GUARDIAN

Valantis

Staking Modules

Security Assessment

November 18th, 2025

Summary

Audit Firm Guardian

Prepared By Wafflemakr, Robert Regeida

Client Firm Valantis

Final Report Date November 18, 2025

Audit Summary

Valantis engaged Guardian to review the security of their Valantis Staking Modules Review. From the 10th of November to the 12th of November, a team of 2 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Confidence Ranking

Given the lack of critical issues detected and minimal code changes following the main review, Guardian assigns a Confidence Ranking of 5 to the protocol. Guardian advises the protocol to consider periodic review with future changes. For detailed understanding of the Guardian Confidence Ranking, please see the rubric on the following page.

Guardian Confidence Ranking

Confidence Ranking	Definition and Recommendation	Risk Profile
5: Very High Confidence	Codebase is mature, clean, and secure. No High or Critical vulnerabilities were found. Follows modern best practices with high test coverage and thoughtful design. **Recommendation:** Code is highly secure at time of audit. Low risk of latent critical issues.	0 High/Critical findings and few Low/Medium severity findings.
4: High Confidence	Code is clean, well-structured, and adheres to best practices. Only Low or Medium-severity issues were discovered. Design patterns are sound, and test coverage is reasonable. Small changes, such as modifying rounding logic, may introduce new vulnerabilities and should be carefully reviewed. **Recommendation:** Suitable for deployment after remediations; consider periodic review with changes.	0 High/Critical findings. Varied Low/Medium severity findings.
3: Moderate Confidence	Medium-severity and occasional High-severity issues found. Code is functional, but there are concerning areas (e.g., weak modularity, risky patterns). No critical design flaws, though some patterns could lead to issues in edge cases. **Recommendation:** Address issues thoroughly and consider a targeted follow-up audit depending on code changes.	1 High finding and ≥ 3 Medium. Varied Low severity findings.
2: Low Confidence	Code shows frequent emergence of Critical/High vulnerabilities (~2/week). Audit revealed recurring anti-patterns, weak test coverage, or unclear logic. These characteristics suggest a high likelihood of latent issues. **Recommendation:** Post-audit development and a second audit cycle are strongly advised.	2-4 High/Critical findings per engagement week.
1: Very Low Confidence	Code has systemic issues. Multiple High/Critical findings (≥5/week), poor security posture, and design flaws that introduce compounding risks. Safety cannot be assured. **Recommendation:** Halt deployment and seek a comprehensive re-audit after substantial refactoring.	≥5 High/Critical findings and overall systemic flaws.

Table of Contents

Project Overview

Project Summary

Project Name	Valantis
Language	Solidity
Codebase	https://github.com/ValantisLabs
Commit(s)	Main Review commit: d66081fb1e0d3fdfed02b82b373471548e7c77c8 Remediation Review Commit: 2d4a1a13b3ec5e8c1f9455cc59ad5ce487b4fea9

Audit Summary

Delivery Date	November 18, 2025
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
● Critical	0	0	0	0	0	0
● High	1	0	0	0	0	1
● Medium	0	0	0	0	0	0
● Low	0	0	0	0	0	0
● Info	6	0	0	2	0	4

Audit Scope & Methodology

Scope and details:

```
contract,source,total,comment
sthype-contracts/src/staking-modules/StakingModuleExternalManagement.sol,124,263,98
source count: {
  total: 263,
  source: 124,
  comment: 98,
  single: 23,
  block: 75,
  mixed: 0,
  empty: 41,
  todo: 0,
  blockEmpty: 0,
  commentToSourceRatio: 0.7903225806451613
}
```

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: *High*	Impact: *Medium*	Impact: *Low*
Likelihood: *High*	🔴 Critical	🟠 High	🟡 Medium
Likelihood: *Medium*	🟠 High	🟡 Medium	🟢 Low
Likelihood: *Low*	🟡 Medium	🟢 Low	🟢 Low

Impact

High	Significant loss of assets in the protocol, significant harm to a group of users, or a core functionality of the protocol is disrupted.
Medium	A small amount of funds can be lost or ancillary functionality of the protocol is affected. The user or protocol may experience reduced or delayed receipt of intended funds.
Low	Can lead to any unexpected behavior with some of the protocol's functionalities that is notable but does not meet the criteria for a higher severity.

Likelihood

High	The attack is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount gained or the disruption to the protocol.
Medium	An attack vector that is only possible in uncommon cases or requires a large amount of capital to exercise relative to the amount gained or the disruption to the protocol.
Low	Unlikely to ever occur in production.

Audit Scope & Methodology

Methodology

Guardian is the ultimate standard for Smart Contract security. An engagement with Guardian entails the following:

- Two competing teams of Guardian security researchers performing an independent review.
- A dedicated fuzzing engineer to construct a comprehensive stateful fuzzing suite for the project.
- An engagement lead security researcher coordinating the 2 teams, performing their own analysis, relaying findings to the client, and orchestrating the testing/verification efforts.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts. Comprehensive written tests as a part of a code coverage testing suite.
- Contract fuzzing for increased attack resilience.

Findings & Resolutions

ID	Title	Category	Severity	Status
H-01	Reverting Burn Recipient Bricks Future Redeems	DoS	● High	Resolved
I-01	SMEM Temporary Underreports Balance	Warning	● Info	Resolved
I-02	SMEM Does Not Emit Any Event	Best Practices	● Info	Resolved
I-03	No Plan For Aligned Quote Assets	Documentation	● Info	Acknowledged
I-04	Withdrawal Amounts Must Be In Wei	Documentation	● Info	Resolved
I-05	Incorrect StHYPE Supply Adjustment	Validation	● Info	Acknowledged

H-01 | Reverting Burn Recipient Bricks Future Redeems

Category	Severity	Location	Status
DoS	● High	Overseer.sol	Resolved

Description PoC

Overseer.burn blindly accepts any to address and simply pushes the request into the burn queue. When someone later calls redeem, the protocol sends native HYPE to the original user and requires the low-level call to succeed.

The queue gating logic _redeemable insists that the contract holds enough idle Hype to cover every previous burn plus the current one. A stuck entry therefore permanently consumes the whole allowance.

An attacker can mint stHYPE, burn it to a contract whose fallback always reverts (or can be toggled to revert) and call redeem. The transfer never succeeds, so completed stays false, redeemed never increases and the jammed burn keeps its amount inside difference.

Because difference + protocolPendingFee must stay ≤ Overseer's idle Hype, any honest user who burns afterwards immediately hits NotRedeemable until governance sources extra liquidity. The attacker can later flip their recipient to accept Hype and exit, so the only cost is temporarily parking capital while everyone else remains blocked.

A dedicated Forge test test/OverseerBurnQueueJammed.t.sol demonstrates the failure end-to-end: a malicious burner locks a 20 Hype burn and a subsequent 30 Hype burn by an honest user reverts until the protocol withdraws fresh Hype from staking modules.

Impact: One reverting recipient can freeze withdrawals for all users, forcing the protocol to keep matching idle liquidity on hand or rely on manual intervention to unblock the queue.

Recommendation

Instead of a low level call to transfer the raw Hype use SafeTransferLib.forceSafeTransferETH. This way, this transfer can never revert.

Resolution

Valantis Team: The issue was resolved in commit 6742792.

I-01 | SMEM Temporary Underreports Balance

Category	Severity	Location	Status
Warning	● Info	StakingModuleExternalManagement.sol	Resolved

Description

StakingModuleExternalManagement.getTotalBalance() only sums the module's HyperCore spot balance and EVM balance together with the stake account's staking balance. Any funds that temporarily sit in the stake account's HyperCore spot balance or are mid-bridge are invisible to this view. That "limbo" period occurs twice.

First, immediately after deposit() the module performs CoreWriterLib.spotSend, which merely enqueues a Core transaction in the current block: the system address already holds the HYPE in EVM, but Core has not yet credited the stake account's balance, so getTotalBalance() still reads the pre-deposit value.

Overseer tries to protect against same-block mistakes by setting stakingModuleLastBlockInteraction inside every deposit/withdraw call and rejecting rebase() in the same block via RebaseLocked, so a deposit and rebase cannot co-exist within one block.

However, there is still at least one full block between the moment funds leave Overseer and the moment Core processes the queued spotSend, and during that block getTotalBalance() continues to show the old value even though native HYPE already left the system. Later, when the external operator moves the stake account's spot balance into staking, the same blind spot reappears because the module never queries that spot balance.

Likewise during withdrawals, undelegated funds live in the stake account's spot balance until the operator forwards them back to the module's HyperCore account, leaving another window where the backing exists but is untracked.

This creates two problems:
• Overseer.getNewSupply() aggregates getTotalBalance() for every module. If a deposit or withdrawal straddles the block boundary between "Overseer already sent HYPE" and "Core has credited the new balance," the reported supply dips while the real assets remain elsewhere. Rebasing during that limbo block fabricates a supply decrease (or subsequent spike) that impacts every stHYPE holder even though it stems from sequencing rather than economics.
• maxRedeemable = balance - totalLiability inherits the same miscount, so large burns queued during a limbo block can be rejected even though HYPE is already en route, again letting timing-sensitive actors grief redemptions.

Recommendation

Consider maintaining a per-block accumulator that adds "limbo" amounts (deposits that have bridged out but not yet hit Core, or withdrawals that have been undelegated but not yet forwarded) and include it inside getTotalBalance() until the corresponding Core transaction settles.

At minimum, encode sequencing rules: e.g., manager should never submit rebase() in the same block as deposit()/requestWithdraw(), so operational scripts cannot accidentally front-run themselves. Either solution turns the temporary mismatch into an accounted liability so supply math and redeemability checks stay monotonic within a block.

Resolution

Valantis Team: The issue was resolved in commit 2d4a1a1.

I-02 | SMEM Does Not Emit Any Event

Category	Severity	Location	Status
Best Practices	● Info	StakingModuleExternalManagement.sol	Resolved

Description

StakingModuleExternalManagement mutates protocol-owned balances (bridging deposits, initiating withdrawals, forwarding native HYPE) without emitting any events. Neither deposit, requestWithdraw, nor withdraw emit structured logs about the amounts, stake account, HyperCore account, or initiating manager.

Even the receive() fallback, which accepts native transfers from previously requested withdrawals, is silent. This makes it impossible for off-chain monitoring, accounting, or incident response to reconstruct capital flows: e.g., when HYPE disappears from Overseer during a deposit, there is no corresponding event proving where it went or whether the stake account acknowledged receipt.

Likewise, withdrawals leave no traces except balance deltas. Since the module hands custody to an external multisig, not emitting events deprives governance and on-chain monitors of the basic audit trail needed to confirm that off-chain agreements are being honored.

Recommendation

Emit structured events for every state-changing action (Deposit, RequestWithdraw, Withdraw) and log inbound HYPE in receive(). Include the manager, stakeAccount, HyperCore account and amount so off-chain indexers can reconcile movements across Overseer, the module, and HyperCore.

Even simple events (e.g., event Deposit(address indexed manager, uint256 amount) and event HyperCoreFundsReceived(uint256 amount)) would provide the minimum observability needed to detect stuck transfers or misbehavior by the external operator.

Resolution

Valantis Team: The issue was resolved in commit 5d61e22.

I-03 | No Plan For Aligned Quote Assets

Category	Severity	Location	Status
Documentation	● Info	StakingModuleExternalManagement.sol	Acknowledged

Description

The protocol plan explicitly assigns 200k HYPE to a StakingModuleExternalManagement instance "to whitelist a quote asset," implying the module will satisfy Hyperliquid's aligned-quote/stable requirements.

However, there is no contract code, config or documented runbook that handles the 1M HYPE stake (200k base + 800k alignment bond), the 50% reserve-yield kickback, or the validator-voted compliance checks described in Hyperliquid's spec:

(https://hyperliquid.gitbook.io/hyperliquid-docs/hypercore/aligned-quote-assets).

Today's module just forwards deposits to a generic stakeAccount and trusts it to "whitelist a quote asset." That may be acceptable if alignment is explicitly out of scope, but the docs already describe a three-year lock and extra yield justified by quote-asset whitelisting, so readers will naturally assume the module is pursuing alignment unless the team says otherwise.

Without instrumentation showing that the extra bond, revenue share, and native minting actually happened, governance and users cannot tell whether the promised fee discounts will ever materialize even though capital is locked under that assumption.

Recommendation

Add documentation (and, if needed, supporting code) that explains how this staking module is supposed to meet Hyperliquid's aligned-quote requirements before user funds are committed to that use case.

Resolution

Valantis Team: Acknowledged.

I-04 | Withdrawal Amounts Must Be In Wei

Category	Severity	Location	Status
Documentation	● Info	StakingModuleExternalManagement.sol: 260	Resolved

Description

The requestWithdraw contains an array of amounts. According to the natspec, this is Array of HYPE amounts to withdraw from each account.

However, the CoreWriterLib.spotSend expects an amount in Wei units and not EVM amount (18 decimals). The natspec does not clearly state the units of this amount param and could cause confusion or incorrect bridged amounts.

Recommendation

Make sure the documentation explicitly states that the HYPE amounts should be in Wei and not Evm amounts.

Resolution

Valantis Team: The issue was resolved in commit 164467d.

I-05 | Incorrect StHYPE Supply Adjustment

Category	Severity	Location	Status
Validation	● Info	StakingModuleExternalManagement.sol: 161	Acknowledged

Description

The `StakingModuleExternalManagement.getTotalBalance()` contains a note that warns manager not to call rebase in the following cases to avoid stHYPE supply adjustments:

• stake account has not deposited the received HYPE into its staking balance

• the stake account's stake balance has been unstaked into spot balance, but not yet sent to this contract's HyperCore spot balance

However, the `rebase` function only prevents executing it in the same block as the last module interaction, and does not check for the staking account HYPE spot balance.

Recommendation

Make sure rebase will wait for any pending tx or transitory state, adding validations in the off chain services.

Resolution

Valantis Team: Acknowledged.

Remediation Findings & Resolutions

ID	Title	Category	Severity	Status
I-01	Receive Function Contains Extra Code	Informational	● Info	Resolved

I-01 | Receive Function Contains Extra Code

Category	Severity	Location	Status
Informational	● Info	StakingModuleExternalManagement.sol: 183	Resolved

Description

The StakingModuleExternalManagement.receive function, now emits HyperCoreFundsReceived event. When bridging from Core, the system address will send HYPE to the contract, with a gas limit of 30,000 gas.

Currently, receiving HYPE and emitting the event will consume around 22k gas. In case that extra code is added to the receive function, bridging from Core can be blocked.

Recommendation

Document this behavior in the contract, to make sure extra care is taken when adding more code in the receive function.

Resolution

Valantis Team: The issue was resolved in commit 9d52ae3.

Disclaimer

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 Guardian 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. Guardian’s position is that each company and individual are responsible for their own due diligence and continuous security. Guardian’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 Guardian 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.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Guardian

Founded in 2022 by DeFi experts, Guardian is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit https://guardianaudits.com

To view our audit portfolio, visit https://github.com/guardianaudits

To book an audit, message https://t.me/guardianaudits