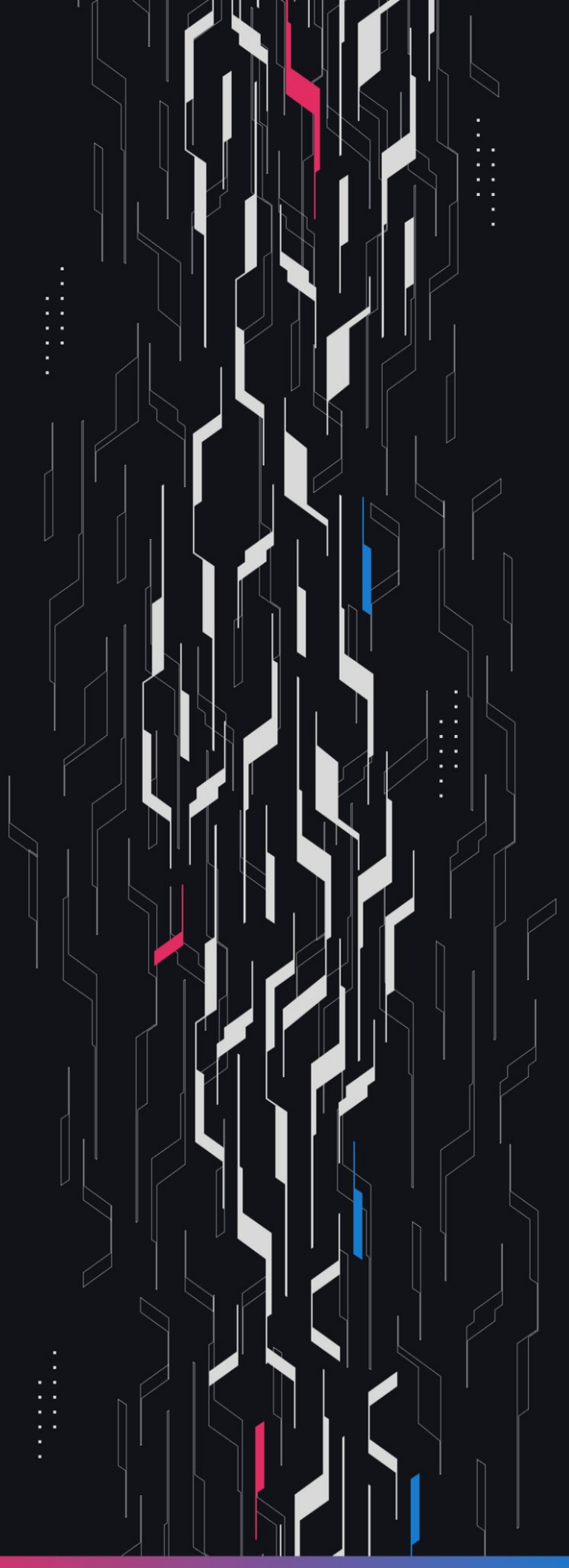
GA GUARDIAN

Valantis

Overseer Review

Security Assessment

January 21st, 2026



Summary

Audit Firm Guardian

Prepared By Robert Regeida, Felipe Gomez

Client Firm Valantis

Final Report Date January 21, 2026

Audit Summary

Valantis engaged Guardian to review the security of their Overseer.Sol. From the 5th of January to the 6th of January, a team of 2 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Confidence Ranking

Given the minimal High and Critical issues detected during the main review, Guardian assigns a Confidence Ranking of 4 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.

✅ Verify the authenticity of this report on Guardian's GitHub: https://github.com/guardianaudits

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 1 Significant issue was uncovered per week. Design patterns are sound, and test coverage is strong. **Recommendation:** Suitable for deployment after remediations; consider periodic review with changes.	0-1 High/Critical findings per engagement week and little to no Medium severity issues. Varied Low 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.	1-2 High/Critical findings per engagement week.
2: Low Confidence	Code shows frequent emergence of Critical/High vulnerabilities. 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. Or additional High/Critical findings uncovered in remediation review which have not been resolved and confirmed by Guardian.
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/sthype-contracts
Commit(s)	Main Review commit: b740c7c71b0952337feeda0ef161742549cebd5e Remediation Review commit: 14c856d6060b755098d12035d94a553a958b61d9

Audit Summary

Delivery Date	January 21, 2026
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	2	0	0	2	0	0
Low	4	0	0	1	0	3
Info	17	0	0	9	1	7

Audit Scope & Methodology

The following diff is in scope:
https://github.com/ValantisLabs/sthype-contracts/compare/2d4a1a13b3ec5e8c1f9455cc59ad5ce487b4fea9...b740c7c71b0952337feeda0ef161742549cebd5e

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	Minting Allowed After Slashing	Logical Error	High	Resolved
M-01	Interim Balance Excluded From Backing	Unexpected Behavior	Medium	Acknowledged
L-01	Legacy Burns Can Revert _redeemable	DoS	Low	Resolved
L-02	Insolvency Can Revert Backing Math	DoS	Low	Acknowledged
L-03	Asymmetric Slashing State Calculation	Logical Error	Low	Resolved
I-01	ERC7201 Slot Change Upgrade Risk	Upgradeability	Info	Acknowledged
I-02	Max Slash Unit Migration Risk	Upgradeability	Info	Acknowledged
I-03	Extreme Slash Can Freeze Rebase Flow	DoS	Info	Acknowledged
I-04	Non-atomic Upgrade Leaves V3 Uninitialized	Upgradeability	Info	Acknowledged
I-05	transferFrom Permits Burns To address(0)	Logical Error	Info	Resolved
I-06	Burns Can Revert During Active Rebase	Logical Error	Info	Acknowledged
I-07	selfDisableTransfer Bypassed By Allowances	Unexpected Behavior	Info	Partially Resolved
I-08	Historical Balance Returns Shares	Unexpected Behavior	Info	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
I-09	Burn To Invalid Recipient Loses Funds	Unexpected Behavior	Info	Resolved
I-10	Redeemable Rounding Can Be Early	Rounding	Info	Resolved
I-11	Unused Error	Superfluous Code	Info	Resolved
I-12	Frontrunning V3 Upgrade	Configuration	Info	Acknowledged

H-01 | Minting Allowed After Slashing

Category	Severity	Location	Status
Logical Error	High	Overseer.sol: 349	Resolved

Description PoC

The `mint` function call sends `msg.value` before checking `pendingSlashExists`. If backing is below expected (slash pending), a minter can "plug" the shortfall with their deposit, making `pendingSlashExists` return false and letting mint proceed without a rebase.

Because `stHYPE` mints 1:1, the minter effectively bails out the deficit and then gets their new tokens slashed later, shifting the loss from existing holders to the minter and bypassing the intended "no mint/redeem until slash is applied" policy.

Recommendation

Decouple pending-slash detection from the current call's inbound funds (`msg.value`).

Resolution

Valantis Team: The issue was resolved in commit 6eeacef.

M-01 | Interim Balance Excluded From Backing

Category	Severity	Location	Status
Unexpected Behavior	Medium	Overseer.sol	Acknowledged

Description

The new accounting for total backing no longer includes the deprecated interimAddress. getTotalBalance only sums the Overseer balance and staking module balances, so any HYPE that remains parked at interimAddress is invisible to the accounting.

That invisible balance makes the protocol believe that backing is lower than expected, which triggers the slashing path in _accountForSlashing and also causes pendingSlashExists to return true. That combination can either over slash holders by reducing supply as if a loss occurred, or block mint and redeem flows until a rebase is forced.

On a forked mainnet upgrade test the interimAddress balance was non zero. The EVM balance was 10999506489256520459 wei (about 10.999506489 HYPE) and the L1 balance was 29130000000000 wei (about 0.00002913 HYPE), for a total of 10999535619256520459 wei (about 10.999535619 HYPE).

With a total supply around 4 million HYPE, that discrepancy is well above the minSlashPercentage threshold, so it would be interpreted as a real slash even though those funds still exist. The impact is incorrect slashing or a protocol wide pause caused by a false pending slash.

Recommendation

Drain interimAddress before the upgrade and verify it is zero, or keep interimAddress in total backing calculations until it is drained. If you want a guardrail, add a pre upgrade check that reverts or warns when the interimAddress balance is non zero.

Resolution

Valantis Team: Acknowledged.

L-01 | Legacy Burns Can Revert _redeemable

Category	Severity	Location	Status
DoS	Low	Overseer.sol	Resolved

Description

The V3 upgrade introduces a per burn mapping called `cumulativeSlashFactor` that is used inside `_redeemable` to compute the current redeemable amount. This mapping is only populated when `burn()` runs in the new logic, so any burns created before the V3 upgrade have `cumulativeSlashFactor` equal to zero.

The `_redeemable` function performs the division by `cumulativeSlashFactor` before checking whether the burn is completed, so a legacy burn ID causes a division by zero revert even if it was already completed. The V3 initializer does not backfill `cumulativeSlashFactor` for historical burns or for the new dummy burn, so this condition persists for old entries. The core issue is the order of operations in `_redeemable`.

The upgrade also has a no pending burns guard, but it only runs if `initializeV3` is actually called:

```
require(burns[burns.length - 1].sum == redeemed, CannotUpgradeWhilePendingBurns());
```

This check prevents pending burns at upgrade time, but it does not initialize `cumulativeSlashFactor` for legacy burns, so the division by zero still occurs for completed historical entries. If the upgrade were executed without calling `initializeV3`, then even the pending burn check would not run.

As a result, `redeemable(burnId)` and `getBurns(account)` revert for accounts with historical burns and `redeem(legacyBurnId)` reverts during its pre check. In the expected upgrade path where `initializeV3` is called, there should be no pending burns, so funds are not stuck.

The persistent impact is a view and integration level DoS for users with legacy burns and any offchain systems that call these view functions. The worst case is a user level lockout from redemption only if an upgrade is executed without calling `initializeV3` or if the pending burn guard is bypassed, because a pending legacy burn would be unrecoverable while the division by zero remains.

Recommendation

Make `_redeemable` safe for legacy entries by returning early when `burns[burnId].completed` is `true` and by guarding zero slash factors. One safe pattern is to read the factor into a local variable and return `false` if it is zero, or treat zero as E18 for legacy burns:

```
if (burns[burnId].completed) return false;
uint256 burnFactor = $.cumulativeSlashFactor[burnId];
if (burnFactor == 0) return false;
```

Resolution

Valantis Team: The issue was resolved in commit 09b6375.

L-02 | Insolvency Can Revert Backing Math

Category	Severity	Location	Status
DoS	Low	Overseer.sol	Acknowledged

Description

Several critical paths subtract `protocolPendingFee` or `totalLiability` directly from `getTotalBalance`. In normal operation this is safe, but after a catastrophic loss it is possible for total backing to fall below `protocolPendingFee` or `totalLiability`.

In that case these subtractions revert, which can block `rebase`, `mint` and `redeem` flows at the moment when the protocol most needs a controlled recovery. This is not an unchecked underflow, but a liveness edge case in an insolvency scenario. It primarily affects incident response and recovery workflows rather than creating a new theft path.

```
uint256 totalStHypeBacking = getTotalBalance() - protocolPendingFee;
```

```
return getTotalBalance() - totalLiability();
```

The impact is that an extreme backing loss can cause core state transitions to revert until governance intervenes.

Recommendation

Consider saturating these subtractions to zero or adding explicit checks that allow a controlled emergency mode instead of reverting. If insolvency is possible, add a governance write down path for `protocolPendingFee` and document an incident response procedure that can restore liveness.

Resolution

Valantis Team: Acknowledged.

L-03 | Asymmetric Slashing State Calculation

Category	Severity	Location	Status
Logical Error	Low	Overseer.sol: 617	Resolved

Description PoC

The `pendingSlashExists` treats `slashPercentage >= minSlashPercentage` as pending, but `_accountForSlashing` only updates `latestCumulativeSlashFactor` when `slashPercentage > minSlashPercentage`.

At equality, the slash is never applied yet the pending flag never clears, leaving mint/redeem permanently blocked (liveness DoS).

Recommendation

Align the comparator between detection and application (use the same inequality or explicitly clear pending at/under the threshold) and add a boundary test for the equality case.

Resolution

Valantis Team: The issue was resolved in commit ff7e607.

I-01 | ERC7201 Slot Change Upgrade Risk

Category	Severity	Location	Status
Upgradeability	Info	StakingModuleExternalManagement.sol	Acknowledged

Description

The storage slot anchor for the `StakingModuleExternalManagement` module was changed to a new `ERC7201` namespace constant. A proxy that was initialized with the old slot keeps its state at the old location, but the new implementation reads from the new location.

If an already deployed proxy is upgraded in place from the old implementation to the new one, rather than being freshly deployed with the new slot, the proxy will appear uninitialized, with manager and stake account reading as zero and deposit cap as zero.

The namespace change is:
• old namespace: `stHYPE.storage.StakingModule`
• new namespace: `stHYPE.storage.StakingModuleExternalManagement`

All `onlyManager` functions will revert, deposits will fail the cap check and any funds already held by the module or its stake account become operationally stuck. In addition, total balance reporting will ignore the real stake account, which can make the protocol think backing has disappeared and may trigger slashing logic or block rebases.

This only happens if an already deployed `StakingModuleExternalManagement` proxy is upgraded in place. The current upgrade script does not upgrade those proxies, and modules 1 through 5 are a different module type, so this does not trigger in the provided flow.

Recommendation

Treat this as an upgrade risk. Do not upgrade existing `StakingModuleExternalManagement` proxies that were initialized with the old slot to the new implementation. Keep the original slot constant for in place upgrades, or only use the new slot for fresh deployments.

Resolution

Valantis Team: Acknowledged.

I-02 | Max Slash Unit Migration Risk

Category	Severity	Location	Status
Upgradeability	Info	Overseer.sol	Acknowledged

Description

The upgrade renames `slashThresholdBps` to `maxSlashPercentage` and changes its unit from basis points to 1e18 scaled percentage, but no migration or conversion is performed. The upgrade scripts pass empty calldata to `upgradeAndCall` and the V3 initializer does not update `maxSlashPercentage`, so any existing non zero value is preserved in storage and interpreted with the new unit.

A prior value like 500 (5 percent in basis points) becomes 500 in 1e18 scale, which is effectively near zero. When a real slashing event occurs, `_accountForSlashing` compares the 1e18 scaled slash percentage against this tiny threshold and reverts, causing rebase to fail.

While a pending slash exists, mint and redeem are blocked and `maxRedeemable` returns zero, so the protocol cannot process exits and supply cannot be updated. On the current fork we observed a pre upgrade value of 0, so the mis scaling does not manifest there; this remains a migration footgun if the value is ever non zero on upgrade.

Recommendation

Migrate the value during upgrade by converting the old basis points value to 1e18 scale, or explicitly reset `maxSlashPercentage` to the intended 1e18 scaled threshold immediately after upgrade via `Overseer.setMaxSlashPercentage(uint256)` (`DEFAULT_ADMIN_ROLE`).

The safest approach is to include a migration step in the `upgradeAndCall` payload or in `initializeV3`, and to assert the post upgrade value is correct.

Resolution

Valantis Team: Acknowledged.

I-03 | Extreme Slash Can Freeze Rebase Flow

Category	Severity	Location	Status
DoS	Info	Overseer.sol	Acknowledged

Description

If the protocol suffers an extreme loss where total backing is essentially zero relative to the expected backing, the slashing factor will round down to zero.

In that case, the computed slash percentage becomes 100%, which exceeds the configured max slash threshold.

The slashing logic reverts and rebase cannot complete. The pending-slash check still returns true, so minting and redeeming are blocked while the system cannot progress via rebase.

This is an edge case, but it creates a freeze where the protocol is stuck until governance intervenes.

Recommendation

Merely informative. Consider documenting this in an emergency playbook.

Resolution

Valantis Team: Acknowledged.

I-04 | Non-atomic Upgrade Leaves V3 Uninitialized

Category	Severity	Location	Status
Upgradeability	Info	Overseer.sol	Acknowledged

Description

The V3 implementation relies on `initializeV3` to set new `ERC7201` storage values, including `latestCumulativeSlashFactor`. If the proxy is upgraded to the new implementation without calling `initializeV3` in the same transaction, key paths can revert because the new slot values remain zero.

For example, burn updates `cumulativeNormalizedBurns` using `latestCumulativeSlashFactor` and `_redeemable` performs divisions that rely on the new per burn factors. With zeroed storage, these divisions revert and burn or redeemable views break until `initializeV3` is executed.

This is primarily an operational risk and appears mitigated by the concrete upgrade flow tested in `test/integration/V3Upgrade.t.sol`, which uses `upgradeAndCall` with `initializeV3`. The issue only manifests if an operator performs a non atomic upgrade or simply skips `initializeV3`.

Recommendation

Enforce an atomic upgrade path that always calls `initializeV3` via `upgradeAndCall` and document this requirement in the upgrade runbook.

Resolution

Valantis Team: Acknowledged.

I-05 | transferFrom Permits Burns To address(0)

Category	Severity	Location	Status
Logical Error	Info	stHYPE.sol: 315	Resolved

Description PoC

The token restricts burns to the BURNER_ROLE and allows pausing burns, but transferFrom does not block a zero-address recipient. As a result, any spender with an allowance, including a holder who self-approves, can call transferFrom with the recipient set to the zero address.

This routes to the internal transfer routine, treats the zero address as a burn, and decreases preSyncSupply and total voting units. This bypasses the burn role and burn pause controls and allows unauthorized burns through the allowance path.

Recommendation

Add the same zero-address recipient check used by transfer to transferFrom, or explicitly gate zero-address burns behind the burner role and burn pause by routing burns through a dedicated burn-only function.

Resolution

Valantis Team: The issue was resolved in commit 43081af.

I-06 | Burns Can Revert During Active Rebase

Category	Severity	Location	Status
Logical Error	Info	stHYPE.sol: 234	Acknowledged

Description

During an active sync interval, total supply is the sum of a base amount (preSyncSupply) plus a linearly accruing rewards component.

The burn path in _transfer always subtracts the full burn amount from preSyncSupply. If a user attempts to burn more than preSyncSupply while rewards are still accruing, the subtraction underflows and reverts.

This means large burns can fail mid interval even though the user balance includes accrued rewards. Because Overseer burns route through stHYPE.burn, a large user exit can be blocked until the interval completes and preSyncSupply is reset by rebase.

The implementation acknowledges this assumption by noting that rewards are not high enough to have to worry about preSyncSupply underflowing, so the issue is mainly a worst case edge scenario.

```
if (to == address(0)) {
preSyncSupply -= SafeCast.toUint96(amount);
}
```

Recommendation

If you want burns to be robust during active intervals, cap the preSyncSupply reduction at its current value and reduce the remaining amount from the rewards component or adjust rewardsToSync accordingly.

Resolution

Valantis Team: Acknowledged.

I-07 | selfDisableTransfer Bypassed By Allowances

Category	Severity	Location	Status
Unexpected Behavior	Info	stHYPE.sol, wstHYPE.sol	Resolved

Description

The `selfDisableTransfer` feature only checks `msg.sender`, not the token owner whose balance is being moved. As a result, a user who sets `selfDisableTransfer` to true can still have tokens moved by any spender that was previously approved.

The spender passes `notSelfDisableTransfer` because it checks the spender address, then `transferFrom` proceeds and moves funds out of the disabled account.

This defeats the intuitive expectation that `selfDisableTransfer` freezes outgoing transfers from the account, at least with respect to already granted allowances. The impact is limited to accounts that have approved spenders, but it can still surprise users who enable `selfDisableTransfer` for self protection.

Recommendation

In `transferFrom`, also enforce that `selfDisableTransfer[from]` is `false` when moving tokens out of an account. If you want stronger semantics, block approvals or clear allowances when a user self disables.

Resolution

Valantis Team: The issue was resolved in commit fc8a082.

I-08 | Historical Balance Returns Shares

Category	Severity	Location	Status
Unexpected Behavior	Info	stHYPE.sol	Acknowledged

Description

The historical balance and supply view functions (`balanceOf(address,uint256)` and `totalSupplyAt(uint256)`) return values from `VotesUpgradeable`, which are stored as shares (voting units), not 18 decimal token balances.

`balanceOf(account, timepoint)` returns `getPastVotes` and `totalSupplyAt(timepoint)` returns `getPastTotalSupply`. Both values are in share units, so any consumer that assumes token units will read mis scaled values.

Converting to token balances would require the historical `totalSupplyRaw` or balance per share at each timepoint, which is not tracked. The impact is limited to offchain consumers and governance analytics that rely on these views without realizing the unit mismatch.

```
function balanceOf(address account, uint256 timepoint) external view returns (uint256) {
return getPastVotes(account, timepoint);
}
function totalSupplyAt(uint256 timepoint) external view returns (uint256) {
return getPastTotalSupply(timepoint);
}
```

Recommendation

Document explicitly that these functions return shares, or rename them to make the unit clear. If historical 18 decimal balances are required, snapshot `totalSupplyRaw` or balance per share at timepoints and convert shares to balances accordingly.

Resolution

Valantis Team: Acknowledged.

I-09 | Burn To Invalid Recipient Loses Funds

Category	Severity	Location	Status
Unexpected Behavior	Info	Overseer.sol	Resolved

Description

The `burn` function does not validate the redemption recipient. A user can pass `address(0)` as the recipient. On `redeem`, the ETH is force sent to that address, so sending to `address(0)` irreversibly burns the ETH.

This is not a permission bypass, but it allows a user to permanently lose their redemption proceeds by mistake.

Recommendation

Validate that the recipient is not the zero address.

Resolution

Valantis Team: The issue was resolved in commit 487b7b5.

I-10 | Redeemable Rounding Can Be Early

Category	Severity	Location	Status
Rounding	Info	Overseer.sol	Resolved

Description

The `redeemable` check computes liabilities up to a `burnId` by multiplying `cumulativeNormalizedBurns` by the latest slash factor with integer division.

Because `cumulativeNormalizedBurns` already stores per burn values rounded down, the extra floor can make the computed sum slightly lower than the true sum of per burn redeemables.

In some scenarios, this can allow a later burn to be deemed redeemable before it actually is by a few wei. The effect is limited to dust level rounding.

```
uint256 sum = ($.cumulativeNormalizedBurns[burnId] * $.latestCumulativeSlashFactor) / E18;
uint256 redeemedHype = ($.normalizedRedeemedHype * $.latestCumulativeSlashFactor) / E18;
uint256 difference = sum < redeemedHype ? 0 : sum - redeemedHype;
```

Recommendation

Compute the difference in normalized units and apply a rounding up conversion to be conservative when checking available balance.

For example, use `Math.mulDiv` with `Rounding.Ceil`. If you want strict burn ordering, track a `nextRedeemableBurnId` pointer and require burns to be redeemed in order.

Resolution

Valantis Team: The issue was resolved in commit 94f9f0e.

I-11 | Unused Error

Category	Severity	Location	Status
Superfluous Code	Info	Overseer.sol	Resolved

Description

The `BelowMinimumBurnAmount` is still declared but never used in the code.

Recommendation

Remove unused error.

Resolution

Valantis Team: The issue was resolved in commit 32e771b.

I-12 | Frontrunning V3 Upgrade

Category	Severity	Location	Status
Configuration	Info	Overseer.sol: 314	Acknowledged

Description

The current V3 upgrade fork test shows the implementation of the actions before/after upgrade. However, there is no pausing enforced, which allows any user to trigger a burn just before the upgrade, forcing the admin to clear the queue before continuing with the upgrade.

Recommendation

Consider enforcing pausing mechanisms before the upgrade.

Resolution

Valantis Team: Acknowledged.

Remediation Findings & Resolutions

ID	Title	Category	Severity	Status
M-01	Min Slash Threshold Skews Losses	Unexpected Behavior	Medium	Acknowledged
L-01	Redemption Pause Also Blocks Mint	Unexpected Behavior	Low	Resolved
I-01	Ops Scripts Mismatch Contract Interface	Best Practices	Info	Acknowledged
I-02	RescueTokens Uses Raw Transfer	Best Practices	Info	Resolved
I-03	Variable Misuse In Rebase Ops Script	Suggestion	Info	Acknowledged
I-04	Redeemable Burns May Still Revert	Unexpected Behavior	Info	Resolved
I-05	Test May Fail Due To Overseer HYPE Balance	Warning	Info	Resolved

M-01 | Min Slash Threshold Skews Losses

Category	Severity	Location	Status
Unexpected Behavior	Medium	Overseer.sol	Acknowledged

Description

The slashing logic only updates the cumulative slash factor when the computed slash percentage meets or exceeds the configured minimum threshold. At the same time, the rebase logic still derives new supply directly from actual backing, so a small backing loss below the threshold can still decrease supply.

Because the cumulative slash factor is not updated in this case, pending burns are not reduced proportionally to that loss, which shifts the dust loss onto current holders rather than distributing it across pending burns. The maximum slash percentage guard is also only applied when the threshold is met, so this path bypasses that guard for small losses.

```
if (slashPercentage >= minSlashPercentage) {
latestCumulativeSlashFactor = latestCumulativeSlashFactor * slashingFactor / 1e18;
}
```

This can lead to a fairness drift where holders absorb tiny losses that pending burns do not, and it weakens the expectation that a zero max slash setting prevents any supply decrease.

Recommendation

Align the threshold behavior with supply updates. If the intent is to ignore dust, then avoid reducing supply for losses below the threshold or explicitly treat the dust as a protocol loss while keeping pending burns consistent. If the intent is that `maxSlashPercentage` equal to zero prevents any decrease, add an explicit guard to stop supply reductions when that setting is in effect.

Resolution

Valantis Team: Acknowledged.

L-01 | Redemption Pause Also Blocks Mint

Category	Severity	Location	Status
Unexpected Behavior	Low	Overseer.sol	Resolved

Description

The `mint` path calls the pending slash check, and that check returns true whenever redemptions are paused. As a result, calling `pauseRedemption` also blocks minting even though there is a separate burn pause mechanism.

This coupling can be intentional for incident response, but it is not obvious from the pause function names and can surprise operators who expect only redemptions to stop.

```
if (isRedemptionPaused()) return true;
...
if (_pendingSlashExists()) revert CannotMintWhilePendingSlash();
```

The behavior means a redemption pause is effectively a global pause for minting, which should be treated as a policy choice rather than an accident.

Recommendation

Document this coupling explicitly. If the intent is to pause redemption while still allowing mint, split the gating conditions so mint checks a dedicated mint pause flag and redemption checks a redemption pause flag.

Resolution

Valantis Team: Resolved.

I-01 | Ops Scripts Mismatch Contract Interface

Category	Severity	Location	Status
Best Practices	Info	rebase.sh, withdraw.sh, mockrebase.sh	Acknowledged

Description

Multiple operational scripts reference contract functions that are not present in the current Overseer and stHYPE interfaces, so running them will revert or target a non existent selector.

In ops/rebase.sh, the script invokes calculateApr with a uint256 argument and rebase with a uint256 argument, but the contract only exposes calculateApr() and rebase() with no parameters.

In ops/withdraw.sh, the script calls withdrawToL1Escrow, which is not part of the Overseer contract.

In mockrebase.sh, the script calls totalAssetSupply on stHYPE and receiveFromL1 on Overseer, neither of which exist, and also uses rebase(uint256) which does not exist.

These scripts can mislead operators into thinking actions were executed successfully when in fact the transactions revert.

Recommendation

Update the scripts to match the deployed ABI and function signatures, or remove or clearly deprecate them to prevent accidental execution.

Treat these scripts as production artifacts by pinning them to the exact deployed commit and ABI, and add a lightweight preflight check that validates selectors against the target contract before sending transactions.

Resolution

Valantis Team: Acknowledged.

I-02 | RescueTokens Uses Raw Transfer

Category	Severity	Location	Status
Best Practices	Info	stHYPE.sol, wstHYPE.sol	Resolved

Description

The `rescueTokens` functions use `IERC20(token).transfer(to, amount)` without checking the return value. Some `ERC20` tokens are non standard and either return false on failure or return no data.

In those cases, a raw transfer can fail silently or revert due to unexpected return data, which undermines the reliability of the admin rescue path.

This is an operator safety issue because it can leave assets stuck or create false confidence that a rescue succeeded.

```
IERC20(token).transfer(to, amount);
```

Recommendation

Use `SafeERC20.safeTransfer` for rescue transfers to handle non standard `ERC20` behavior consistently.

Resolution

Valantis Team: The issue was resolved in commit 9f6cc07.

I-03 | Variable Misuse In Rebase Ops Script

Category	Severity	Location	Status
Suggestion	Info	rebase.sh	Acknowledged

Description

The operational `rebase` script will import some environment variables but also declares some internal ones like `RPC`, `overseer` and `PRIVATE_KEY`.

The following issues arise:

• Script uses the env `rpc` for some calls and `RPC` for others. That means you can end up with different `RPC` endpoints in the same script.

• The `overseer` address is already declared in the environment, but overwritten in the script (currently the same address but could introduce bugs if changes)

• `PRIVATE_KEY` is unused, as the ledger is utilized during `cast send`

Recommendation

Consider using one single `RPC` url, use the overseer address from env and delete the `PRIVATE_KEY` if not used.

Resolution

Valantis Team: Acknowledged.

I-04 | Redeemable Burns May Still Revert

Category	Severity	Location	Status
Unexpected Behavior	Info	Overseer.sol: 577	Resolved

Description

The `redeemable()` only checks balances/liabilities and does not consider pending slashes or redemption pauses.

However, `redeem()` enforces `!_pendingSlashExists()` and pause status. As a result, `redeemable()` can return true while `redeem()` still reverts, misleading UIs/integrations and prompting failed user transactions.

Recommendation

Align `redeemable()` with `redeem()` semantics by incorporating the same gating conditions (`pendingSlashExists` and `isRedemptionPaused`) or document that `redeemable()` assumes no pending slash and no pause status.

Resolution

Valantis Team: Resolved.

I-05 | Test May Fail Due To Overseer HYPE Balance

Category	Severity	Location	Status
Warning	Info	V3Upgrade.t.sol: 155	Resolved

Description

Currently, the `Overseer` contract has the following stats:

• protocol pending fees 1181 HYPE

• total pending burns 66,817 HYPE

• contract balance 69,145 HYPE

Therefore, when running in a forked environment with the current block, the V3 upgrade test will fail when calculating the `remainingToMint` as the contract balance exceeds pending fees + burns, causing an underflow.

Keep in mind this is an issue with the test script, not a contract logic issue.

Recommendation

Consider only minting stHYPE if pending fees + burns exceed overseer's balance:

```
uint contractBalance = address(overseer).balance;
uint totalPendingBurns = overseer.totalPendingBurns();
uint protocolPendingFee = overseer.protocolPendingFee();
if(contractBalance < protocolPendingFee + totalPendingBurns) {
uint256 remainingToMint = protocolPendingFee + totalPendingBurns - contractBalance;
deal(owner, remainingToMint);
overseer.mint{value: remainingToMint}(owner);
}
```

Resolution

Valantis Team: Resolved.

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