// Private Smart Contract Security Assessment 02.16.2024 - 03.12.2024

Moonwell Cross-Chain Governance

Moonwell

HALBORN

Moonwell Cross-Chain Governance - Moonwell

Prepared by: HALBORN

Last Updated 03/16/2024

Date of Engagement by: February 16th, 2024 - March 12th, 2024

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
6	0	0	0	2	4

TABLE OF CONTENTS

1. Introduction
2. Assessment summary
3. Test approach and methodology
4. Manual testing
5. Risk methodology
6. Scope
7. Assessment summary & findings overview
8. Findings & Tech Details
    - 8.1 Cannot grant guardian role after kick or unpause
    - 8.2 Wrong event emission on `_grantguardian` function
    - 8.3 Centralization risk for trusted owners
    - 8.4 Lack of validations can brick proposal state
    - 8.5 Use custom errors
    - 8.6 Events are missing `indexed` attribute

1. Introduction

`Moonwell` engaged `Halborn` to conduct a security assessment on their smart contracts beginning on February 16th and ending on March 12th. The security assessment was scoped to the smart contracts provided in the moonwell-fi/moonwell-contracts-v2 GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

2. Assessment Summary

`Halborn` was provided 3.5 weeks for the engagement and assigned 1 full-time security engineer to review the security of the smart contracts in scope. The engineer is a blockchain and smart contract security expert with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the smart contracts.
- Ensure that smart contract functionality operates as intended.

In summary, Halborn identified some security risks that were mostly addressed by the Moonwell team.

3. Test Approach And Methodology

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the assessment:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions (`solgraph`).
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic-related vulnerability classes.
- Manual testing by custom scripts.
- Static Analysis of security for scoped contract, and imported functions (`slither`).
- Testnet deployment (`Foundry`).

4. Manual Testing

The contracts in scope were thoroughly and manually analyzed for potential vulnerabilities and bugs, as well as known optimizations and best practices when developing Smart Contracts in Solidity.

While no major vulnerabilities were found within the scope and time frame provided, it's always important to highlight good practices that were identified during the assessment, which contribute positively to the security maturity of the contracts in-scope, such as:

- Thorough documentation using NatSpec.

- Correct increment of `i` in `for` loops inside `unchecked` blocks for gas optimization.

- Pause mechanism (`ConfigurablePauseGuardian` contract and `break glass` functionality) implemented to protect the overall integrity of the ecosystem, protecting mission-critical functions to be called when the contracts are paused.

- The usage of `Ownable2Step` pattern is considered a good security practice and mitigates this risk by introducing a two-step process for ownership transfer. The current owner initiates the transfer by proposing a new owner, but the transfer only completes when the proposed new owner accepts it.

These security practices are applied industry-wide and should be considered in future implementations and developments.

5. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

5.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILIY METRIC (m_e)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2

EXPLOITABILIY METRIC (m_e)	METRIC VALUE	NUMERICAL VALUE
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

5.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (m_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

5.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9

SEVERITY	SCORE VALUE RANGE
Low	2 - 4.4
Informational	0 - 1.9

6. SCOPE

FILES AND REPOSITORY

(a) Repository: moonwell-contracts-v2

(b) Assessed Commit ID: 8e5cfe6

(c) Contracts in scope:

- src/Governance/MultichainGovernor/MultichainGovernor.sol
- src/Governance/MultichainGovernor/IMultichainGovernor.sol
- src/Governance/MultichainGovernor/MultichainVoteCollection.sol
- src/Governance/MultichainGovernor/IMultichainVoteCollection.sol
- src/stkWell/StakedWell.sol
- src/wormhole/WormholeBridgeBase.sol
- src/xWELL/ConfigurablePauseGuardian.sol

Out-of-Scope:

REMEDIATION COMMIT ID:

- 147

Out-of-Scope: New features/implementations after the remediation commit IDs.

7. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	2	4

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-04 - CANNOT GRANT GUARDIAN ROLE AFTER KICK OR UNPAUSE	Low	SOLVED - 03/13/2024

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-05 - WRONG EVENT EMISSION ON `_GRANTGUARDIAN` FUNCTION	Low	SOLVED - 03/13/2024
HAL-02 - CENTRALIZATION RISK FOR TRUSTED OWNERS	Informational	ACKNOWLEDGED - 03/12/2024
HAL-06 - LACK OF VALIDATIONS CAN BRICK PROPOSAL STATE	Informational	ACKNOWLEDGED - 03/12/2024
HAL-01 - USE CUSTOM ERRORS	Informational	ACKNOWLEDGED - 03/12/2024
HAL-03 - EVENTS ARE MISSING `INDEXED` ATTRIBUTE	Informational	ACKNOWLEDGED - 03/12/2024

8. FINDINGS & TECH DETAILS

8.1 (HAL-04) CANNOT GRANT GUARDIAN ROLE AFTER KICK OR UNPAUSE

// LOW

Description

The `ConfigurablePauseGuardian` contract, which is inherited by `MultichainGovernor` contract, implements a custom pausing mechanism, defining the `pause guardian` role, which represents the address that is allowed to call the `pause()` function within the `ConfigurablePauseGuardian` contract. Considering that `MultichainGovernor` contract inherits from `ConfigurablePauseGuardian`, we can infer from the following code snippet that a initial address is being attributed as `pause guardian` when initializing the `MultichainGovernor` contract:
- src/Governance/MultichainGovernor/MultichainGovernor.sol [Line: 204]

```
    function initialize(
        InitializeData memory initData,
        WormholeTrustedSender.TrustedSender[] memory trustedSenders,
        bytes[] calldata calldatas
    ) external initializer {

        { ... }

        /// set the pause guardian
        _grantGuardian(initData.pauseGuardian);

        { ... }
    }
```

The `_grantGuardian` internal function in `ConfigurablePauseGuardian` contract will set the `newPauseGuardian` address, as follows.
- src/xWELL/ConfigurablePauseGuardian.sol [Line: 93]

```
    function _grantGuardian(address newPauseGuardian) internal {
        address previousPauseGuardian = newPauseGuardian;
        pauseGuardian = newPauseGuardian;

        /// if a new guardian is granted, the contract is automatically
unpaused
        _setPauseTime(0);
```

```
        emit PauseGuardianUpdated(previousPauseGuardian, newPauseGuardian);
    }
```

Moving forward, it is important to highlight that whenever we call the functions `unpause` or `kickGuardian`, the `_resetPauseState` internal function is called, which, from other things, **sets the pauseGuardian address to the** `address(0)`.
- src/xWELL/ConfigurablePauseGuardian.sol [Line: 47]

```
    function _resetPauseState() private {
        address previousPauseGuardian = pauseGuardian;

        pauseGuardian = address(0); /// remove the pause guardian

        _setPauseTime(0); /// fully unpause, set pauseStartTime to 0

        emit PauseGuardianUpdated(previousPauseGuardian, address(0));
    }
```

While effectively validating whether the caller of the `pause` function is indeed authorized to perform the pause operation (by checking if the `msg.sender == pauseGuardian`), and by implementing a safety mechanism such as `kickGuardian` to avoid unintended entities to stay as `pause guardians` , the current implementation of `MultichainGovernor` and `ConfigurablePauseGuardian` does not expose any `public` or `external` method, not even access-controlled, to grant guardian role to new addresses.
In other words, this means that after `unpause` or `kickGuardian` functions are called, effectively changing the `pauseGuardian` to `address(0)`, it is impossible to set a new `pauseGuardian` because the only time the `_grantGuardian` function is called is within the initialization of the `MultichainGovernor` contract. This will prevent the `MultichainGovernor` contract from setting new `pause guardians` addresses.

BVSS

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:C (3.1)

Recommendation

It is recommended to add an access-controlled external function to the `MultichainGovernor` contract, that will enable setting the `pauseGuardian` to a new address, assuring that the `pauseGuardian` value is not bricked to `address(0)` after calling `kickGuardian` or `unpause` functions.

Remediation Plan

SOLVED: The **Moonwell team** has solved this issue by creating an external, access-controlled function `grantPauseGuardian` in the `MultichainGovernor` contract.

Remediation Hash

https://github.com/moonwell-fi/moonwell-contracts-v2/pull/147

8.2 (HAL-05) WRONG EVENT EMISSION ON `_GRANTGUARDIAN` FUNCTION

// LOW

Description

It is important to highlight the `_grantGuardian` internal function in `ConfigurablePauseGuardian` contract as follows:
- src/xWELL/ConfigurablePauseGuardian.sol [Line: 93]

```
    function _grantGuardian(address newPauseGuardian) internal {
        address previousPauseGuardian = newPauseGuardian;
        pauseGuardian = newPauseGuardian;

        /// if a new guardian is granted, the contract is automatically
unpaused
        _setPauseTime(0);

        emit PauseGuardianUpdated(previousPauseGuardian, newPauseGuardian);
    }
```

We can observe that the event `PauseGuardianUpdated` is emitted with the same values for `oldPauseGuardian` and `newPauseGuardian`. This happens because the value attributed for `address previousPauseGuardian` is `newPauseGuardian` instead of `pauseGuardian`.
The `MultichainGovernor` contract is also impacted because it inherits from `ConfigurablePauseGuardian`.

BVSS

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:C (3.1)

Recommendation

It is recommended to modify the `_grantGuardian` function in `ConfigurablePauseGuardian` as follows:

```
    function _grantGuardian(address newPauseGuardian) internal {
        address previousPauseGuardian = pauseGuardian;
        pauseGuardian = newPauseGuardian;

        /// if a new guardian is granted, the contract is automatically
unpaused
        _setPauseTime(0);
```

```
        emit PauseGuardianUpdated(previousPauseGuardian, newPauseGuardian);
    }
```

Remediation Plan

SOLVED: The **Moonwell team** has solved this issue as recommended in the provided code snippet.

Remediation Hash

https://github.com/moonwell-fi/moonwell-contracts-v2/pull/147

8.3 (HAL-02) CENTRALIZATION RISK FOR TRUSTED OWNERS

// INFORMATIONAL

Description

The smart contracts under analysis have owners with privileged rights to perform administrative operations and need to be trusted not to act maliciously.
- src/Governance/MultichainGovernor/MultichainVoteCollection.sol [Line: 371]

```
        function setGasLimit(uint96 newGasLimit) external onlyOwner {
```

- src/Governance/MultichainGovernor/MultichainVoteCollection.sol [Line: 382]

```
        function setNewStakedWell(address newStakedWell) external onlyOwner
{
```

BVSS

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:P/S:C (1.6)

Recommendation

It is recommended to mitigate centralization issues by implementing multi-signature mechanisms. This prevents a single entity from performing administrative and protected tasks unilaterally.

Remediation Plan

ACKNOWLEDGED: The **Moonwell team** has acknowledged the issue and informed that potential centralization concerns are mitigated because in this case, the `owner` of the contract is the `Temporal Governor`.

8.4 (HAL-06) LACK OF VALIDATIONS CAN BRICK PROPOSAL STATE

// INFORMATIONAL

Description

In the `MultichainGovernor` contract, proposals can be executed if they were previously succeeded, with arbitrary `calldatas`, `values` and `targets` contract addresses provided by the creator when calling the `propose` function, accordingly to the current implemented logic.
- src/Governance/MultichainGovernor.sol [Line: 721]

```
    function execute(
        uint256 proposalId
    ) external payable override whenNotPaused {
        /// Checks
        require(
            state(proposalId) == ProposalState.Succeeded,
            "MultichainGovernor: proposal can only be executed if it is
Succeeded"
        );

        uint256 totalValue = 0;

        Proposal storage proposal = proposals[proposalId];

        for (uint256 i = 0; i < proposal.targets.length; ) {
            totalValue += proposal.values[i];
            unchecked {
                i++;
            }
        }

        require(totalValue == msg.value, "MultichainGovernor: invalid
value");

        /// Effects

        proposal.executed = true;

        /// remove the proposal that is about to be executed from all
proposals,
        /// and remove from inactive proposals from user list
```

```
        _syncTotalLiveProposals();

        /// Interactions

        unchecked {
            for (uint256 i = 0; i < proposal.targets.length; i++) {
                proposal.targets[i].functionCallWithValue(
                    proposal.calldatas[i],
                    proposal.values[i],
                    "MultichainGovernor: execute call failed"
                );
            }
        }

        emit ProposalExecuted(proposalId);
    }
```

There are no verifications in place that validate whether the values provided by the creator when calling the `propose` function are legitimate. Likewise, there are no such verifications on the `execute` function, what could lead to malformed calls being performed to `targets`.
Moving forward, the call to `targets[i].functionCallWithValue` will revert in cases the constructed calldatas and values are not valid for that specific target. This could lead to a scenario where the proposal status is set to, but`succeeded` can never be set to `executed` because of the failed calls.

BVSS

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:P/S:C (1.6)

Recommendation

It is recommended to perform validation on user-provided data (inputs) when the function `propose` is called. A whitelist mechanism, similar to the one applied to the `break glass functionality` would be a good starting point.
By whitelisting possible `calldatas` and `targets`, preferably, using mappings, it is possible to ensure that only valid and authorized arguments are passed when calling `functionCallWithValue`, making it less likely to revert.
Alternatively, enhancing the proposal state management, for example, adding a `bool executionFailed` element to the `Proposal` struct or adding `ExecutionFailed` to the `ProposalState` enum, both in the `IMultichainGovernor` could improve the handling of such scenarios. Moving in this direction, it would also be possible to modify the `cancel` function to handle these scenarios appropriately.

Remediation Plan

ACKNOWLEDGED: The **Moonwell team** accepted the risk in benefit of the contract being as future-proof as possible, as restricting that `calldata` only interacts with an allowed list of contracts, or must be

executable/would not revert on propose, would be too restrictive and inflexible. The team mentioned it's an intended behavior because the additional contracts and calls the governor might need to make in the future are yet unknown.

Remediation Hash

https://github.com/moonwell-fi/moonwell-contracts-v2/pull/147

8.5 (HAL-01) USE CUSTOM ERRORS

// INFORMATIONAL

Description

In Solidity smart contract development, replacing hard-coded revert message strings with the `Error()` syntax is an optimization strategy that can significantly reduce gas costs. Hard-coded strings, stored on the blockchain, increase the size and cost of deploying and executing contracts.
The `Error()` syntax allows for the definition of reusable, parameterized custom errors, leading to a more efficient use of storage and reduced gas consumption. This approach not only optimizes gas usage during deployment and interaction with the contract but also enhances code maintainability and readability by providing clearer, context-specific error information.

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (1.0)

Recommendation

It is recommended to replace hard-coded revert strings in `require` statements for custom errors, which can be done following the logic below.
1. Standard require statement (to be replaced):

```
require(condition, "Condition not met");
```

2. Declare the error definition to state:

```
error ConditionNotMet();
```

3. As currently is not possible to use custom errors in combination with `require` statements, the standard syntax is:

```
if (!condition) revert ConditionNotMet();
```

More information about this topic in Official Solidity Documentation.

Remediation Plan

ACKNOWLEDGED: The **Moonwell team** has acknowledged this finding and has opted not to perform modifications as an style decision.

Remediation Hash

https://github.com/moonwell-fi/moonwell-contracts-v2/pull/147

8.6 (HAL-03) EVENTS ARE MISSING `INDEXED` ATTRIBUTE

// INFORMATIONAL

Description

Indexed event fields make the data more quickly accessible to off-chain tools that parse events, and adds them to a special data structure known as “topics” instead of the data part of the log. A topic can only hold a single word (32 bytes) so if you use a reference type for an indexed argument, the Keccak-256 hash of the value is stored as a topic instead.

Each event can use up to three indexed fields. If there are fewer than three fields, all of the fields can be indexed. It is important to note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed fields per event (three indexed fields).

This is specially recommended when gas usage is not particularly of concern for the emission of the events in question, and the benefits of querying those fields in an easier and straight-forward manner surpasses the downsides of gas usage increase.

- src/Governance/MultichainGovernor/IMultichainGovernor.sol [Line: 15]

```
        event StartBlockSet(uint256 proposalId, uint256 startBlock);
```

- src/Governance/MultichainGovernor/IMultichainGovernor.sol [Line: 18]

```
        event VoteCast(
```

- src/Governance/MultichainGovernor/IMultichainGovernor.sol [Line: 26]

```
        event ProposalCreated(
```

- src/Governance/MultichainGovernor/IMultichainGovernor.sol [Line: 38]

```
        event ProposalCanceled(uint256 id);
```

- src/Governance/MultichainGovernor/IMultichainGovernor.sol [Line: 41]

```
        event ProposalQueued(uint256 id, uint256 eta);
```

- src/Governance/MultichainGovernor/IMultichainGovernor.sol [Line: 44]

```
        event ProposalExecuted(uint256 id);
```

- src/Governance/MultichainGovernor/IMultichainGovernor.sol [Line: 47]

```
        event BreakGlassExecuted(
```

- src/Governance/MultichainGovernor/IMultichainGovernor.sol [Line: 54]

```
        event QuroumVotesChanged(uint256 oldValue, uint256 newValue);
```

- src/Governance/MultichainGovernor/IMultichainGovernor.sol [Line: 57]

```
        event ProposalThresholdChanged(uint256 oldValue, uint256 newValue);
```

- src/Governance/MultichainGovernor/IMultichainGovernor.sol [Line: 60]

```
        event VotingPeriodChanged(uint256 oldValue, uint256 newValue);
```

- src/Governance/MultichainGovernor/IMultichainGovernor.sol [Line: 63]

```
        event BreakGlassGuardianChanged(address oldValue, address
    newValue);
```

- src/Governance/MultichainGovernor/IMultichainGovernor.sol [Line: 66]

```
        event GovernanceReturnAddressChanged(address oldValue, address
    newValue);
```

- src/Governance/MultichainGovernor/IMultichainGovernor.sol [Line: 69]

```
        event CrossChainVoteCollectionPeriodChanged(
```

- src/Governance/MultichainGovernor/IMultichainGovernor.sol [Line: 75]

```
        event UserMaxProposalsChanged(uint256 oldValue, uint256 newValue);
```

- src/Governance/MultichainGovernor/IMultichainGovernor.sol [Line: 83]

```
        event CrossChainVoteCollected(
```

- src/Governance/MultichainGovernor/IMultichainGovernor.sol [Line: 94]

```
        event CalldataApprovalUpdated(bytes data, bool approved);
```

- src/Governance/MultichainGovernor/IMultichainGovernor.sol [Line: 99]

```
        event ProposalRebroadcasted(uint256 proposalId, bytes data);
```

- src/Governance/MultichainGovernor/IMultichainGovernor.sol [Line: 102]

```
        event NewStakedWellSet(address newStakedWell, bool
 toUseTimestamps);
```

- src/Governance/MultichainGovernor/IMultichainVoteCollection.sol [Line: 17]

```
        event ProposalCreated(
```

- src/Governance/MultichainGovernor/IMultichainVoteCollection.sol [Line: 29]

```
        event VotesEmitted(
```

- src/Governance/MultichainGovernor/IMultichainVoteCollection.sol [Line: 41]

```
        event VoteCast(
```

- src/Governance/MultichainGovernor/IMultichainVoteCollection.sol [Line: 49]

```
        event NewStakedWellSet(address newStakedWell);
```

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:P/S:C (0.0)

Recommendation

It is recommended to add the `indexed` keyword when declaring events, considering the following example:

```
    event Indexed(
        address indexed from,
        bytes32 indexed id,
        uint indexed value
    );
```

Remediation Plan

ACKNOWLEDGED: The **Moonwell team** has acknowledged the finding.

Remediation Hash

https://github.com/moonwell-fi/moonwell-contracts-v2/pull/147

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.