// Private Smart Contract Security Assessment

07.30.2024 - 08.12.2024

Liquid Staking Cadence & EVM

Ankr

HALBORN

Report a Bug

Liquid Staking Cadence & EVM - Ankr

Prepared by: HALBORN Last Updated 08/26/2024 Date of Engagement by: July 30th, 2024 - August 12th, 2024

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
12	1	1	1	4	5

1. Introduction

Ankr engaged our security analysis team to conduct a comprehensive security audit of their smart contract ecosystem. The primary aim was to meticulously assess the security architecture of the smart contracts to pinpoint vulnerabilities, evaluate existing security protocols, and offer actionable insights to bolster security and operational efficacy of their smart contract framework. Our assessment was strictly confined to the smart contracts provided, ensuring a focused and exhaustive analysis of their security features.

TABLE OF CONTENTS

2. Assessment Summary

Our engagement with Ankr spanned a 1-week period, during which we dedicated one full-time security engineer equipped with extensive experience in blockchain security, advanced penetration testing capabilities, and profound knowledge of various blockchain protocols. The objectives of this assessment were to:

- Verify the correct functionality of smart contract operations.

- Identify potential security vulnerabilities within the smart contracts.

- Provide recommendations to enhance the security and efficiency of the smart contracts.

3. Test Approach And Methodology

Our testing strategy employed a blend of manual and automated techniques to ensure a thorough evaluation. While manual testing was pivotal for uncovering logical and implementation flaws, automated testing offered broad code coverage and rapid identification of common vulnerabilities. The testing process included:

- A detailed examination of the smart contracts' architecture and intended functionality.

- Comprehensive manual code reviews and walkthroughs.

- Functional and connectivity analysis utilizing tools like Solgraph.

- Customized script-based manual testing and testnet deployment using Foundry.

This executive summary encapsulates the pivotal findings and recommendations from our security assessment of Ankr smart contract ecosystem. By addressing the identified issues and implementing the recommended fixes, Ankr can significantly boost the security, reliability, and trustworthiness of its smart contract platform.

4. 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.

4.1 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
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:

$$E = \prod m_e$$

4.2 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

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
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:

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

4.3 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
Low	2 - 4.4
Informational	0 - 1.9

5. SCOPE

FILES AND REPOSITORY

(a) Repository: stakefi-smart-contract

(b) Assessed Commit ID: 6b06658

(c) Items in scope:

- cadence/contracts/StakingManager_v0_0_2.cdc
- cadence/transactions/owner/initStakingCollection.cdc
- cadence/transactions/admin/transferAdmin.cdc
- cadence/transactions/admin/registerDelegator.cdc
- cadence/transactions/admin/unpauseStakingManager.cdc
- cadence/transactions/pauser/claimPauserCapability.cdc
- cadence/transactions/pauser/pauseStakingManager.cdc
- cadence/transactions/operator/claimOperatorCapability.cdc
- cadence/transactions/operator/unstake.cdc
- cadence/transactions/operator/restakeRewards.cdc
- cadence/transactions/operator/transferAndStake.cdc
- cadence/transactions/operator/stake.cdc
- solidity/contracts/tokens/AnkrRatioFeed.sol
- solidity/contracts/FlowStakingPool.sol

Out-of-Scope:

REMEDIATION COMMIT ID:

- 7b80e2e

- 96bbcb4
- d51f80b
- 845f8d6

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

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
1	1	1	4

INFORMATIONAL
5

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
UNVALIDATED EVM EXECUTION RESULT IN TRANSFERUNSTAKEDTOEVM	CRITICAL	SOLVED - 07/31/2024
INCORRECT RESOURCE HANDLING IN TRANSFERADMIN TRANSACTION	HIGH	SOLVED - 07/31/2024

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
LACK OF DUPLICATION CHECK IN ADDDELEGATION FUNCTION	MEDIUM	SOLVED - 07/31/2024
HARDCODED VALUES IN REGISTERDELEGATOR TRANSACTION	LOW	SOLVED - 08/15/2024
MISSING INITIALIZATION IN FLOWSTAKINGPOOL CONTRACT	LOW	SOLVED - 07/31/2024
UNUSED PUBLISHPAUSERCAPABILITY FUNCTION IN STAKINGMANAGER CONTRACT	LOW	SOLVED - 08/15/2024
POTENTIAL GAS LIMIT ISSUES IN TRANSFERUNSTAKEDTOEVM	LOW	SOLVED - 08/15/2024
UNUSED PARAMETER IN STAKER RESOURCE INITIALIZATION	INFORMATIONAL	SOLVED - 08/15/2024
INEFFICIENT INITIALIZATION PROCESS IN STAKINGMANAGER CONTRACT	INFORMATIONAL	SOLVED - 08/15/2024

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
POTENTIAL FAILURE IN DECODING ETHEREUM ADDRESSES	INFORMATIONAL	SOLVED - 08/15/2024
OVERLY PERMISSIVE ADMIN RESOURCE ACCESS	INFORMATIONAL	ACKNOWLEDGED
OVERLY PERMISSIVE MUTABILITY FOR STAKINGCONTRACTEVM	INFORMATIONAL	SOLVED - 08/15/2024

7. FINDINGS & TECH DETAILS

7.1 UNVALIDATED EVM EXECUTION RESULT IN TRANSFERUNSTAKEDTOEVM

// CRITICAL

Description

In the `TransferUnstakedToEVM` function of the `StakingManager` contract, the EVM execution result from the `ownerCOA.call()` is not validated. This oversight could lead to silent failures where the EVM execution fails, but the function continues as if it succeeded. Such behavior could result in inconsistencies between the Cadence and EVM sides of the system, potentially leading to loss of funds or incorrect accounting of unstaked tokens.

BVSS

AO:A/AC:L/AX:L/C:N/I:H/A:N/D:N/Y:N/R:N/S:C (9.4)

Recommendation

Modify the `TransferUnstakedToEVM` function to check the status of the EVM execution result. After the `ownerCOA.call()`, add a condition to verify that `Result.status == successful`. If the status is not successful, the function should revert the transaction and provide an error message. This change will ensure that any failures in the EVM execution are caught and handled appropriately, maintaining the consistency and integrity of the cross-chain staking operations.

Remediation Progress

SOLVED: The call result status is now verified.

Remediation Hash

7b80e2e480a0a9b64f2b6fba7a74b3fb135690ce

7.2 INCORRECT RESOURCE HANDLING IN TRANSFERADMIN TRANSACTION

// HIGH

Description

The `transferAdmin` transaction in the `StakingManager` contract contains critical errors in resource handling. First, it incorrectly uses `=` instead of `<-` when loading the Admin resource, which violates Cadence's resource management rules. Second, the post-condition attempts to access the resource after it has been moved, which is invalid and will fail to compile or execute.

BVSS

AO:A/AC:L/AX:L/C:N/I:C/A:C/D:N/Y:N/R:P/S:C (7.8)

Recommendation

Revise the `transferAdmin` transaction to correctly move the Admin resource using `<-`. For the post-condition, obtain a reference to the stored resource after moving it. This ensures proper resource management and allows for valid post-condition checks. Consider removing the post-condition if it's not essential, as the successful execution of the resource move implicitly confirms the transfer. These changes will correct the resource handling, prevent potential loss of the Admin resource, and ensure the transaction's validity and safety.

Remediation Progress

SOLVED: The code now transfers the resource as expected

Remediation Hash

7b80e2e480a0a9b64f2b6fba7a74b3fb135690ce

7.3 LACK OF DUPLICATION CHECK IN ADDDELEGATION FUNCTION

// MEDIUM

Description

The `AddDelegation` function in the `StakingManager` contract's `Admin` resource lacks a crucial duplication check for the `_stakingNodeId` and `_delegatorId` parameters. This oversight allows for the addition of duplicate delegation entries, which could lead to several issues:

1. Inflated delegation counts
2. Inconsistent state management
3. Potential for erroneous calculations or distributions based on duplicate entries
4. Increased gas costs for unnecessary storage

The absence of this check could result in operational inefficiencies and potentially compromise the integrity of the staking system.

BVSS

AO:A/AC:L/AX:L/C:N/I:H/A:M/D:N/Y:N/R:P/S:C (5.1)

Recommendation

Implement a duplication check in the `AddDelegation` function before adding new entries. This check should verify that no existing delegation combines the same `_stakingNodeId` and `_delegatorId`. Consider using a composite key or a mapping to efficiently track existing delegations. If a duplicate is found, the function should either revert the transaction or return an error code, depending on the desired behavior. This change will ensure the uniqueness of delegations, maintain data integrity, and prevent potential issues arising from duplicate entries.

Remediation Progress

SOLVED: A new function `isEqual` was added and used as a filter for the delegations list. If the resulting list after filtering is none zero, the transaction will assert. The `isEqual` function does verify

that both `_stakingNodeId` and `_delegatorId` are not present.

Remediation Hash

7b80e2e480a0a9b64f2b6fba7a74b3fb135690ce

7.4 HARDCODED VALUES IN REGISTERDELEGATOR TRANSACTION

// LOW

Description

The `registerDelegator` transaction in the `StakingManager` contract uses hardcoded values for `nodeID` and `amount` when calling the `registerDelegator` function. This approach lacks flexibility and prevents users from specifying their desired node and delegation amount. It could lead to incorrect delegations, limit the contract's functionality, and potentially cause users to delegate to unintended nodes or with unintended amounts.

BVSS

AO:A/AC:L/AX:L/C:N/I:M/A:L/D:M/Y:L/R:P/S:U (3.8)

Recommendation

Modify the `registerDelegator` transaction to accept `nodeID` and `amount` as parameters. Update the function call within the transaction to use these parameters instead of hardcoded values. This change allows users to specify their intended node and delegation amount, providing the necessary flexibility for various delegation scenarios. It will also enhance the transaction's reusability and reduce the risk of unintended delegations. Ensure to add appropriate input validation for these new parameters to maintain the security and integrity of the delegation process.

Remediation Progress

SOLVED: The transaction now accepts a parameter.

Remediation Hash

96bbcb4a375b6dac63bc0378d1fbf76cd862606b

7.5 MISSING INITIALIZATION IN FLOWSTAKINGPOOL CONTRACT

// LOW

Description

In the `FlowStakingPool` contract, there is a critical oversight in the initialization process. The contract fails to call the `__QueuePool_init` function during its initialization. This omission has significant implications, particularly for the `_DISTRIBUTE_GAS_LIMIT` variable, which is typically set during this initialization step.

The absence of this initialization means that the `_DISTRIBUTE_GAS_LIMIT` will not be set to its default value. This limit is crucial for controlling gas consumption during the distribution of pending rewards, which is a key functionality of the staking pool. Without a properly set gas limit, the contract may encounter issues when attempting to distribute rewards, potentially leading to failed transactions or unexpected behavior.

This oversight forces the governance to manually intervene by calling the `setDistributeGasLimit` function post-deployment to set the appropriate gas limit. This not only introduces an additional step in the deployment process but also opens up a window of vulnerability where the contract could be in an inconsistent state if this step is forgotten or delayed.

BVSS

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

Recommendation

To address this issue, modify the `initialize` function in the `FlowStakingPool` contract to include a call to `__QueuePool_init`. This ensures that all necessary initializations, including setting the `_DISTRIBUTE_GAS_LIMIT`, are performed during contract deployment.

Remediation Progress

SOLVED: The `distributeGasLimit` parameter was added and used to initialize the `__QueuePool_init`.

Remediation Hash

`d51f80b8ab251ded621866b7bbec18f63d01ec06`

7.6 UNUSED PUBLISHPAUSERCAPABILITY FUNCTION IN STAKINGMANAGER CONTRACT

// LOW

Description

The `StakingManager` contract contains a `PublishPauserCapability` function that is defined but never called within the contract. This function is designed to issue a pauser capability and publish it to a specified address. However, its lack of usage raises concerns about the contract's design and functionality.
Unused functions in smart contracts can lead to several issues:
1. Code bloat: They increase the contract's size without providing any benefit, potentially leading to higher deployment costs.
2. Confusion: They may mislead developers or auditors about the contract's actual functionality.
3. Potential security risks: Unused functions might be accidentally used in future updates, leading to unintended consequences.
4. Maintenance overhead: They require ongoing consideration during code reviews and updates, despite not contributing to the contract's operations.
The presence of this unused function suggests that either a planned feature was not fully implemented, or that the contract's pause functionality is being managed differently than originally intended.

BVSS

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

Recommendation

To address this issue, consider the following options:
1. If the pauser capability is intended to be used, implement the necessary logic to call `PublishPauserCapability` at the appropriate time, such as during contract initialization or as part of an admin function.

2. If the function is no longer needed, remove it entirely from the contract. This will simplify the codebase and reduce potential confusion.
Before making any changes, carefully review the contract's requirements and design to ensure that removing or modifying this function won't impact any planned future functionality. If the function is removed, ensure that any related events or state variables are also cleaned up to maintain consistency throughout the contract.
By addressing this unused function, you'll improve the contract's clarity, reduce its complexity, and minimize potential sources of confusion or future errors.

Remediation Progress

SOLVED: The `AddPauser` function is now calling the `StakingManager.PublishPauserCapability` directly.

Remediation Hash

96bbcb4a375b6dac63bc0378d1fbf76cd862606b

7.7 POTENTIAL GAS LIMIT ISSUES IN TRANSFERUNSTAKEDTOEVM

// LOW

Description

The `TransferUnstakedToEVM` function in the `StakingManager` contract uses a hardcoded gas limit of 10000 for the EVM call. This approach lacks flexibility and may not account for variations in gas costs or the base transaction cost. It fails to consider that EVM transactions typically have a fixed base cost of 21000 gas, which may or may not be included in the `gasLimit` parameter for Cadence EVM calls.

BVSS

AO:A/AC:L/AX:L/C:N/I:H/A:M/D:L/Y:L/R:F/S:U (2.5)

Recommendation

Modify the `TransferUnstakedToEVM` function to accept a `gasLimit` parameter. Implement pre-execution checks to ensure the provided gas limit is sufficient for the operation, including any base transaction costs. Conduct thorough testing on Cadence EVM to determine if the base 21000 gas is included in the `gasLimit` parameter. Update the function to dynamically calculate or adjust the gas limit based on these findings, ensuring it covers all necessary costs while remaining efficient. This approach will provide better control over gas usage and improve the function's adaptability to different network conditions.

Remediation Progress

SOLVED: The `TransferUnstakedToEVM` function does now accept a `gasLimit` parameter. The call will revert in case the transaction uses more than the specified gas on the EVM side and with the verification on the call result the entire cadence transaction revert too.

Remediation Hash

96bbcb4a375b6dac63bc0378d1fbf76cd862606b

7.8 UNUSED PARAMETER IN STAKER RESOURCE INITIALIZATION

// INFORMATIONAL

Description

In the `StakingManager` contract, the `Staker` resource's initializer function takes an `operator` parameter that is not used within the function body.
The `operator` parameter is declared but never utilized within the function. This unused parameter introduces unnecessary complexity to the contract's interface and could lead to confusion for developers interacting with or maintaining the contract. It may also mislead users into thinking that the operator address is being used or stored when it actually isn't.
While this doesn't pose an immediate security risk, it's a code quality issue that could indirectly lead to misunderstandings or errors in the future. It violates the principle of least astonishment and potentially impacts the contract's maintainability.

BVSS

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:P/S:U (1.3)

Recommendation

To address this issue, consider one of the following approaches:
1. If the `operator` parameter is truly not needed for the `Staker` resource initialization, remove it from the function signature.
2. If the `operator` address should be stored or used within the `Staker` resource, add the necessary logic to utilize it.
By implementing one of these changes, you'll improve the clarity and maintainability of the contract. It will also ensure that the contract's interface accurately reflects its actual behavior and requirements. This change will reduce potential confusion for future developers and users interacting with the `StakingManager` contract.

Remediation Progress

SOLVED: The parameter was removed.

Remediation Hash

96bbcb4a375b6dac63bc0378d1fbf76cd862606b

7.9 INEFFICIENT INITIALIZATION PROCESS IN STAKINGMANAGER CONTRACT

// INFORMATIONAL

Description

The current implementation of the `StakingManager` contract and its initialization process in the `initStakingCollection` transaction has several inefficiencies and potential security issues:
1. The `stakingManagerCodeHex` is required to be passed as a parameter in the `initStakingCollection` transaction. This approach is problematic because it requires the contract code to be available at the time of initialization, which may not always be the case and could lead to deployment complexities.
2. The creation of essential resources like `StakingCollection` and `CadenceOwnedAccount` is not performed during the contract's initialization. Instead, these resources are expected to exist beforehand or are created in a separate step. This multistep initialization process increases the risk of incomplete setup and potential vulnerabilities if all steps are not executed correctly.
3. The current design doesn't take full advantage of the fact that the contract creator (who will be the admin) has the authority to create these resources during the initialization phase.
These issues collectively make the deployment and setup process more complex, error-prone, and potentially insecure. It also increases the chances of the contract being in an inconsistent state if all initialization steps are not completed correctly.

BVSS

AO:S/AC:L/AX:L/C:N/I:H/A:M/D:N/Y:N/R:P/S:C (1.1)

Recommendation

To address these issues and improve the overall security and efficiency of the `StakingManager` contract, implement the following changes:
Move the creation of `StakingCollection` and other essential resources into the contract's initializer.

Remediation Progress

SOLVED: The initialization process was moved to the `StakingManager` instead of relaying on an initialization transaction.

Remediation Hash

`96bbcb4a375b6dac63bc0378d1fbf76cd862606b`

7.10 POTENTIAL FAILURE IN DECODING ETHEREUM ADDRESSES

// INFORMATIONAL

Description

The `StakingManager` contract's initialization process uses a manual method to decode the `stakingContractEvmHex` into a byte array. This approach may fail if the input includes the '0x' prefix common in Ethereum addresses. The current implementation doesn't account for this prefix, potentially causing decoding errors or invalid address interpretations.

BVSS

AO:A/AC:L/AX:L/C:N/I:L/A:L/D:N/Y:N/R:F/S:U (0.8)

Recommendation

Replace the manual decoding process with the `EVM.addressFromString` function. This function handles the '0x' prefix gracefully and provides robust address conversion. Update the `Staker` resource to accept an `EVM.EVMAddress` instead of a raw byte array. This change will improve the contract's robustness in handling various address input formats, reduce the risk of decoding errors, and simplify the initialization process. It also aligns the contract more closely with standard Ethereum address handling practices, enhancing its interoperability and user-friendliness.

Remediation Progress

SOLVED: The entire codebase is now using `EVM.ddressFromString`.

Remediation Hash

96bbcb4a375b6dac63bc0378d1fbf76cd862606b

7.11 OVERLY PERMISSIVE ADMIN RESOURCE ACCESS

// INFORMATIONAL

Description

The `Admin` resource in the `StakingManager` contract allows the resource owner to call the `Pause` method without proper entitlement restrictions. This oversight could lead to unintended pausing of the contract by the admin, bypassing the intended access control mechanism.

BVSS

AO:S/AC:L/AX:L/C:N/I:M/A:L/D:N/Y:N/R:F/S:U (0.3)

Recommendation

When interacting with the `Admin` resource, use entitled references instead of accessing the entire resource. Specifically, use `auth(StakingManager.StakingAdmin) &StakingManager.Admin` when borrowing the resource. This ensures that only methods authorized for `StakingAdmin` can be called, preventing unauthorized access to sensitive functions like `Pause`. Additionally, always use `borrow` instead of `load` when accessing the resource in transactions to maintain proper access control and prevent unintended privilege escalation.

Remediation Progress

ACKNOWLEDGED: The **Ankr team** acknowledged this finding.

7.12 OVERLY PERMISSIVE MUTABILITY FOR STAKINGCONTRACTEVM

// INFORMATIONAL

Description

In the `StakingManager` contract, the `StakingContractEvm` field within the `Staker` resource is declared as a variable (`var`) instead of a constant (`let`). The current declaration is:

```
access(all) var StakingContractEvm: EVM.EVMAddress
```

While the access control mechanisms in place prevent unauthorized modifications to this field, declaring it as mutable when it's not intended to be changed introduces unnecessary flexibility. This could potentially lead to confusion or unintended modifications in future development or maintenance of the contract.
Although there isn't an immediate security risk due to the access restrictions in place, following the principle of least privilege and immutability where possible is a best practice in smart contract development. It helps prevent potential bugs and makes the code's intentions clearer.

BVSS

AO:S/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:F/S:U (0.1)

Recommendation

If there are no plans to change the `StakingContractEvm` address after initialization, it's recommended to declare it as a constant using the `let` keyword. This change would more accurately reflect the intended immutability of the EVM address and prevent any accidental modifications in the future.

Remediation Progress

SOLVED: The intended behavior is to allow changing the `StakingContractEvm` in case it is required. The code changes add an entitled `UpdateStakingContractEvm` function to `StakingAdmin` and a contract only function on the `Staker` resource named `SetStakingContractEvm`. The entitled function does call.

Remediation Hash

`845f8d6196b9a5277e9031cc8755ee739976e2e4`

8. REVIEW NOTES

Transactions Review

Transaction	Details
cadence/transactions/owner/initStakingCollection.cdc	No issues. But most of the funcionality could be moved into the `StakingManager` initialisation.
cadence/transactions/admin/transferAdmin.cdc	Invalid usage of resource `<-` / `=`. Reported as issue.
cadence/transactions/admin/registerDelegator.cdc	`nodeID` and `amount` should be a parameter.
cadence/transactions/admin/unpauseStakingManager.cdc	No issues.
cadence/transactions/pauser/claimPauserCapability.cdc	No issues.
cadence/transactions/pauser/pauseStakingManager.cdc	No issues.
cadence/transactions/operator/claimOperatorCapability.cdc	No issues.
cadence/transactions/operator/unstake.cdc	No issues.
cadence/transactions/operator/restakeRewards.cdc	No issues.
cadence/transactions/operator/transferAndStake.cdc	No issues.
cadence/transactions/operator/stake.cdc	No issues.

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.