

Obsidian
AUDITS

Valantis Security Review

Auditors

0xjuaan

0xSpearmint

April 6, 2026

Introduction

Obsidian Audits

Obsidian audits is a team of top-ranked security researchers, with a publicly proven track record, specialising in DeFi protocols across EVM chains and Solana.

The team has achieved 10+ top-2 placements in audit competitions, placing 1st in competitions for Wormhole, Pump.fun, Yearn Finance, and many more.

Find out more: obsidianaudits.com

Audit Overview

Valantis Prime is a smart contract account system for trading on Hyperliquid. stHYPE is a liquid staking token backed by HYPE staked on HyperCore, allowing holders to earn staking rewards while maintaining liquidity.

Valantis engaged Obsidian Audits to perform a security review of the `StakeAccountManager` and `StakingModuleExternalAccount` contracts, which manage the staking lifecycle for stHYPE-backed sub-accounts. These contracts enable Prime Account holders to receive Hyperliquid's staking-tier trading fee discounts by staking HYPE that belongs to stHYPE. The review was conducted from March 31, 2026 to April 6, 2026.

Scope

Files in scope

Repo	Files in scope
`valantis-account` **Audit commit hash:** d6b5b1aacf27228bf76860e6c7d8542ebafa9028 **Fix review commit hash:** a7396e5b0ef0ef0947bd846020bfe4832841f2f4	src/StakeAccountManager.sol
`sthype-contracts` **Audit commit hash:** 3d5c00ed7cf2fdf5ff11dc30211d39e07aaac2b1 **Fix review commit hash:** c7261830783a99746b06d5c05d20914299299773	src/staking-modules/StakingModuleExternalAccount.sol

Summary of Findings

Severity Breakdown

A total of **5** issues were identified and categorized based on severity:

- **1 Medium severity**
- **1 Low severity**
- **3 Informational**

Findings Overview

ID	Title	Severity	Status
M-01	Stale precompile read in `abortProvisioning` allows abort with non-zero staking balance, causing permanent loss to stHYPE	Medium	Fixed
L-01	Missing delegation lock period check allows silent undelegation failure	Low	Acknowledged
I-01	Cached storage pointer not used in `StakingModuleExternalAccount`	Informational	Fixed
I-02	Redundant deletes of shutdown-only mappings in pre-shutdown functions	Informational	Fixed
I-03	Users cannot configure session settings on trading actions during the active state	Informational	Fixed

Severity Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact

- **High -** leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- **Medium -** leads to a moderate material loss of assets in the protocol or moderately harms a group of users. Alternatively, breaking a core aspect of the protocol's intended functionality
- **Low -** leads to a minor material loss of assets in the protocol or harms a small group of users.

Likelihood

- **High -** attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- **Medium -** the attack/vulnerability requires minimal or no preconditions, but there is limited or no incentive to exploit it in practice
- **Low -** requires highly unlikely precondition states, or requires a significant attacker capital with little or no incentive.

Findings

[M-01] Stale precompile read in `abortProvisioning` allows abort with non-zero staking balance, causing permanent loss to stHYPE

Description

HyperCore precompile state updates (e.g. `STAKING_DEPOSIT` via CoreWriter) are only reflected in precompile reads (`delegatorSummary`) at block N+1. Same-block reads return stale state, which doesn't account for any CoreWriter actions.

`abortProvisioning()` checks that the sub-account has zero staking exposure before deleting managed tracking:

```
PrecompileLib.DelegatorSummary memory summary =
PrecompileLib.delegatorSummary(subAccount);
require(
    summary.delegated == 0 && summary.undelegated == 0 &&
summary.totalPendingWithdrawal == 0,
    StakeAccountManager__InvalidPreState()
);
```

If `stake()` (which submits `STAKING_DEPOSIT` via CoreWriter) executes in block N and an emergency operator calls `abortProvisioning()` in the same block N, the precompile read returns the pre-deposit state (all zeros), allowing the abort to pass.

After abort:

1. SAM deletes the managed sub-account record (`_managedSubAccounts.remove`, `delete _managedSubAccountByAddress`)
2. SAM sets `_abortedProvisioningHypeRecallAllowed[subAccount] = true` for emergency spot HYPE recall
3. The pending `STAKING_DEPOSIT` settles in block N+1, moving HYPE into the `undelegated` staking balance

The staked HYPE (belonging to the stHYPE protocol) is now permanently stranded:

- SAM no longer tracks the sub-account, so no lifecycle functions (`withdrawStake`, `recallHype`, `finalizeDetach`) are available
- `emergencyRecallProvisioningHype` only recovers **spot** HYPE (reads `spotBalance`, executes `SEND_HYPE_TO_EXTERNAL_RECIPIENT`); it cannot access staking balances
- No other code path exists to withdraw staked HYPE on an unmanaged sub-account

Recommendation

Add a constraint to ensure that the subAccount didn't queue any CoreWriter actions in the same block as the precompile read:

```
  function abortProvisioning(address subAccount) external
override nonReentrant {
      _requireEmergencyOperator();
      _requireManagedSubAccount(subAccount);
      _requireProvisioningLifecycleState(subAccount);

+     require(
+         block.number >
HyperCoreSubAccount(payable(subAccount)).lastBlockInteractio
nCheckpoint()
+     );
+
      bool isActiveOnHyperCore =
PrecompileLib.coreUserExists(subAccount);
      if (isActiveOnHyperCore) {
          PrecompileLib.DelegatorSummary memory summary =
PrecompileLib.delegatorSummary(subAccount);
```

This ensures at least one block has elapsed since the last CoreWriter action on the sub-account, guaranteeing the `delegatorSummary` precompile read reflects correct state.

Remediation: Fixed in commit 9f768b4

[L-01] Missing delegation lock period check allows silent undelegation failure

Description

After delegating HYPE to a validator, HyperCore enforces a 1-day lock period during which undelegation is not allowed. This lock period is readable on-chain via the `delegations` precompile, which returns `lockedUntilTimestamp` per validator (in **milliseconds**).

`StakeAccountManager.updateStakeDelegation` passes the undelegation request through to `HyperCoreSubAccount`, which calls `CoreWriterLib.delegateToken(validator, amountWei, true)`. Neither contract checks whether the lock has expired. CoreWriter calls always succeed on the EVM regardless of whether HyperCore accepts or rejects the action, so the transaction and event emission completes without error.

A silently failed undelegation is not immediately detected onchain, the operator only discovers it by reading delegation state in a later block and seeing it unchanged.

Recommendation

Consider adding a check to ensure the lock period has passed in `HyperCoreSubAccount._stakingDelegate`, when undelegating:

```
function _stakingDelegate(bytes memory actionData) private {
    (address validator, uint64 amountWei, bool undelegate) =
abi.decode(actionData, (address, uint64, bool));
    _requireNonZeroAmount(amountWei);
    _requireNonZeroAddress(validator);

    if (undelegate) {
        PrecompileLib.Delegation[] memory delegations =
PrecompileLib.delegations(address(this));
        for (uint256 i = 0; i < delegations.length; i++) {
            if (delegations[i].validator == validator) {
                require(
                    block.timestamp * 1000 >=
delegations[i].lockedUntilTimestamp, "Cannot undelegate due
to 1 day lock period"
                );
                break;
            }
```

```
        }
    }

    _beforeCoreWriter();
    CoreWriterLib.delegateToken(validator, amountWei,
undelegate);
}
```

This converts `block.timestamp` (seconds) to milliseconds before comparing against the HyperCore lock timestamp, and reverts early with a clear error instead of silently failing on HyperCore.

Remediation: Issue acknowledged

[I-01] Cached storage pointer not used in `StakingModuleExternalAccount`

Description

The storage pointer `$` is cached earlier in the following functions, but then not consistently used. The subsequent calls to `_getStakingModuleStorage()` re-derive the storage pointer from inline assembly, which is not necessary.

In `deposit()`:

```
StakingModuleStorage storage $ = _getStakingModuleStorage();
 // line 359 — caches pointer
...
_getStakingModuleStorage().totalManagedStakingBalanceWeiChec
kpoint += amountHyperCore;  // line 383 — re-fetches
```

In `setGroundTruthStakingBalanceCheckpoint()`:

```
StakingModuleStorage storage $ = _getStakingModuleStorage();
 // line 274 — caches pointer
...
amountWei >=
_getStakingModuleStorage().totalManagedStakingBalanceWeiChec
kpoint,  // line 296 — re-fetches
...
_getStakingModuleStorage().totalManagedStakingBalanceWeiChec
kpoint = amountWei;  // line 302 — re-fetches
```

In `updateStakingBalanceCheckpoint()`:

```
StakingModuleStorage storage $ = _getStakingModuleStorage();
 // line 320 — caches pointer
...
_getStakingModuleStorage().totalManagedStakingBalanceWeiChec
kpoint = amountWei;  // line 335 — re-fetches
```

Recommendation:

Use the cached `$` consistently:

- Line 383: `$.totalManagedStakingBalanceWeiCheckpoint += amountHyperCore;`
- Line 296: `amountWei >= $.totalManagedStakingBalanceWeiCheckpoint`
- Line 302: `$.totalManagedStakingBalanceWeiCheckpoint = amountWei;`
- Line 335: `$.totalManagedStakingBalanceWeiCheckpoint = amountWei;`

Remediation: Fixed in commit c726183

[I-02] Redundant deletes of shutdown-only mappings in pre-shutdown functions

Description

Three functions delete `_shutdownUnwindStatusBySubAccount[subAccount]` and `_protectedSpotHypeWeiBySubAccount[subAccount]` at lifecycle stages where these mappings can never hold non-default values:

1. `abortProvisioning()` **(lines 767-768):**

```
delete _shutdownUnwindStatusBySubAccount[subAccount];
delete _protectedSpotHypeWeiBySubAccount[subAccount];
```

Requires state `PROVISIONING` or `PROVISIONING_UNWIND`. Neither mapping is ever written before `SHUTDOWN_UNWIND` / `SHUTDOWN_EXECUTING`, which are unreachable from provisioning states.

2. `startShutdownNotice()` **(lines 801-802):**

```
delete _shutdownUnwindStatusBySubAccount[subAccount];
delete _protectedSpotHypeWeiBySubAccount[subAccount];
```

Requires state `ACTIVE`. `_shutdownUnwindStatusBySubAccount` is first populated in `beginShutdownUnwind()` (requires `SHUTDOWN_NOTICE`). `_protectedSpotHypeWeiBySubAccount` is first populated in `enterShutdownExecution()` (requires `SHUTDOWN_UNWIND`). Neither can be populated while the sub-account is in `ACTIVE`.

3. `cancelShutdownNotice()` **(lines 837-838):**

```
delete _shutdownUnwindStatusBySubAccount[subAccount];
delete _protectedSpotHypeWeiBySubAccount[subAccount];
```

Requires state `SHUTDOWN_NOTICE`. `_shutdownUnwindStatusBySubAccount` is first populated in `beginShutdownUnwind()`, which transitions from `SHUTDOWN_NOTICE` to `SHUTDOWN_UNWIND`. Since `cancelShutdownNotice` also requires `SHUTDOWN_NOTICE`, `beginShutdownUnwind` cannot have been called yet. `_protectedSpotHypeWeiBySubAccount` is even further downstream (requires `SHUTDOWN_UNWIND` → `SHUTDOWN_EXECUTING`).

Recommendation

Consider removing the redundant `delete` statements.

Remediation: Fixed in commit 2fd2237

[I-03] Users cannot configure session settings on trading actions during the active state

Description

`HyperCoreSubAccount` gates all configuration changes (`setSkipSessionSignature`, `setSessionExpiry`, `setAuthorizedSender`, `setPendingAdmin`) behind the per-action admin. Direct external calls to `setSkipSessionSignature`, `setSessionExpiry`, and `setAuthorizedSender` require `msg.sender == adminByActionType[actionType]`.

The `Account.executeOwner()` path for all four routes through `_validateAdminForActionType()`, which requires the caller to be the current admin, or when admin is `address(0)`, the Account owner via `executeOwner()`. On a normal sub-account, `adminByActionType` defaults to `address(0)`, meaning the Account owner can configure freely via `executeOwner()`.

```
// HyperCoreSubAccount.sol
function _validateAdminForActionType(ExecuteActionType
actionType, address accountSender, bool isAccountSender)
    private
    view
{
    address admin = adminByActionType[actionType];
    // If admin is set, only that admin can configure
    // If admin is not set, must go through Account (owner
via executeOwner)
    require(admin == accountSender || (admin == address(0)
&& isAccountSender), HyperCoreSubAccount__Unauthorized());
}
```

When `StakeAccountManager` deploys a sub-account via `createManagedSubAccount()`, it sets itself as `adminByActionType` for all 16 action types and retains this role until `finalizeDetach()`. During `completeProvisioning()`, SAM releases `authorizedSenderByActionType` to `address(0)` for non-staking actions, giving the user execution rights for trading. However, admin is not released.

```
// StakeAccountManager.sol — completeProvisioning()
managed.state = LifecycleState.ACTIVE;
_protocolFeeEligibleSubAccounts[subAccount] = true;

// Grant execution rights (but not admin) to Account path
for non-staking actions.
HyperCoreSubAccount subAccount_ =
HyperCoreSubAccount(payable(subAccount));

_setAuthorizedSenderByMask(subAccount_,
ACTION_MASK_ACTIVE_NON_STAKING, address(0));
```

This creates a gap: the user can execute trading actions but cannot configure `skipSessionSignature` or `sessionExpiry` on those same actions. These values are fixed to whatever was set at deployment or modified by the operator before `completeProvisioning()`. After entering ACTIVE state, `_requireOperatorSessionConfigMutable()` also blocks the operator from changing them.

The user cannot call `setSkipSessionSignature` to disable agent execution on specific trading actions. The user also cannot call `setSessionExpiry` to adjust signature validity for individual trading actions.

Recommendation

Consider documenting that SAM-managed sub-account users cannot modify `skipSessionSignature` or `sessionExpiryByActionType` on their own trading actions for the duration of the managed lifecycle.

Remediation: Fixed in commit 00d0dcb