_token]);
      Token.Info memory _withdrawInfo = _receipt.info;
      _withdrawInfo.quantity = _receipt.info.quantity - _feeInfo.quantity;

      _feeInfo.handleAssetTransfer(payable(msg.sender), _token, wrappedNativeToken);
      _withdrawInfo.handleAssetTransfer(payable(_receipt.mainchain.addr), _token,
wrappedNativeToken);
    } else {
      _receipt.info.handleAssetTransfer(payable(_receipt.mainchain.addr), _token,
wrappedNativeToken);
    }
```

```
    emit Withdrew(_receiptHash, _receipt);
}
```

RECOMMENDATION

The WITHDRAWAL_UNLOCKER_ROLE must be granted for another account, which is neither a validator nor governor.

UPDATES

- *May 17, 2022*: This issue has been acknowledged and fixed by Sky Mavis team in pull request *https://github.com/axieinfinity/ronin-smart-contracts-v2/pull/15*.

2.3.3. RoninValidator.sol - Incorrect handling can lead to duplicated governor LOW

When updating a validator using the _updateValidator function, if the governor is changed, it will be updated immediately without checking for duplication.

```
function _updateValidator(WeightedValidator memory _v) internal virtual {
    require(_v.weight > 0, "RoninValidator: invalid weight");

    uint256 _weight = _validatorWeight[_v.validator];
    if (_weight == 0) {
      revert(
        string(
          abi.encodePacked("RoninValidator: ", Strings.toHexString(uint160(_v.validator),
20), " is not a validator")
        )
      );
    }

    uint256 _count = _validators.length;
    for (uint256 _i = 0; _i < _count; _i++) {
      if (_validators[_i] == _v.validator) {
        _totalWeights -= _weight;
        _totalWeights += _v.weight;

        if (_governors[_i] != _v.governor) {
          require(_governorWeight[_v.governor] == 0, "fixed");
          delete _governorWeight[_governors[_i]];
          _governors[_i] = _v.governor; // GOVERNOR MAY BE DUPLICATED
        }

        _validatorWeight[_v.validator] = _v.weight;
        _governorWeight[_v.governor] = _v.weight;
```

```
            return;
        }
    }
}
```

RECOMMENDATION

The duplication check _governorWeight[_v.governor] > 0 should be required before updating the new governor.

UPDATES

- *May 17, 2022*: This issue has been acknowledged and fixed by Sky Mavis team in pull request *https://github.com/axieinfinity/ronin-smart-contracts-v2/pull/7*.

2.3.4. RoninGatewayV2.sol - bulkAcknowledgeMainchainWithdrew can be abused to prevent calling of requestWithdrawalSignatures LOW

When a user submits a withdrawal request by calling requestWithdrawalFor function, he will need to call the requestWithdrawalSignatures function after that to trigger transaction signing from validators. However, a single validator may call the bulkAcknowledgeMainchainWithdrew function before that to set the value of mainchainWithdrew[_withdrawalId] to true which will prevent calling of requestWithdrawalSignatures function. In that case, user won't be able to withdraw their tokens from mainchain.

```
function requestWithdrawalSignatures(uint256 _withdrawalId) external whenNotPaused
{
    require(!mainchainWithdrew[_withdrawalId], "RoninGatewayV2: withdrew on mainchain
already");
    Transfer.Receipt memory _receipt = withdrawal[_withdrawalId];
    require(_receipt.ronin.chainId == block.chainid, "RoninGatewayV2: query for invalid
withdrawal");
    emit WithdrawalSignaturesRequested(_receipt.hash(), _receipt);
}

function bulkAcknowledgeMainchainWithdrew(uint256[] calldata _withdrawalIds) external
{
    // This authorized method caller already
    _getValidatorWeight(msg.sender);

    uint256 _withdrawalId;
    for (uint256 _i; _i < _withdrawalIds.length; _i++) {
```

```
        _withdrawalId = _withdrawalIds[_i];
        require(!mainchainWithdrew[_withdrawalId], "RoninGatewayV2: withdrew on mainchain
already");

        mainchainWithdrew[_withdrawalId] = true;
        Transfer.Receipt memory _withdrawal = withdrawal[_withdrawalId];
        emit MainchainWithdrew(_withdrawal.hash(), _withdrawal);
    }
}
```

RECOMMENDATION

We can remove the !mainchainWithdrew[_withdrawalId] check inside the requestWithdrawalSignatures function since one withdrawal receipt can only be used once in the MainchainGatewayV2 contract.

UPDATES

- *May 17, 2022*: This issue has been acknowledged and fixed by Sky Mavis team in pull request *https://github.com/axieinfinity/ronin-smart-contracts-v2/pull/15*.

2.3.5. DEFAULT_ADMIN_ROLE should be granted to the GovernanceAdmin contract LOW

In this current implementation, the DEFAULT_ADMIN_ROLE of the MainchainGatewayV2 and RoninGatewayV2 contracts is granted to the accounts that can be controlled by just a single user. An account with the DEFAULT_ADMIN_ROLE can manage the WITHDRAWAL_UNLOCKER_ROLE in the MainchainGatewayV2, and the WITHDRAWAL_MIGRATOR role in the RoninGatewayV2 contract. For example, an account with the DEFAULT_ADMIN_ROLE can assign another user as a withdrawal unlocker who can approve tier-3 transactions and also collect the transaction fees.

RECOMMENDATION

The DEFAULT_ADMIN_ROLE of the MainchainGatewayV2 and RoninGatewayV2 contracts should be set to the address of the GovernanceAmin contract. That way, some important actions like granting the WITHDRAWAL_UNLOCKER_ROLE or WITHDRAWAL_MIGRATOR role will only be performed via the voting process.

UPDATES

- *May 17, 2022*: This issue has been acknowledged by Sky Mavis team.

2.4. Additional notes and recommendations

2.4.1. Ballot.sol - Governor cannot see proposal data when signing the ballot INFORMATIVE

When signing the ballot using EIP-712, the signers can only see the proposalHash and support values. Because of that, they still cannot see the detail of the proposal they are signing for. So, this approach will not take advantage of typed structure data hashing and signing defined by EIP-712.

```
library Ballot {
  using ECDSA for bytes32;

  enum VoteType {
    For,
    Against
  }

  // keccak256("Ballot(bytes32 proposalHash,uint8 support)");
  bytes32 public constant BALLOT_TYPEHASH =
0xd900570327c4c0df8dd6bdd522b7da7e39145dd049d2fd4602276adcd511e3c2;


  function hash(bytes32 _proposalHash, VoteType _support) internal pure returns (bytes32)
{
    return keccak256(abi.encode(BALLOT_TYPEHASH, _proposalHash, _support));
  }
}
```

RECOMMENDATION

The ballot type hash should be changed to keccak256("Ballot(ProposalDetail proposal,uint8 support)ProposalDetail(uint256 nonce,uint256 chainId,address[] targets,uint256[] values,bytes[] calldatas)") (the hash function should also be updated accordingly).

UPDATES

- *May 17, 2022*: This issue has been acknowledged by Sky Mavis team.

2.4.2. MainchainGatewayV2.test.ts - Using wrong comparison operator INFORMATIVE

In the MainchainGatewayV2 testcases file, the network.name check is using the wrong comparison operator =. It should be changed to ==.

```
describe('Mainchain Gateway V2 test', () => {
    before(async () => {
        let signers: SignerWithAddress[];
        [deployer, normalUser, relayer, ...signers] = await ethers.getSigners();
        validators = signers.slice(0, signers.length / 2);
        governors = signers.slice(signers.length / 2);
        if (validators.length > governors.length) {
            validators.pop();
        } else if (validators.length < governors.length) {
            governors.pop();
        }

        weth = await new WETH__factory(deployer).deploy();
        erc20 = await new ERC20Mintable__factory(deployer).deploy();
        await erc20.addMinters([deployer.address]);
        erc721 = await new MockERC721__factory(deployer).deploy('ERC721', 'ERC721', '');

        if ((network.name = Network.Hardhat)) { // WRONG COMPARISON OPERATOR
            validatorSet[network.name] = validators.map((v, i) => ({
                validator: v.address,
                governor: governors[i].address,
                weight: 1,
            }));
        }
        // ...
    })
})
```

UPDATES

- *May 17, 2022*: This issue has been acknowledged and fixed by Sky Mavis team in pull request *https://github.com/axieinfinity/ronin-smart-contracts-v2/pull/7*.

2.4.3. MainchainGatewayV2.test.ts - Wrong testcase description INFORMATIVE

In the MainchainGatewayV2 testcases file, the description is wrong for the following testcase.

```
it('Should be able to deposit empty amount', async () => {
  await expect(
```

```
    normalUser.sendTransaction({
      to: gateway.address,
      value: 0,
    })
  ).revertedWith('Token: invalid info');
  await expect(
    gateway.connect(normalUser).requestDepositFor({
      recipientAddr: deployer.address,
      tokenAddr: weth.address,
      info: { ...info, quantity: 0 },
    })
  ).revertedWith('Token: invalid info');
});
```

RECOMMENDATION

The testcase description should be changed to Should not be able to deposit empty amount.

UPDATES

- *May 17, 2022*: This issue has been acknowledged and fixed by Sky Mavis team in pull request *https://github.com/axieinfinity/ronin-smart-contracts-v2/pull/7*.

2.4.4. MainchainGatewayV2.sol - Withdrawal limit check for ERC721 should be skipped for gas saving INFORMATIVE

In the _submitWithdrawal function, there is a required check !_reachedWithdrawalLimit(_tokenAddr, _quantity) that is used to check the withdrawal limit of the ERC20 tokens. In case of ERC721 token withdrawal, we should skip this limit check for gas-saving.

```
function _submitWithdrawal(Transfer.Receipt calldata _receipt, Signature[] memory
_signatures)
    internal
    virtual
    returns (bool _locked)
{
    uint256 _id = _receipt.id;
    uint256 _quantity = _receipt.info.quantity;
    address _tokenAddr = _receipt.mainchain.tokenAddr;

    _receipt.info.validate();
    require(_receipt.kind == Transfer.Kind.Withdrawal, "MainchainGatewayV2: invalid receipt
kind");
    require(_receipt.mainchain.chainId == block.chainid, "MainchainGatewayV2: invalid chain
```

```
id");
    require(withdrawalHash[_id] == bytes32(0), "MainchainGatewayV2: query for processed
withdrawal");
    require(!_reachedWithdrawalLimit(_tokenAddr, _quantity), "MainchainGatewayV2: reached
daily withdrawal limit"); // SKIP FOR ERC721
    // ...
}
```

RECOMMENDATION

The _submitWithdrawal function can be fixed as below.

```
function _submitWithdrawal(Transfer.Receipt calldata _receipt, Signature[] memory
_signatures)
    internal
    virtual
    returns (bool _locked)
{
    uint256 _id = _receipt.id;
    uint256 _quantity = _receipt.info.quantity;
    address _tokenAddr = _receipt.mainchain.tokenAddr;

    _receipt.info.validate();
    require(_receipt.kind == Transfer.Kind.Withdrawal, "MainchainGatewayV 2: invalid
receipt kind");
    require(_receipt.mainchain.chainId == block.chainid, "MainchainGatewayV2: invalid chain
id");
    require(withdrawalHash[_id] == bytes32(0), "MainchainGatewayV2: query for processed
withdrawal");
    require(_receipt.info.erc == Token.Standard.ERC721 ||
!_reachedWithdrawalLimit(_tokenAddr, _quantity), "MainchainGatewayV2: reached daily
withdrawal limit"); // FIXED
    // ...
}
```

UPDATES

- *May 17, 2022*: This issue has been acknowledged and fixed by Sky Mavis team in pull request *https://github.com/axieinfinity/ronin-smart-contracts-v2/pull/15*.

2.4.5. MainchainGatewayV2.test.ts - BigNumber cannot be added to JS number directly INFORMATIVE

While reviewing the MainchainGatewayV2.test.ts testcase file, we notice the following code pattern (defaultWithdrawalReceipt.info.quantity as number) + 1. In this case, the type of defaultWithdrawalReceipt.info.quantity is BigNumber which will yield the wrong result when

being added to JS numbers. The result of withdrawalReceipt.info.quantity in this testcase might not be important, however, we still highlight this issue here to prevent similar future mistakes.

```
it('Should not be able to withdraw with the signatures the other receipt', async () =>
{
  const withdrawalReceipt: ReceiptStruct = {
    ...defaultWithdrawalReceipt,
    id: BigNumber.from(1),
    info: {
      ...defaultWithdrawalReceipt.info,
      quantity: (defaultWithdrawalReceipt.info.quantity as number) + 1,
    },
  };
  await expect(gateway.submitWithdrawal(withdrawalReceipt, signatures)).revertedWith(
    'MainchainGatewayV2: query for insufficient vote weight'
  );
});
```

UPDATES

- *May 17, 2022*: This issue has been acknowledged and fixed by Sky Mavis team in pull request *https://github.com/axieinfinity/ronin-smart-contracts-v2/pull/15*.

2.4.6. Governance.sol - Missing length check for _supports and _signatures arrays INFORMATIVE

In the _relayVotesBySignatures function, the length of _supports and _signatures arrays should be checked before processing.

```
function _relayVotesBySignatures(
    Proposal.ProposalDetail memory _proposal,
    Ballot.VoteType[] calldata _supports,
    Signature[] calldata _signatures,
    bytes32 _forDigest,
    bytes32 _againstDigest
) internal {
    uint256 _forVoteCount;
    uint256 _againstVoteCount;
    address[] memory _forVoteSigners = new address[](_signatures.length);
    address[] memory _againstVoteSigners = new address[](_signatures.length);

    {
        address _signer;
        address _lastSigner;
```

```
        Ballot.VoteType _support;
        Signature memory _sig;

        for (uint256 _i; _i < _signatures.length; _i++) {
            _sig = _signatures[_i];
            _support = _supports[_i];
            // ...
        }
        // ...
    }
    // ...
}
```

RECOMMENDATION

The condition _supports.length == _signatures.length && _signatures.length > 0 should be checked before processing.

UPDATES

- *May 17, 2022*: This issue has been acknowledged and fixed by Sky Mavis team in pull request *https://github.com/axieinfinity/ronin-smart-contracts-v2/pull/15*.

2.4.7. Token.sol - Waste of gas when trying to send native tokens two times INFORMATIVE

In the handleAssetTransfer function, the contract tries to call _to.send(0) to check if the recipient can receive native tokens. And then the actual amount will be sent with _to.transfer(_info.quantity), which is quite gas-consuming.

```
function handleAssetTransfer(
    Info memory _info,
    address payable _to,
    address _token,
    IWETH _wrappedNativeToken
) internal {
    bool _success;
    if (_token == address(_wrappedNativeToken)) {
      // Check whether the `_to` address receives native token
      if (_to.send(0)) { // WASTE OF GAS
        _to.transfer(_info.quantity);
      } else {
        // Convert to wrapped token before sending them
        _wrappedNativeToken.deposit{ value: _info.quantity }();
        transfer(_info, _to, _token);
```

```
      }
    } // ...
}
```

RECOMMENDATION

These steps can be optimized with just a single statement _to.send(_info.quantity) as below.

```
function handleAssetTransfer(
    Info memory _info,
    address payable _to,
    address _token,
    IWETH _wrappedNativeToken
) internal {
    bool _success;
    if (_token == address(_wrappedNativeToken)) {
      // Check whether the `_to` address receives native token
      if (!_to.send(_info.quantity)) { // FIXED
        // Convert to wrapped token before sending them
        _wrappedNativeToken.deposit{ value: _info.quantity }();
        transfer(_info, _to, _token);
      }
    } // ...
}
```

UPDATES

- *May 17, 2022*: This issue has been acknowledged and fixed by Sky Mavis team in pull request *https://github.com/axieinfinity/ronin-smart-contracts-v2/pull/15*.

2.4.8. WithdrawalLimitation.sol - fullSigThreshold and lockedThreshold are not consistent with the description INFORMATIVE

Base on the provided documentation of Ronin Bridge, we get the following table:

Tier	Withdrawal Value	Threshold
Tier 1	-	The normal withdrawal/deposit threshold
Tier 2	> $1M	All signatures from validators are required
Tier 3	> $10M	All signatures from validators are required, one additional human review to unlock the fund

The withdrawal value is checked to be **greater than** a specified threshold. However, we notice that the withdrawal tier is changed when the withdrawal values are **greater than or equal to** the specified thresholds. Below are some usages of these thresholds in the code:

```
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);
    if (_erc == Token.Standard.ERC20) {
        if (fullSigThreshold[_token] <= _quantity) { // `<=` IS USED
            _weight = _totalWeights;
        }
        _locked = _lockedWithdrawalRequest(_token, _quantity);
    }
}

function _lockedWithdrawalRequest(address _token, uint256 _quantity) internal view virtual
returns (bool) {
    return lockedThreshold[_token] <= _quantity; // `<=` IS USED
}
```

UPDATES

- *May 17, 2022*: The documentation has been updated by Sky Mavis team in pull request *https://github.com/axieinfinity/ronin-smart-contracts-v2/pull/15*.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*May 17, 2022*	Private Report	Verichains Lab
1.1	*Jun 28, 2022*	Public Report	Verichains Lab

Table 3. Report versions history