Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Navi Protocol

Veridise Inc.

January 19, 2026

► **Prepared For:**

Navi Protocol
www.naviprotocol.io

► **Prepared By:**

Aayushman Thapa Magar
Evgeniy Shishkin

► **Contact Us:**

contact@veridise.com

► **Version History:**

Feb. 10, 2026	V1
Feb. 5, 2026	Initial Draft

Contents

1 Executive Summary

From Jan. 19, 2026 to Jan. 30, 2026, Navi Protocol engaged Veridise to conduct a security assessment covering several extensions to the existing Navi Protocol.

The project was previously audited by Veridise at commit `945878c`. This engagement reviews only the changes introduced since commit `2954d3b`, focusing on the security impact of the new additions. These additions introduce several enhancements, including EMode functionality, multi-market support, asset-specific borrowing weights, and integration of the Switchboard oracle.

Veridise conducted the assessment over 4 person-weeks, with 2 security analysts reviewing the project over 2 weeks on commit `bc86742`. The review strategy involved a tool-assisted analysis of the program source code performed by Veridise security analysts as well as thorough code review.

Project Summary. Navi Protocol is a lending protocol built on the Sui blockchain. Liquidity providers deposit funds into the protocol's pools, while borrowers draw liquidity by posting collateral and paying interest on their loans. Liquidity providers earn yield from the interest paid by borrowers.

If a borrower's collateral value falls below the required liquidation threshold, the position becomes eligible for liquidation. In such cases, liquidators may repay the outstanding debt and acquire the collateral at a discounted price.

Navi Protocol also supports flash loans, enabling users to borrow assets without collateral provided that the funds are repaid within the same transaction. A fee is charged for this service and distributed between liquidity providers and the protocol.

To incentivize participation, the protocol distributes rewards to users based on their activity.

Asset pricing is sourced from multiple on-chain oracle providers. Currently supported oracles include Supra, Pyth, and Switchboard.

The recent protocol extensions under review include the following:

- **Efficiency Mode (EMode).** Provides preferential risk parameters and more attractive rates for selected collateral–debt pairs with lower perceived price volatility.
- **Multi-market architecture.** Introduces multiple coexisting markets, each with independent interest rate models and reward configurations.
- **Borrowing weights.** Enables asset-specific borrowing parameters to better reflect individual risk profiles.
- **Switchboard oracle integration.** Adds an additional decentralized price feed source.

Code Assessment. The Navi Protocol developers provided the source code of the Navi Protocol contracts for review. To support the Veridise security analysts' understanding of the changes, the Navi Protocol developers provided documentation describing the design and implementation of the new features. The Veridise analysts also referenced prior audit reports and existing project documentation to further contextualize the review.

The source code included a comprehensive test suite that covered many expected user flows and a substantial portion of the protocol's functionality, including the newly introduced features.

Summary of Issues Detected. The security assessment uncovered 8 issues. Among these, 1 was a medium-severity issue in which several functions do not enforce consistency of `market_id` between `Storage` and `Pool`. This may result in cross-market state being mixed and can impact calculations or enable unintended treasury withdrawals across markets that share the same coin type (V-NAVP-VUL-001).

The Veridise analysts also identified 4 low-severity issues, including flash loan entry points that are not gated by the protocol pause mechanism and remain callable while the protocol is paused (V-NAVP-VUL-002); missing sanity checks when updating EMode parameters (V-NAVP-VUL-003); an incorrect market identifier emitted in the new market creation event (V-NAVP-VUL-004); and missing validation that oracle price-feed bounds are well-formed (i.e., max ≥ min) at creation time (V-NAVP-VUL-005).

3 warnings were also identified, such as issued `OwnerCap` objects that cannot be revoked, increasing the impact of key loss or compromise by granting persistent administrative privileges (V-NAVP-VUL-007), as well as maintainability and defense in depth concerns (V-NAVP-VUL-006) and the continued accessibility of deprecated or legacy incentive logic after upgrade (V-NAVP-VUL-008).

The Navi Protocol developers provided fixes for the identified issues in a timely manner and left helpful comments for some issues regarding their reasoning.

Recommendations. After conducting the assessment of the protocol, the security analysts had a few suggestions to improve the Navi Protocol.

Reconsider the access control design. Currently, Navi Protocol relies on at least seven distinct privileged user groups, potentially more depending on how protocol modules are categorized. Each group is represented by a dedicated Capability object governing specific administrative functions.

The separation of administrative responsibilities across multiple roles can be beneficial in limiting the operational scope of individual privileges. However, in the absence of any mechanism to revoke or rotate issued Capability objects, the proliferation of such roles materially increases systemic risk. Because each Capability grants high-impact control over critical protocol functions, the compromise of even a single administrative key may lead to severe protocol disruption.

As the number of independently privileged roles grows, so does the likelihood of credential leakage and the overall attack surface. It is therefore recommended to reassess the access control model to incorporate revocable permissions, clearer privilege boundaries, and stronger lifecycle management for administrative authorities.

Introduce timelock or governance-style controls for critical parameter changes. Several sensitive protocol parameters — including asset borrowing rates and EMode liquidation thresholds —

can currently be modified by an administrator in a single transaction. Such immediate changes may expose user positions to unexpected liquidation risk.

Across the DeFi ecosystem, it is common practice to apply a timelock or cooldown period to high-impact configuration updates. This allows users sufficient time to assess the changes and take appropriate action, such as adjusting or closing their positions.

It is recommended to consider implementing similar protective mechanisms within Navi Protocol to improve transparency and user safety.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
Navi Protocol	`bc86742`	Move	SUI

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Jan. 19–Jan. 30, 2026	Manual & Tools	2	4 person-weeks

Table 2.3: Vulnerability Summary.

Name	Number	Acknowledged	Fixed
Critical-Severity Issues	0	0	0
High-Severity Issues	0	0	0
Medium-Severity Issues	1	1	1
Low-Severity Issues	4	4	4
Warning-Severity Issues	3	3	1
Informational-Severity Issues	0	0	0
TOTAL	8	8	6

Table 2.4: Category Breakdown.

Name	Number
Data Validation	3
Access Control	3
Usability Issue	1
Maintainability	1

3 Security Assessment Goals and Scope

3.1 Security Assessment Methodology

The security assessment process consists of the following steps:

1. Gather an initial understanding of the protocol's business logic, users, and workflows by reviewing the provided documentation and consulting with the developers.
2. Identify all valuable assets in the protocol.
3. Identify the main workflows for managing these assets.
4. Identify the most significant security risks associated with these assets.
5. Systematically review the codebase for execution paths that could trigger the identified security risks, considering different assumptions.
6. Prioritize one finding over another by assigning a severity level to each.

3.2 Identified Security Risks

After the initial phase of the security assessment was completed, a list of potential security risks was generated. Security analysts used this list during the code review as a starting point to identify potential attack vectors. A few of these risks, when expressed as questions, include the following:

- Can an eMode be created with more or fewer than two assets?
- Can assets be swapped or mutated after eMode creation?
- Can an eMode reference the same asset twice?
- Can an asset marked as non-borrowable still be borrowed within eMode?
- Can an asset marked as non-collateralizable still be used as collateral?
- Can both flags be bypassed through edge-case calls or upgrades?
- Are default LTV, liquidation threshold, or liquidation bonus still applied instead of eMode values?
- Can partial overrides occur (e.g., LTV updated but LT not)?
- Can a user enter multiple eModes simultaneously?
- Can a market be initialized using a Storage, Pool, Incentive, or FlashLoanConfig object from another market?
- Can a mix of old and new market objects be combined during market creation?
- Can an attacker intentionally reuse a privileged shared object across markets?
- Can a new market be created with a manually specified or reused market_id?
- Can market_id be skipped, overwritten, or reset through upgrades or admin calls?
- Can race conditions lead to two markets sharing the same market_id?
- Can functions be called using objects with mismatched market_id values?
- Are there any execution paths where the market_id check is missing or bypassed?
- Can multiple Storage objects with identical parameters and market_id be created?
- Can multiple Pools exist for the same asset and market_id?
- Can duplicate objects cause inconsistent accounting or liquidity fragmentation?
- Can a borrow weight be set for a non-existent or improperly initialized asset?
- Can unauthorized users invoke borrow weight updates?

- Can borrow weight be set below 1 or above 10?
- Can overflow, underflow, or type casting bypass the intended limits?
- Can zero or extreme values be introduced through upgrades or migrations?
- Is borrow weight consistently applied across all calculations (health factor, max borrow, liquidation checks)?
- Can partial application occur (e.g., applied to borrowing power but not liquidation logic)?
- Is Switchboard queried through the same validation and normalization pipeline as existing oracles?
- Can Switchboard bypass safety checks applied to other oracle providers?
- Can stale Switchboard prices be used without freshness checks?
- Is the timestamp verified against a maximum allowed delay?
- Can attackers exploit low update frequency during high volatility periods?
- Are Switchboard prices normalized to protocol precision correctly?

3.3 Scope

The scope of this security assessment is limited to a specific set of source files from the repository, as agreed upon with the Navi Protocol developers:

- `lending_core/sources/account.move`
- `lending_core/sources/dynamic_calculator.move`
- `lending_core/sources/error.move`
- `lending_core/sources/event.move`
- `lending_core/sources/flash_loan.move`
- `lending_core/sources/incentive_v2.move`
- `lending_core/sources/incentive_v3.move`
- `lending_core/sources/lending.move`
- `lending_core/sources/logic.move`
- `lending_core/sources/manage.move`
- `lending_core/sources/pool.move`
- `lending_core/sources/pool_manager.move`
- `lending_core/sources/storage.move`
- `oracle/sources/adaptor_swtichboard.move`
- `oracle/sources/oracle_manage.move`
- `oracle/sources/oracle_pro.move`
- `oracle/sources/oracle_provider.move`

3.4 Classification of Vulnerabilities

When Veridise security analysts discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise.

The severity of a vulnerability is evaluated according to the Table 3.1.

The likelihood of a vulnerability is evaluated according to the Table 3.2.

The impact of a vulnerability is evaluated according to the Table 3.3:

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Security Review Assumptions

4.1 Operational Assumptions

The following assumptions were made for the purposes of this security review:

- Users retain exclusive control over their Capability objects and do not share them with third parties.
- All functionality present in the base commit (prior to the introduction of the reviewed changes) was assumed to be correct and secure with respect to core protocol workflows, including deposits, withdrawals, borrowing, liquidations, flash loans, reward distribution, and related accounting logic.
- The Staking Pool and Volo Vault components of the Navi Protocol are assumed to operate in alignment with the protocol's intended design and best interests.
- Oracle configurations, including provider selection, freshness thresholds, and risk parameters, are assumed to be properly designed and conservatively calibrated, as the protocol's security model relies heavily on accurate and timely pricing data.

4.2 Privileged Roles

During the review, Veridise analysts assumed that the privileged users perform their responsibilities as intended. Protocol exploits relying on the below users acting outside of their privileged scope or intentionally abstaining from doing their duties are considered to be outside of scope of this security review.

- **Storage Admin**. Authorized to configure flash-loan parameters, withdraw funds from the protocol treasury, set reward configurations, and manage EMode parameters. This role can also issue Owner capability objects.
- **Owner**. Authorized to pause the protocol and modify critical market parameters, including loan-to-value ratios (LTV), liquidation thresholds, base interest rates, and optimal utilization settings. This role can also designate liquidators and protected liquidation accounts.
- **Borrow Fee Cap**. Authorized to set and remove borrow fee rates for specific assets and users.
- **Pool Admin**. Authorized to create new liquidity pools, withdraw protocol fees from individual pools, and configure key pool-level parameters.
- **Incentive Owner**. Authorized to create reward pools and configure reward distribution parameters.
- **Pool Manager Operator**. Authorized to set validator weights used to distribute funds allocated for liquid staking.
- **Oracle Admin.** Administers all oracle-related functionality, including enabling and configuring price feeds, defining maximum price freshness intervals, and managing provider-specific settings.

Operational Recommendations. Highly-privileged operations should be operated by a multi-sig contract or decentralized governance system. These operations should be guarded by a timelock to ensure there is enough time for incident response. All such operations should be tested in example scenarios to ensure the role operators are available and ready to respond when necessary.

Full validation of operational security practices is beyond the scope of this review. Users of the protocol should ensure they are confident that the operators of privileged keys are following best practices such as:

- Never storing a protocol key in plaintext, on a regularly used phone, laptop, or device, or relying on a custom solution for key management.
- Using separate keys for each separate function.
- Storing multi-sig keys in a diverse set of key management software/hardware services and geographic locations.
- Enabling 2FA for key management accounts. SMS should *not* be used for 2FA, nor should any account which uses SMS for 2FA. Authentication apps or hardware are preferred.
- Validating that no party has control over multiple multi-sig keys.
- Performing regularly scheduled key rotations for high-frequency operations.
- Securely storing physical, non-digital backups for critical keys.
- Actively monitoring for unexpected invocation of critical operations and/or deployed attack contracts.
- Regularly drilling responses to situations requiring emergency response such as pausing/unpausing.

5 Vulnerability Report

This section presents the vulnerabilities found during the security assessment. For each issue found, the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.) is specified. Table 5.1 summarizes the issues discovered:

Table 5.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-NAVP-VUL-001	Potential mismatch of Storage and Pool . . .	Medium	Fixed
V-NAVP-VUL-002	Flashloans do not respect protocol pausing	Low	Fixed
V-NAVP-VUL-003	Emode parameters change lack sanity . . .	Low	Fixed
V-NAVP-VUL-004	New market creation event parameter is . . .	Low	Fixed
V-NAVP-VUL-005	Maximum effective price of Oracle . . .	Low	Fixed
V-NAVP-VUL-006	Maintainability issues	Warning	Fixed
V-NAVP-VUL-007	Uncontrolled management of OwnerCaps	Warning	Acknowledged
V-NAVP-VUL-008	Irrelevant functions still accessible after . . .	Warning	Acknowledged

5.1 Detailed Description of Issues

5.1.1 V-NAVP-VUL-001: Potential mismatch of Storage and Pool markets in functions

Severity	Medium	**Commit**	bc86742
Type	Data Validation	**Status**	Fixed
Location(s)	▶ `lending_core/sources/storage.move` ▶ `oracle/sources/oracle_dynamic_getter.move`		
Confirmed Fix At	e3b318d		

Description The `market_id` field identifies a non-main market instance. Because multiple markets can be created concurrently, `market_id` must be consistently tracked and enforced across related objects, such as `Config`, `RewardFund`, `Storage`, `Pool`, and `Incentive`, to ensure they remain aligned.

It was identified that `dynamic_calculator::dynamic_health_factor()`, `dynamic_calculator::dynamic_calculate_apy()`, and `storage::withdraw_treasury_v2()` do not verify that the `market_id` of the `Storage` and `Pool` objects match.

Impact This can result in cross-market state being mixed, leading to incorrect calculations and potentially unauthorized treasury withdrawals across markets that share the same `CoinType`.

Recommendation It is recommended to add a check that enforces consistency of the `market_id` across the relevant objects.

Developers Response The developers implemented checks to ensure that `market_id` remains consistent between `Storage` and `Pool` across identified functions.

5.1.2 V-NAVP-VUL-002: Flashloans do not respect protocol pausing

Severity	Low	**Commit**	bc86742
Type	Access Control	**Status**	Fixed
Location(s)	`lending_core/sources/flash_loan.move`		
Confirmed Fix At	f9f4d63		

Description All user-facing protocol entry points are correctly guarded by a pause check, with the exception of the flash loan family of functions: `flash_loan_with_ctx()`, `flash_loan_with_ctx_v2()`, `flash_loan_with_account_cap()`, `flash_loan_with_account_cap_v2()` and corresponding repaying functions.

These functions remain callable even when the protocol is in a paused state, allowing users to execute flash loans while other functionality is intentionally disabled. This behavior is inconsistent with the protocol's pause mechanism and breaks the expectation that pausing the protocol fully suspends all user-initiated actions.

Impact Allowing flash loans while the protocol is paused may lead to unexpected or unsafe behavior, particularly if the pause is triggered in response to an incident. In addition, the protocol may continue to accrue or distribute fees during a paused state, potentially resulting in unintended economic outcomes or missed profit opportunities once normal operation resumes.

Recommendation Apply the same pause guard to the flash loan function as is used for all other protocol entry points, ensuring that no flash loans can be executed while the protocol is paused.

Developers Response The developers acknowledged the issue and agreed to provide a fix.

5.1.3 V-NAVP-VUL-003: Emode parameters change lack sanity checking

Severity	Low	**Commit**	bc86742
Type	Data Validation	**Status**	Fixed
Location(s)	`lending_core/sources/manage.move:210`		
Confirmed Fix At	adaa0e3		

Description The protocol allows an admin to modify EMode parameters. However, the new values are applied without any on-chain sanity checks or validation. As a result, invalid, inconsistent, or economically unsafe parameter values can be set and immediately take effect.

Impact Incorrect EMode configuration may break core risk assumptions of the protocol. Depending on the parameter set, this could result in mispriced risk, incorrect borrowing or liquidation behavior, or temporary disruption of normal protocol operation. Even accidental misconfiguration by a trusted admin could have protocol-wide consequences.

Recommendation Introduce explicit validation for all EMode parameters before applying updates. At a minimum, enforce bounds checks, non-zero and monotonicity constraints (where applicable), and invariants that must hold between related parameters. This reduces the risk of accidental misconfiguration and provides defense in depth against administrative error.

Developers Response The developers implemented checks to ensure that the provided parameter is less or equal to 1e27.

5.1.4 V-NAVP-VUL-004: New market creation event parameter is incorrect

Severity	Low	**Commit**	bc86742
Type	Usability Issue	**Status**	Fixed
Location(s)	`lending_core/sources/storage.move:1089`		
Confirmed Fix At	61d4292		

Description The `storage::create_new_market` function, which is responsible for creating a market-specific `Storage` object, emits an event containing metadata about the newly created market. However, the event reports an incorrect market identifier, which does not correspond to the actual market created by the function.

Impact Consumers of this event (e.g., off-chain indexers, back-end services, or monitoring infrastructure) may associate subsequent actions with an incorrect market. This can lead to data inconsistencies, operational confusion, and potential downstream errors in systems that rely on event data as a source of truth.

Recommendation The fix is straightforward: use the `market_info.market_id` instead of the current one.

Developers Response The event now emits `market_info.market_id` and increments the value afterwards.

5.1.5 V-NAVP-VUL-005: Maximum effective price of Oracle PriceFeed is not enforced to be greater than minimum

Severity	Low	**Commit**	bc86742
Type	Data Validation	**Status**	Fixed
Location(s)	`oracle/sources/oracle_manage.move:24`		
Confirmed Fix At	86996ef		

Description For each supported coin type, the protocol requires a corresponding oracle price feed to be created via the `oracle_manage::create_price_feed()` function. This function accepts parameters defining the minimum and maximum effective price for the asset.

However, there is no validation enforcing a logical relationship between these parameters. In particular, the protocol does not ensure that the maximum effective price is greater than (or at least equal to) the minimum effective price at creation time.

Impact Misconfigured price bounds may break core pricing invariants relied upon by the protocol. In the worst case, inverted or inconsistent bounds could lead to incorrect price calculations, inconsistent oracle behavior, or unexpected failures in components that assume a valid price range, potentially affecting risk management and user-facing operations.

Recommendation Enforce validation checks in `oracle_manage::create_price_feed()` to ensure that the maximum effective price is strictly greater than the minimum effective price. Additional sanity checks (e.g., non-zero values and reasonable upper bounds) are recommended to prevent accidental misconfiguration and to preserve oracle consistency guarantees.

Developers Response The developers added an assertion to check if minimum value is less than or equal to the maximum value for effective price.

5.1.6 V-NAVP-VUL-006: Maintainability issues

Severity	Warning	**Commit**	bc86742
Type	Maintainability	**Status**	Fixed
Location(s)	▶ `lending_core/sources/` • `account.move` • `incentive_v3.move` • `manage.move` • `pool.move` • `storage.move` ▶ `oracle/sources/config.move`		
Confirmed Fix At	4901eda		

Description The following maintainability issues were identified throughout the project:

Misleading comments

- `storage.move@L842` : The code contradicts the comment. If the given liquidator isn't considered designated, then the user is not liquidatable, but the comment says the opposite.
- `storage.move@L110` : The comment for `current_emode_id` field suggests it represents the "current" emode, however, it actually represents the "next" emode, since this value will be used as an identifier for the next emode created.
- `storage.move@L427` : The comment is incorrect and does not reflect the behavior of the function.
- `incentive_v3.move@L559` : The comment is incorrect, as the function exits instead of continuing.

Missing checks and validations

- `pool.move@L372` :
  The enabled flag isn't checked, even though the operation is going to mutate the stake.
- `storage.move@L800` : The `init_protected_liquidation_fields()` function does not validate the version of the provided `Storage` object.
- `storage.move@L724` :
  No version check for `Storage` in `withdraw_treasury_v2` function.
- `storage.move@L888-941` : No version checks for `Storage` in `create_emode_pair` , `set_emode_config_active` , `set_emode_asset_lt` , `set_emode_asset_ltv` , `set_emode_config_active` , `set_emode_asset_liquidation_bonus` functions.
- `config.move@L215-416` :
  The following functions do not check if `OracleConfig` is paused before updating its state:
  - `config::new_price_feed()`
  - `config::set_enable_to_price_feed()`
  - `config::set_max_timestamp_diff_to_price_feed()`
  - `config::set_price_diff_threshold1_to_price_feed()`

- config::set_price_diff_threshold2_to_price_feed()
- config::set_max_duration_within_thresholds_to_price_feed()
- config::set_maximum_allowed_span_percentage_to_price_feed()
- config::set_maximum_effective_price_to_price_feed()
- config::set_minimum_effective_price_to_price_feed()
- config::set_oracle_id_to_price_feed()
- config::set_primary_oracle_provider()
- config::set_secondary_oracle_provider()
- config::new_oracle_provider_config()
- config::set_oracle_provider_config_enable()
- config::new_oracle_provider_config()
- config::set_oracle_provider_config_pair_id()
- config::set_oracle_provider_config_enable()
- config::set_primary_oracle_provider()
- config::set_secondary_oracle_provider()

- storage.move@L273 :
  The existence of oracle_id isn't checked by config.is_price_feed_exists()

Miscellaneous

- storage.move@L922 :
  The 4th parameter is called param_type and denotes which parameter has been changed. It is recommended to introduce named constants for this purpose instead of relying on numeric literals.
- storage.move@L973 : The function is called is_passed_emode_check() but returns true when the user is not in emode. It is suggested to change the name of the function to something that better reflects its intention.
- storage.move@L1371 : The
  get_reserve_for_testing()
  test function is missing the #[test_only] attribute.
- account.move@L74 : The
  get_or_create_account_field() function is currently public, even when there is no need for this. It is recommended to restrict its visibility to shrink the potential attack surface even though we don't currently see any threats.

Developers Response The developers have made all the suggested changes except the following points:

- storage.move@L273 : The existence of oracle_id isn't checked by
  config.is_price_feed_exists()
- pool.move@L372 : The enabled flag isn't checked, even though the operation is going to mutate the stake.
- Changes in config.move regarding OracleConfig

5.1.7 V-NAVP-VUL-007: Uncontrolled management of OwnerCaps

Severity	Warning	**Commit**	bc86742
Type	Access Control	**Status**	Acknowledged
Location(s)	`lending_core/sources/storage.move:796`		
Confirmed Fix At	N/A		

Description The protocol allows privileged *Owner Cap* objects to be created by the admin. However, once an Owner Cap is created, there is no on-chain mechanism to revoke, invalidate, or otherwise disable it. As a result, any issued Owner Cap remains permanently valid for the lifetime of the protocol. If an Owner Cap is lost, leaked, or transferred to an unintended party, the protocol provides no way to recover from this state.

Impact Holders of an Owner Cap are authorized to modify protocol-critical parameters. Therefore, compromise of even a single Owner Cap would grant an attacker persistent administrative privileges, potentially allowing them to alter core protocol behavior, compromise economic assumptions, or disrupt protocol operations. The lack of a revocation mechanism significantly increases the blast radius of key compromise and prevents effective incident response.

Recommendation It is recommended to implement a mechanism that would allow to revoke privileges of issued OwnerCaps.

Developers Response The developers acknowledged the issue but chose not to implement code changes, instead relying on stricter operational procedures.

5.1.8 V-NAVP-VUL-008: Irrelevant functions still accessible after upgrade

Severity	Warning	**Commit**	bc86742
Type	Access Control	**Status**	Acknowledged
Location(s)	`lending_core/sources/` ▶ `incentive.move` ▶ `incentive_v2.move` ▶ `incentive_v3.move:1065`		
Confirmed Fix At	N/A		

Description There were several places in the code identified that seem to have left functionality available even though it is no longer relevant.

- The comment in `incentive_v3::get_borrow_fee` states that the function is deprecated; however, it remains callable and will still compute a borrow fee when `incentive.borrow_fee_rate > 0`. Additionally, in cases where the fee is not applicable, the function returns `0` rather than aborting.
- The protocol still includes the older `incentive.move` and `incentive_v2.move` modules whose methods are intended to be replaced by the updated logic defined in `incentive_v3.move` module. This means that the older logic is still accessible publicly to users in the newer version.

Impact The issues defined above can lead to unintended reliance on legacy incentive logic, potentially resulting in inconsistent fee behavior and unexpected protocol outcomes.

Recommendation When deprecating older functionalities, it is recommended to remove the logic from the body of the function and abort with error, or redirect to the newer version if called, to shrink the potential attack surface as many relevant reward functions can still be used by users.

Developers Response In commit f9d7dc4, the developers opted to add clarifying comments indicating that the logic is no longer in use, rather than removing or modifying the underlying code.