PacNetwork

Security Assessment

CertiK Assessed on Oct 8th, 2025

CertiK Assessed on Oct 8th, 2025

PacNetwork

The security assessment was prepared by CertiK.

Executive Summary

TYPES	ECOSYSTEM	METHODS
StableCoin	EVM Compatible	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE
Solidity	Preliminary comments published on 08/29/2025 Final report published on 10/09/2025

Vulnerability Summary

16	7	0	9	0
Total Findings	Resolved	Partially Resolved	Acknowledged	Declined

2	Centralization	2 Acknowledged	Centralization findings highlight privileged roles & functions and their capabilities, or instances where the project takes custody of users' assets.
0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
0	Major		Major risks may include logical errors that, under specific circumstances, could result in fund losses or loss of project control.
1	Medium	1 Acknowledged	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
9	Minor	6 Resolved, 3 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
4	Informational	1 Resolved, 3 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | PACNETWORK

CODEBASE | PACNETWORK

Repository

PacNetwork Contracts

Commit

- 0dd9f84a930b420bc37d418dac6ecc093641c1ca
- 84090e345d12bc6c46722558a481d00c350e0b32
- fab6783757a52c50b32d723e5aa4b8ae9366fb5a

Audit Scope

The file in scope is listed in the appendix.

APPROACH & METHODS | PACNETWORK

This audit was conducted for PacNetwork to evaluate the security and correctness of the smart contracts associated with the PacNetwork project. The assessment included a comprehensive review of the in-scope smart contracts. The audit was performed using a combination of Formal Verification, Manual Review, and Static Analysis.

The review process emphasized the following areas:

- Architecture review and threat modeling to understand systemic risks and identify design-level flaws.
- Identification of vulnerabilities through both common and edge-case attack vectors.
- Manual verification of contract logic to ensure alignment with intended design and business requirements.
- Dynamic testing to validate runtime behavior and assess execution risks.
- Assessment of code quality and maintainability, including adherence to current best practices and industry standards.

The audit resulted in findings categorized across multiple severity levels, from informational to critical. To enhance the project's security and long-term robustness, we recommend addressing the identified issues and considering the following general improvements:

- Improve code readability and maintainability by adopting a clean architectural pattern and modular design.
- Strengthen testing coverage, including unit and integration tests for key functionalities and edge cases.
- Maintain meaningful inline comments and documentations.
- Implement clear and transparent documentation for privileged roles and sensitive protocol operations.
- Regularly review and simulate contract behavior against newly emerging attack vectors.

REVIEW NOTES | PACNETWORK

Overview

The **PacNetwork** project involves the following smart contracts:

- **AddressFactory**: This contract facilitates deterministic deployments of proxy and implementation contracts using CREATE2. It allows the computation of contract addresses for specific implementations (PacUSD, Staking, and MMFVault) based on unique salts and bytecode hashes.
- **MMFVaultDeployFactory**: This contract facilitates the deterministic deployment of `MMFVault` contracts using the `CREATE2` opcode, enabling predictable deployment addresses. It deploys both the `MMFVault` implementation and its `ERC1967Proxy` proxies, ensuring they match expected addresses derived from an external `AddressFactory`.
- **PacUSDDeployFactory**: The contract facilitates deterministic deployment of PacUSD contracts using the CREATE2 opcode, ensuring predictable addresses. It deploys both the PacUSD implementation and its ERC1967 proxy, following the UUPS proxy upgrade pattern.
- **StakingDeployFactory**: This contract facilitates the deployment of `PacUSDStaking` smart contracts using the CREATE2 opcode and UUPS proxy pattern.
- **PacUSD**: A stablecoin contract implementing an ERC20 token with advanced features such as role-based access control, minting and burning with transaction state tracking, pausing, and account blocklisting. It utilizes OpenZeppelin upgradeable contracts for security and upgradability. The contract supports minting and burning operations via pre-approved transaction IDs, rewards distribution, and fee-specific minting.
- **PacUSDStaking**: This contract enables users to stake PacUSD tokens, earn rewards based on their staking duration and amount, and claim or restake those rewards. It supports the management of multiple reward schemes, allows for a configurable minimum staking period, and ensures that staked tokens remain locked for the defined period.
- **MMFVault**: This contract facilitates the swapping of MMF tokens for PacUSD and vice versa, while incorporating a reward distribution mechanism. It uses a UUPS upgradeable pattern, integrates with a pricing oracle for accurate conversions, and applies fee rates for minting and redemption processes.

External Dependencies

The following are the main external addresses used within the contracts:

MMFVault

- mmfToken, pricer, feeReceiver, admin, upgrader.

PacUSDStaking

- rewardSchemes, RESERVE, admin, upgrader.

We assume these addresses are valid and non-vulnerable actors and implement proper logic to collaborate with the current project.

The following library/contract are considered as the third-party dependencies:

- @openzeppelin/contracts
- @openzeppelin/contracts-upgradeable

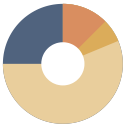
Privileged Functions

In the **PacNetwork** project, multiple roles are adopted to ensure the dynamic runtime updates of the project,which were specified in the **Centralization** findings.

The advantage of this privileged role in the codebase is that the client reserves the ability to adjust the protocol according to the runtime required to best serve the community. It is also worth of note the potential drawbacks of these functions, which should be clearly stated through the client's action/plan. Additionally, if the private key of the privileged account is compromised, it could lead to devastating consequences for the project.

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

FINDINGS | PACNETWORK



16	0	2	0	1	9	4
Total Findings	Critical	Centralization	Major	Medium	Minor	Informational

This report has been prepared for PacNetwork to identify potential vulnerabilities and security issues within the reviewed codebase. During the course of the audit, a total of 16 issues were identified. Leveraging a combination of Formal Verification, Manual Review & Static Analysis the following findings were uncovered:

ID	Title	Category	Severity	Status
THA-05	**Centralized Control Of Contract Upgrade**	**Centralization**	**Centralization**	**Acknowledged**
THA-06	**Centralization Related Risks**	**Centralization**	**Centralization**	**Acknowledged**
THA-09	Incompatibility With Deflationary Tokens	Logical Issue	Medium	Acknowledged
THA-07	No Upper Limit Of Rates	Logical Issue	Minor	Resolved
THA-08	Missing Slippage Protection	Logical Issue	Minor	Acknowledged
THA-10	Potential Out Of Gas Due To Missing Array Length Limits	Logical Issue	Minor	Resolved
THA-11	Lack Of Storage Gap Or NameSpaced Storage Layout In Upgradeable Contract	Design Issue	Minor	Resolved
THA-14	Potential Inconsistent Staking Timestamp Update	Inconsistency, Logical Issue	Minor	Acknowledged
THA-15	Incorrect Factory Contract Return Value	Logical Issue	Minor	Resolved
THA-18	Potential Precision Loss In Reward Rate Calculation Leading To Locked Rewards	Logical Issue	Minor	Acknowledged
THA-19	Incorrect Boundary Check In `addRewardScheme` Function	Logical Issue	Minor	Resolved

ID	Title	Category	Severity	Status
THA-20	Comment And Implementation Mismatch For Access Control Role In `mintReward` Function	Inconsistency	Minor	Resolved
THA-03	Staking Rewards Not Time-Based	Design Issue	Informational	Acknowledged
THA-04	Mint/Redeem Operations May Fail On Price Decrease	Design Issue	Informational	Acknowledged
THA-16	Inherited Contracts Not Initialized In Initializer	Logical Issue	Informational	Resolved
THA-17	Third-Party Dependencies	Volatile Code	Informational	Acknowledged

THA-05 | Centralized Control Of Contract Upgrade

Category	Severity	Location	Status
Centralization	Centralization	pacusd/PacUSD.sol (08/25-0dd9f8): 115; staking/PacUSDStaking.sol (08/25-0dd9f8): 44; vault/MMFVault.sol (08/25-0dd9f8): 111	Acknowledged

Description

In the following contracts, the role `owner` has the authority to update the implementation contract behind the contracts.

- PacUSD
- PacUSDStaking
- MMFVault

Any compromise to the `owner` account may allow a hacker to take advantage of this authority and change the implementation contract which is pointed by proxy and therefore execute potential malicious functionality in the implementation contract.

Recommendation

We recommend that the team make efforts to restrict access to the admin of the proxy contract. A strategy of combining a time-lock and a multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. In addition, the team should be transparent and notify the community in advance whenever they plan to migrate to a new implementation contract.

Here are some feasible short-term and long-term suggestions that would mitigate the potential risk to a different level and suggestions that would permanently fully resolve the risk.

Short Term:

A combination of a time-lock and a multi signature (⅔, ⅗) wallet mitigate the risk by delaying the sensitive operation and avoiding a single point of key management failure.

- A time-lock with reasonable latency, such as 48 hours, for awareness of privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to a private key compromised;
  AND
- A medium/blog link for sharing the time-lock contract and multi-signers addresses information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Long Term:

A combination of a time-lock on the contract upgrade operation and a DAO for controlling the upgrade operation mitigate the contract upgrade risk by applying transparency and decentralization.

- A time-lock with reasonable latency, such as 48 hours, for community awareness of privileged operations;
  AND
- Introduction of a DAO, governance, or voting module to increase decentralization, transparency, and user involvement;
  AND
- A medium/blog link for sharing the time-lock contract, multi-signers addresses, and DAO information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Permanent:

Renouncing ownership of the `admin` account or removing the upgrade functionality can *fully* resolve the risk.

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

Note: we recommend the project team consider the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

[PacNetwork, 09/08/2025]: In short term, we will use multi-sig wallet to control the roles defined in the contract. We are using Safe Wallet for multi-sig operations and will provide the addresses after the deployment if needed. In long term, once the code stablized, we will consider implementing the suggested measures.

[CertiK, 09/08/2025]: It is suggested to implement the aforementioned methods to avoid centralized failure. Also, CertiK strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

THA-06 | Centralization Related Risks

Category	Severity	Location	Status
Centralization	**Centralization**		**Acknowledged**

Description

AddressFactory

owner

In the contract `AddressFactory` the role `owner` has authority over the functions below.

- computeAddress()
- computeVaultAddress()

Any compromise to the `owner` account may allow the hacker to take advantage of this authority and compute the addresses.

MMFVaultDeployFactory

owner

In the contract `MMFVaultDeployFactory` the role `owner` has authority over the functions below.

- deployContracts()

Any compromise to the `owner` account may allow the hacker to take advantage of this authority and deploy contracts.

PacUSDDeployFactory

owner

In the contract `PacUSDDeployFactory` the role `owner` has authority over the functions below.

- deployContracts()

Any compromise to the `owner` account may allow the hacker to take advantage of this authority and deploy contracts.

StakingDeployFactory

owner

In the contract `StakingDeployFactory` the role `owner` has authority over the functions below.

- deployContracts()

Any compromise to the `owner` account may allow the hacker to take advantage of this authority and deploy contracts.

PacUSD

DEFAULT_ADMIN_ROLE

In the contract `PacUSD` the role `DEFAULT_ADMIN_ROLE` has authority over the functions below.

- grantRole()
- revokeRole()

Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and manage the configuration of the contract.

APPROVER_ROLE

In the contract `PacUSD` the role `APPROVER_ROLE` has authority over the functions below.

- setMintByTx()
- cancelMintByTx()
- setBurnByTx()
- cancelBurnByTx()

Any compromise to the `APPROVER_ROLE` account may allow the hacker to take advantage of this authority and manage the burn/mint operations.

BLOCKLISTER_ROLE

In the contract `PacUSD` the role `BLOCKLISTER_ROLE` has authority over the functions below.

- addToBlocklist()
- removeFromBlocklist()

Any compromise to the `BLOCKLISTER_ROLE` account may allow the hacker to take advantage of this authority and manage the blocklist.

PAUSER_ROLE

In the contract `PacUSD` the role `PAUSER_ROLE` has authority over the functions below.

- pause()
- unpause()

Any compromise to the `PAUSER_ROLE` account may allow the hacker to take advantage of this authority and pause/unpause the functionality of the contract.

RESCUER_ROLE

In the contract `PacUSD` the role `RESCUER_ROLE` has authority over the functions below.

- rescueTokens()

Any compromise to the `RESCUER_ROLE` account may allow the hacker to take advantage of this authority and rescue tokens stuck in the contract.

_minters

In the contract `PacUSD` the role `_minters` has authority over the functions below.

- mintByTx()
- burnByTx()
- mintReward()
- mintFee()

Any compromise to the `_minters` account may allow the hacker to take advantage of this authority and mint/burn tokens.

_owner

In the contract `PacUSD` the role `_owner` has authority over the functions below.

- _authorizeUpgrade()

Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and upgrade the contract.

BaseStaking

_owner

In the contract `BaseStaking` the role `_owner` has authority over the functions below.

- _authorizeUpgrade()

Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and upgrade the contract.

PacUSDStaking

DEFAULT_ADMIN_ROLE

In the contract `PacUSDStaking` the role `DEFAULT_ADMIN_ROLE` has authority over the functions below.

- setMinStakingPeriod()
- grantRole()
- revokeRole()

Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and manage the configuration of the contract.

PAUSER_ROLE

In the contract `PacUSDStaking` the role `PAUSER_ROLE` has authority over the functions below.

- pause()
- unpause()

Any compromise to the `PAUSER_ROLE` account may allow the hacker to take advantage of this authority and pause/unpause the functionality of the contract.

RESERVE_SET_ROLE

In the contract `PacUSDStaking` the role `RESERVE_SET_ROLE` has authority over the functions below.

- setReserve()

Any compromise to the `RESERVE_SET_ROLE` account may allow the hacker to take advantage of this authority and set the `RESERVE` account.

REWARD_SCHEME_ROLE

In the contract `PacUSDStaking` the role `REWARD_SCHEME_ROLE` has authority over the functions below.

- addRewardScheme()
- removeRewardScheme()

Any compromise to the `REWARD_SCHEME_ROLE` account may allow the hacker to take advantage of this authority and manage the reward schemes.

updater

In the contract `PacUSDStaking` the role `updater` has authority over the functions below.

- distributeReward()

Any compromise to the `_updater` account may allow the hacker to take advantage of this authority and distributing rewards.

MMFVault

DEFAULT_ADMIN_ROLE

In the contract `MMFVault` the role `DEFAULT_ADMIN_ROLE` has authority over the functions below.

- grantRole()
- revokeRole()

- updateFeeReceiver()
- updateMintFeeRate()
- updateRedeemFeeRate()

Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and manage the configuration of the contract.

PAUSER_ROLE

In the contract `MMFVault` the role `PAUSER_ROLE` has authority over the functions below.

- pause()
- unpause()

Any compromise to the `PAUSER_ROLE` account may allow the hacker to take advantage of this authority and pause/unpause the functionality of the contract.

MINT_REWARD_ROLE

In the contract `MMFVault` the role `MINT_REWARD_ROLE` has authority over the functions below.

- mintReward()

Any compromise to the `MINT_REWARD_ROLE` account may allow the hacker to take advantage of this authority and mint the rewards.

_owner

In the contract `MMFVault` the role `_owner` has authority over the functions below.

- _authorizeUpgrade()

Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and upgrade the contract.

Recommendation

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

Short Term:

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

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

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

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

Permanent:

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

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

Alleviation

[PacNetwork, 09/08/2025]: In the short term, we will use multi-sig wallet to control different roles defined in the smart contract. In the long term, after the code is stablized, we will consider implementing the suggested measures to mitigate risks.

[CertiK, 09/08/2025]: It is suggested to implement the aforementioned methods to avoid centralized failure. Also, CertiK strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

THA-09 | Incompatibility With Deflationary Tokens

Category	Severity	Location	Status
Logical Issue	Medium	vault/MMFVault.sol (08/25-0dd9f8): 200, 206~208, 214	Acknowledged

Description

The `mintPacUSD` function in the `MMFVault` contract does not account for transfer fees or rebase mechanics present in some ERC20 tokens. Currently, `_totalMMFToken` is incremented by the user-supplied `amount` before confirming how many tokens were actually received.

If the MMF token implements transfer fees or rebases, the contract's internal tracking will not match its real MMF balance.

Moreover, the function mints `PacUSD` based on the input `amount` rather than the actual MMF tokens received, which may allow users to mint more `PacUSD` than the contract's MMF balance supports. This mismatch introduces accounting errors that could be exploited or lead to inconsistencies between the vault and its token balances, especially when handling deflationary or rebase tokens.

Recommendation

It's recommended to update `_totalMMFToken` based on the actual received amount and add validation for tokens with transfer fees or rebase mechanisms.

Alleviation

[PacNetwork, 09/08/2025]: The fee will be charged in PacUSD and therefore, won't affect the amount of MMFToken held by MMFVault

THA-07 | No Upper Limit Of Rates

Category	Severity	Location	Status
Logical Issue	Minor	vault/MMFVault.sol (08/25-0dd9f8): 155, 169	Resolved

Description

The `updateMintFeeRate` and `updateRedeemFeeRate` functions of `MMFVault` contract do not enforce a proper upper limit on the rates that can be set for minting and redemption fees. As a result, an administrator could set these rates to excessively high values, up to and including 100%. This would allow fees that could discourage or prevent users from participating in minting or redeeming activities, impacting protocol usability.

Recommendation

We recommend adding reasonable boundaries for the rates.

Alleviation

[PacNetwork, 09/08/2025]: The team heeded the advice and resolved the issue in the updated version fab6783757a52c50b32d723e5aa4b8ae9366fb5a.

THA-08 | Missing Slippage Protection

Category	Severity	Location	Status
Logical Issue	Minor	vault/MMFVault.sol (08/25-0dd9f8): 179~184, 239~244	Acknowledged

Description

In the `MMFVault` contract, both `mintPacUSD` (line 183) and `redeemMMF` (line 243) functions lack slippage protection. Users have no way to specify minimum amounts they expect to receive, making them vulnerable to front-running attacks and price manipulation.

This vulnerability could lead to users receiving significantly less than expected due to price movements or rates changes.

Recommendation

It's recommended to add slippage protection parameters in both functions.

Alleviation

[PacNetwork, 09/08/2025]:
The frequency of price updating is relatively low (e.g., once or twice per day). Moreover, all the swaps, according to the current implementation, need to be authorized, and therefore, do not suffer from front-running attacks.

THA-10 | Potential Out Of Gas Due To Missing Array Length Limits

Category	Severity	Location	Status
Logical Issue	Minor	staking/PacUSDStaking.sol (08/25-0dd9f8): 196, 236, 283, 339	Resolved

Description

In the `PacUSDStaking` contract, multiple loops iterate over arrays without length limits (e.g., line 195 in `stake` function). If these arrays grow too large, transactions could fail due to gas limits, potentially preventing users from staking, unstaking and reward distribution.

This could lead to denial of service for users when the system scales or gas consumption escalates.

Recommendation

It's recommended to introduce maximum array length limits or implement alternative safeguards to reduce reliance on unbounded loops, in order to ensure that transaction gas costs remain manageable and to mitigate potential denial-of-service risks as the system scales.

Alleviation

[PacNetwork, 09/08/2025]: The team heeded the advice and resolved the issue in the updated version fab6783757a52c50b32d723e5aa4b8ae9366fb5a.

THA-11 Lack Of Storage Gap Or NameSpaced Storage Layout In Upgradeable Contract

Category	Severity	Location	Status
Design Issue	Minor	staking/BaseStaking.sol (08/25-0dd9f8): 16	Resolved

Description

When updating upgradeable smart contracts for new features or bug fixes, keeping the state variables' declaration order unchanged is essential to avoid storage layout issues.

A practical solution is to include unused state variables or explicitly named storage gaps (like `__gap`) in the base contracts. This foresight allows reserved slots for future use, ensuring that any additions to the contract's state won't disrupt the storage pattern of derived contracts or the compatibility with previously deployed versions. After `ERC-7201`, it is also possible to place all storage variables of a contract into one or more structs like `Namespaced Storage Layout`.

The problem of "Lack of Storage Gap Or NameSpaced Storage Layout in Upgradeable Contract" occurs when **these storage gaps are not incorporated into the base contract's logic nor the base contract defines the namespace storage layout**. As a result, if new state variables are added to the base contract, they might overwrite existing variables in the child contracts due to storage slot collisions.

In the current contract, the contract allows for future upgrades and is also inherited by other contracts. However, the storage gap is missing for the the current contract, nor is the `namespaced storage layout` used.

For detailed guidelines and best practices, refer to the following OpenZeppelin documentation:

- https://docs.openzeppelin.com/contracts/4.x/upgradeable#storage_gaps
- https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#storage-gaps

Recommendation

To mitigate this issue:

1. For enhanced flexibility in future upgrades of the logic contract, it is prudent to reserve a storage gap of an appropriate size in the base contract. This is achieved by declaring a fixed-size array, typically of `uint256` elements, each occupying a 32-byte slot, in the base contract. Label this array with the identifier `__gap` or any name prefixed with `__gap_` to indicate its purpose as a reserved space clearly.
2. it is also possible by placing all storage variables of a contract into one or more structs like `Namespaced Storage Layout`.

More detailed info :

- https://docs.openzeppelin.com/contracts/4.x/upgradeable#storage_gaps

Alleviation

[PacNetwork, 09/08/2025]: The team heeded the advice and resolved the issue in the updated version fab6783757a52c50b32d723e5aa4b8ae9366fb5a.

THA-14 | Potential Inconsistent Staking Timestamp Update

Category	Severity	Location	Status
Inconsistency, Logical Issue	Minor	staking/PacUSDStaking.sol (08/25-0dd9f8): 264	Acknowledged

Description

In the `PacUSDStaking` contract, there is an inconsistency in staking timestamp management. In the `stake` function (line 195), the staking timestamp is reset every time a user stakes, but in the `restake` function (line 290), the timestamp is not reset. This creates inconsistent behavior where users can potentially bypass the minimum staking period requirement by using the `restake` function.

This could lead to users circumventing the intended staking period restrictions.

Recommendation

It's recommended to reset `stakingTimestamps` in the `restake` function.

Alleviation

[PacNetwork, 09/08/2025]: The current implementation is consistent with our product design.

THA-15 | Incorrect Factory Contract Return Value

Category	Severity	Location	Status
Logical Issue	Minor	factory/MMFVaultDeployFactory.sol (08/25-0dd9f8): 44	Resolved

Description

In the `deployContracts` function of `MMFVaultDeployFactory` contract (line 43), the function returns `mmfVaultProxy` which contains the address of the last deployed vault proxy, not all deployed addresses. This could lead to confusion for callers who expect to receive information about all deployed contracts.

Recommendation

It's recommended to return an array of all deployed vault addresses.

Alleviation

[PacNetwork, 09/08/2025]: The team heeded the advice and resolved the issue in the updated version fab6783757a52c50b32d723e5aa4b8ae9366fb5a.

THA-18 Potential Precision Loss In Reward Rate Calculation Leading To Locked Rewards

Category	Severity	Location	Status
Logical Issue	Minor	staking/PacUSDStaking.sol (08/25-0dd9f8): 330, 399~400, 410	Acknowledged

Description

In the `_calculateRateInc` function of `PacUSDStaking` contract (line 410), there is a precision loss vulnerability that can result in rewards being permanently locked in the contract. The function calculates the reward rate increment as:

```
405     function _calculateRateInc(
406         uint256 newReward,
407         uint256 totalSupply
408     ) internal view virtual returns (uint256) {
409         return
410     @>      (newReward * RATE_PRECISION) / totalSupply;
411     }
```

When `newReward * RATE_PRECISION < totalSupply`, the division operation results in `rateInc = 0`. Although the `RATE_PRECISION` is set to `10^(token_decimals * 2)` (providing high precision), this can still occur when:

1. The total supply is extremely large
2. The reward amount is relatively small
3. The ratio between reward and total supply is below the precision threshold

When `rateInc = 0`, the `accumulatedRewardRate` remains unchanged in the `distributeReward` function, even though the reward tokens would be minted to the contract. This creates a situation where:

- Users cannot claim these rewards through the normal reward calculation mechanism
- Rewards are locked in the contract

Additionally, in the `_calculateRewardIncrement` function (lines 399-400), user reward calculations can also result in zero:

```
    uint256 reward = (bal * (accumulatedRewardRate - entryRewardRate)) /
RATE_PRECISION;
```

When `bal * (accumulatedRewardRate - entryRewardRate) < RATE_PRECISION`, the user's calculated reward becomes 0, even though they should be entitled to some reward based on their staking amount and the reward rate difference. This affects:

- Small stakers who may receive no rewards despite being eligible
- Users with small reward rate differences who lose their entitled rewards

Recommendation

It's recommended to refactor the code to avoid truncating small values that should rightfully accrue to users.

Alleviation

[PacNetwork, 09/08/2025]: We've noticed the issue raised here. However, considering what happens when running the project in practice as well as the rate precision set, we believe the risk has been well taken care of. Therefore, we won't make any changes for the current version.

THA-19 | Incorrect Boundary Check In `addRewardScheme` Function

Category	Severity	Location	Status
Logical Issue	Minor	staking/PacUSDStaking.sol (09/08-84090e): 118	Resolved

Description

The `addRewardScheme` function contains a flawed boundary check that fails to properly enforce the `MAX_REWARD_SCHEMES` limit. The current implementation checks if the array length exceeds the maximum **before** adding the new scheme, rather than checking if adding the new scheme would exceed the limit.

```
109     function addRewardScheme(
110         address scheme
111     ) external onlyRole(REWARD_SCHEME_ROLE) {
112         if (scheme == address(0)) revert ZeroAddress();
113
114         // check if the scheme is already added
115         if (schemeIndexMap[scheme] != 0)
116             revert RewardSchemeAlreadyAdded(scheme);
117
118     @>  if (rewardSchemes.length > MAX_REWARD_SCHEMES) {
119             revert RewardSchemeArrayTooLong();
120         }
121         // add the scheme
122         rewardSchemes.push(scheme);
123         uint256 index = rewardSchemes.length;
124         schemeIndexMap[scheme] = index;
125
126         emit RewardSchemeAdded(scheme);
127     }
```

The check occurs before `rewardSchemes.push(scheme)` is executed. When `rewardSchemes.length == 5` (equal to `MAX_REWARD_SCHEMES`), the check passes and allows adding a 6th scheme, which exceeds its intended `MAX_REWARD_SCHEMES = 5` limit.

Recommendation

It's recommended to `>=` instead of `>` in the boundary check.

Alleviation

[PacNetwork, 09/18/2025]: Issue acknowledged. Changes have been reflected in the commit hash: fab6783757a52c50b32d723e5aa4b8ae9366fb5a.

THA-20 Comment And Implementation Mismatch For Access Control Role In `mintReward` Function

Category	Severity	Location	Status
Inconsistency	Minor	vault/MMFVault.sol (09/08-84090e): 309, 315~317	Resolved

Description

In the `mintReward` function of `MMFVault` contract, there is a discrepancy between the function comment and the actual implementation regarding the required access control role. The comment states that the function requires `DEFAULT_ADMIN_ROLE`, while the implementation actually uses `MINT_REWARD_ROLE`, which causes inconsistency.

```
    /**
     * @dev Mints rewards for stakers based on price increases
     *
     * This function is called by an administrator to calculate and distribute
rewards
     * to the staking contract when the latest price is higher than the last
recorded price.
     * The reward amount is calculated based on the price difference, total staked
MMF tokens,
     * and precision conversion factors.
     *
     * Restrictions:
     * - Contract must not be paused (whenNotPaused)
     * - Protected against reentrancy attacks (nonReentrant)
     * - Only callable by addresses with DEFAULT_ADMIN_ROLE (onlyRole)
     *
     * @param rewardPrice The price provided by the caller, which must match the
latest price
     *                    to ensure accuracy
     *
     */
    function mintReward(
        uint256 rewardPrice
    ) public whenNotPaused nonReentrant onlyRole(MINT_REWARD_ROLE) {
        uint256 currentPrice = pricer.getLatestPrice();
        if (rewardPrice != currentPrice) revert MismatchPrice();
        if (currentPrice < lastPrice) revert InvalidPrice();
        if (currentPrice == lastPrice) return; // No price change, no reward
        if (currentPrice > lastPrice) {
            if (_totalMMFToken > uint256(0)) {
                uint256 priceDifference = currentPrice - lastPrice;
                uint256 rewardAmount = (priceDifference *
                    _totalMMFToken *
                    pacUSDPrecision) /
                    mmfTokenPrecision /
                    PRICER_PRECISION;
                uint256 tempLastPrice = lastPrice;
                lastPrice = currentPrice;
                staking.distributeReward(rewardAmount);
                pacUSD.mintReward(rewardAmount, address(staking));
                emit RewardMinted(
                    address(staking),
                    rewardAmount,
                    tempLastPrice,
                    currentPrice,
                    _totalMMFToken
                );
            } else {
                lastPrice = currentPrice; //need update price
```

```
            }
        }
    }
```

Recommendation

It's recommended to refactor the code to ensure the correct role is used.

Alleviation

[PacNetwork, 09/18/2025]: Issue acknowledged. Changes have been reflected in the commit hash: fab6783757a52c50b32d723e5aa4b8ae9366fb5a.

THA-03 | Staking Rewards Not Time-Based

Category	Severity	Location	Status
Design Issue	Informational	staking/PacUSDStaking.sol (08/25-0dd9f8): 11, 39	Acknowledged

Description

In the `PacUSDStaking` contract, the `stakingTimestamps` mapping is only used to restrict users from performing consecutive unstake operations within a short timeframe. However, the reward distribution mechanism does not consider the duration of the stake.

The reward amount is solely dependent on the `accumulatedRewardRate`, which is typically updated by the vault contract, rather than on how long a user has staked. As a result, two users who stake the same amount of tokens, but for vastly different durations (e.g., 1 day vs. 1 week), may receive the same reward amount as long as they have staked before the reward distribution occurs.

Recommendation

This design may not align with common staking models, where rewards are typically proportional to both the staked amount and the staking duration. It raises the question of whether this behavior is intentional or a potential oversight in the reward logic.

Alleviation

[PacNetwork, 09/05/2025]: Issue acknowledged. I won't make any changes for the current version. This behavior is intentional according to our design. The intention is just to enforce the minimal staking period for claiming reward.

THA-04 | Mint/Redeem Operations May Fail On Price Decrease

Category	Severity	Location	Status
Design Issue	Informational	vault/MMFVault.sol (08/25-0dd9f8): 202~203, 262~263, 301	Acknowledged

Description

In the `MMFVault` contract, the reward distribution logic depends on price changes. When distributing rewards, if the current price is lower than the `lastPrice`, the transaction reverts and `lastPrice` is not updated.

However, the `mintPacUSD` and `redeemMMF` functions enforce a requirement that the `currentPrice` must equal the stored `lastPrice`. As a result, if the price decreases, `lastPrice` remains outdated, and subsequent calls to `mintPacUSD` or `redeemMMF` will revert. This behavior effectively prevents users from minting or redeeming when the price moves downward.

Recommendation

This implementation may unintentionally lock core vault operations during price declines. It is unclear whether this behavior is intentional or an oversight in the reward distribution and state update logic.

Alleviation

[PacNetwork, 09/08/2025]: According to our design, the price should unilaterally increase. It is therefore, no need to consider the locking situation. We won't make any changes for the current version.

THA-16 | Inherited Contracts Not Initialized In Initializer

Category	Severity	Location	Status
Logical Issue	Informational	pacusd/PacUSD.sol (08/25-0dd9f8): 26; vault/MMFVault.sol (08/25-0dd9f8): 26	Resolved

Description

Contracts `PacUSD` and `MMFVault` extend `Ownable2StepUpgradeable`, but the extended contract is not initialized by the current contract. Generally, the initializer function of a contract should always call all the initializer functions of the contracts that it extends.

Recommendation

We recommend initializing `Ownable2StepUpgradeable`.

Alleviation

[PacNetwork, 09/08/2025]: The team heeded the advice and resolved the issue in the updated version fab6783757a52c50b32d723e5aa4b8ae9366fb5a.

THA-17 | Third-Party Dependencies

Category	Severity	Location	Status
Volatile Code	Informational		Acknowledged

Description

The project is serving as the underlying entity to interact with third-party contracts, **pricer**, **rewardSchemes**, **mmfToken**. The scope of the audit treats third-party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets.

Recommendation

We recommend that the project team constantly monitor the functionality of the dependency to mitigate any side effects that may occur when unexpected changes are introduced.

Alleviation

[PacNetwork, 09/08/2025]: Issue acknowledged. The third-party dependencies will be constantly monitored.

OPTIMIZATIONS | PACNETWORK

ID	Title	Category	Severity	Status
THA-01	Variables That Could Be Declared As Immutable	Gas Optimization	Optimization	Resolved
THA-02	Duplicate Role Assignment In Initialization	Logical Issue	Optimization	Resolved

THA-01 | Variables That Could Be Declared As Immutable

Category	Severity	Location	Status
Gas Optimization	Optimization	factory/AddressFactory.sol (08/25-0dd9f8): 40; factory/MMFVaultDeployFactory.sol (08/25-0dd9f8): 17; factory/PacUSDDeployFactory.sol (08/25-0dd9f8): 17; factory/StakingDeployFactory.sol (08/25-0dd9f8): 12	Resolved

Description

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

Recommendation

We recommend declaring these variables as immutable. Please note that the `immutable` keyword only works in Solidity version `v0.6.5` and up.

Alleviation

[PacNetwork, 09/08/2025]: The team heeded the advice and resolved the issue in the updated version fab6783757a52c50b32d723e5aa4b8ae9366fb5a.

THA-02 | Duplicate Role Assignment In Initialization

Category	Severity	Location	Status
Logical Issue	Optimization	staking/PacUSDStaking.sol (08/25-0dd9f8): 88	Resolved

Description

In the `PacUSDStaking` contract's `initialize` function (line 87), there is a comment indicating duplicate code for role assignment. The `_grantRole(DEFAULT_ADMIN_ROLE, admin)` is called both in the parent `__BaseStaking_init` function and in the child contract's initialize function.

Recommendation

It's recommended to remove the duplicate role assignment from the child contract's initialize function since it's already handled in the parent contract.

Alleviation

[PacNetwork, 09/08/2025]: The team heeded the advice and resolved the issue in the updated version fab6783757a52c50b32d723e5aa4b8ae9366fb5a.

APPENDIX | PACNETWORK

Audit Scope

PacNetwork/evm-contracts
pacusd/PacUSD.sol
staking/PacUSDStaking.sol
vault/MMFVault.sol
factory/AddressFactory.sol
factory/MMFVaultDeployFactory.sol
factory/PacUSDDeployFactory.sol
factory/StakingDeployFactory.sol
staking/BaseStaking.sol
staking/PacUSDStaking.sol
vault/MMFVault.sol
factory/AddressFactory.sol
factory/MMFVaultDeployFactory.sol
factory/PacUSDDeployFactory.sol
factory/StakingDeployFactory.sol
pacusd/PacUSD.sol
staking/BaseStaking.sol
staking/PacUSDStaking.sol
vault/MMFVault.sol
factory/AddressFactory.sol

PacNetwork/evm-contracts
factory/MMFVaultDeployFactory.sol
factory/PacUSDDeployFactory.sol
factory/StakingDeployFactory.sol
pacusd/PacUSD.sol
staking/BaseStaking.sol

Finding Categories

Categories	Description
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

DISCLAIMER | CERTIK

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

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

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

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

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

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

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

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

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

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

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

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

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

Elevating Your Web3 Journey

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