

Obsidian
AUDITS

Valantis Security Review

Auditors

0xjuaan

0xSpearmint

21 February, 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 Account is a modular smart-contract account designed for secure interaction with HyperCore and other protocols through dedicated sub-accounts with explicit business logic, with agents and time-bounded sessions enabling delegated execution for improved UX.

Valantis engaged Obsidian Audits to perform a security review of the smart contracts in the `valantis-account` repo. The review was conducted from 9th February 2026 to 17th February 2026.

Scope

Files in scope

Repo	Files in scope
`valantis-account` **Commit hash:** 60a84f4ddc9ef885c531376370d7786e346fd292	src/Account.sol src/HyperCoreSubAccount.sol src/STEXLendingSubAccount.sol

Summary of Findings

Severity Breakdown

A total of **7** issues were identified and categorized based on severity:

- **1 Medium severity**
- **6 Informational**

Findings Overview

ID	Title	Severity	Status
M-01	Agent can spend account allowance beyond session-signed token amounts	Medium	Fixed
I-01	IERC2612.permit can potentially fail if the permit signature is already used	Informational	Fixed
I-02	selfRevokeAdminForActionType does not clear pendingAdminByActionType	Informational	Fixed
I-03	Missing BorrowLend Support in HyperCoreSubAccount	Informational	Fixed
I-04	Repetitive calling of `_requireCoreUserExists()` in the `HyperCoreSubAccount`	Informational	Acknowledged
I-05	Inconsistent event emission when incrementing session nonce	Informational	Fixed
I-06	Unused error	Informational	Fixed

Severity Classification

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

Severity	Impact: High	Impact: Medium	Impact: Low
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] Agent can spend account allowance beyond session-signed token amounts

Summary

In `HyperCoreSubAccount`, the `BRIDGE_TO_CORE` action accepts a non-zero `_amount` parameter from `Account.execute()`, but never validates it against the `evmAmount` in the session-signed `actionData`. An agent can pair a session-signed bridge for a small amount with an arbitrarily large `ExecuteParams.amount`, causing the Account to deduct excess allowance and transfer excess tokens to the sub-account, exceeding what the session signer authorized.

Vulnerability Detail

When `Account.execute()` is called by an agent, `Account._execute()` performs two operations using `ExecuteParams`:

1. Deducts `ExecuteParams.amount` of `ExecuteParams.token` from the sub-account's allowance
2. Transfers that amount to the sub-account

These values are entirely agent-controlled and are **not covered by any signature**.

HyperCoreSubAccount

In `HyperCoreSubAccount.execute()`, the session account signs:

```
Action(uint256 actionType, bytes actionData, uint256 nonce, uint256 sessionEpochId)
```

For `BRIDGE_TO_CORE`, the signed `actionData` contains `(address token, uint256 evmAmount)`. However, the sub-account never validates that `_amount` (from Account) matches `evmAmount` (from the signed data):

```
// HyperCoreSubAccount.sol#L554-L556
// Only non-BRIDGE_TO_CORE actions enforce _amount == 0
if (action.actionType != ExecuteActionType.BRIDGE_TO_CORE) {
    _requireZeroAmount(_amount);
}

// HyperCoreSubAccount.sol#L565-L568
} else if (action.actionType == ExecuteActionType.BRIDGE_TO_CORE) {
    // _amount is never validated against evmAmount in actionData
    _bridge(action.actionData, false);
}
```

Inside `_bridge`, only the signed `evmAmount` is used for the actual bridge call:

```
// HyperCoreSubAccount.sol#L644-L653
(address token, uint256 evmAmount) = abi.decode(actionData, (address,
uint256));
CoreWriterLib.bridgeToCore(token, evmAmount);
```

Example scenario:

1. Session signer authorizes bridging 100 USDC (`evmAmount = 100e6` in signed `actionData`)
2. Agent calls `Account.execute()` with `ExecuteParams{token: USDC, amount: 10_000e6, ...}`
3. Account deducts 10,000 USDC from allowance and transfers 10,000 USDC to sub-account
4. Sub-account bridges only 100 USDC (the signed amount)
5. 9,900 USDC sits idle in the sub-account, and the Account's allowance is drained by 100x what the session signer intended

Additionally, since `_token` is completely ignored by `HyperCoreSubAccount` (parameter name is commented out at line 420-421), the agent controls which token's allowance is consumed at the Account level regardless of what token the signed action actually operates on.

Recommendation

HyperCoreSubAccount

Validate that `_amount` and `_token` match the signed `evmAmount` and `token` when called through Account:

```
} else if (action.actionType == ExecuteActionType.BRIDGE_TO_CORE) {
+    if (isAccountSender) {
+      (address token, uint256 evmAmount) = abi.decode(action.actionData,
(address, uint256));
+      require(_amount == evmAmount);
+     require(_token == token);
+   }
    _bridge(action.actionData, false);
}
```

The `isAccountSender` guard ensures direct callers (authorized senders bypassing Account) are unaffected, since `_amount` is only meaningful when Account transfers tokens through its allowance system.

Remediation: Fixed in commit a83d3db

[I-01] IERC2612.permit can potentially fail if the permit signature is already used

Description

The `Account.sendToOwner()` function uses an `IERC2612.permit` call to gain approval of the user's tokens:

```
IERC2612(_params.token)
    .permit(
        owner_, address(this), _params.amount, _params.deadline,
_permitSig.v, _permitSig.r, _permitSig.s
    );
}
```

The issue with this is that if the permit signature was somehow made public, anyone can directly call `permit()` on the token contract, causing `Account.sendToOwner()` to fail when called since `.permit()` would revert.

Recommendation

Consider wrapping the call in a try-catch block to ensure that even if the permit signature has already been consumed, the `Account.sendToOwner()` call will still succeed.

Remediation: Fixed in commit 2274965

[I-02] selfRevokeAdminForActionType does not clear pendingAdminByActionType

Description

When an admin calls `selfRevokeAdminForActionType`, `adminByActionType[actionType]` is deleted but `pendingAdminByActionType[actionType]` is not. A previously nominated pending admin can still call `acceptAdminForActionType()` and seize the admin role after the original admin has appeared to relinquish control.

Furthermore, the owner cannot cancel the stale pending admin directly, since setting `pendingAdmin` to `address(0)` via `SET_PENDING_ADMIN_BY_ACTION_TYPE` reverts because the check `require(pendingAdmin != adminByActionType[actionType])` fails when both are `address(0)`. A multi-step workaround to this does exist (owner overwrites the pending admin with their own address, accepts, then self-revokes).

Recommendation

Add `delete pendingAdminByActionType[actionType]` in `selfRevokeAdminForActionType` function so that pending admin nominations are cleared when the current admin relinquishes control.

Remediation: Fixed in commit 4ef3104

[I-03] Missing BorrowLend Support in HyperCoreSubAccount

Description

`HyperCoreSubAccount` does not support corewriter's borrow lend operation.

This operation is currently only available on the testnet CoreWriter and will likely ship to mainnet soon. Since Accounts are not upgradeable, existing deployments will have no way to access borrow/lend functionality.

Recommendation

Consider adding a `BORROW_LEND` entry to `ExecuteActionType` and a corresponding handler in `execute()` that calls:

```
CoreWriterLib.borrowLend(uint8 encodedOperation, uint64 token, uint64
amountWei)
```

Remediation: Fixed in commit 94718da

[I-04] Repetitive calling of `_requireCoreUserExists()` in the `HyperCoreSubAccount`

Description

In the `HyperCoreSubAccount`, the `_beforeCoreWriter()` function is called before all CoreWriter actions.

`_requireCoreUserExists(address(this))` is called within `_beforeCoreWriter()`, so the `coreUserExists` precompile is called for each CoreWriter action.

This is unnecessarily since the account existence only needs to be checked once, so can be checked when calling `setAccount()` on the `HyperCoreSubAccount`, instead of checking it each time a CoreWriter action is called.

Recommendation

Consider removing the call to `_requireCoreUserExists(address(this))` from `_beforeCoreWriter()` and including it in `HyperCoreSubAccount.setAccount()` instead.

Remediation: Issue acknowledged

[I-05] Inconsistent event emission when incrementing session nonce

Description

In `Account::_validateSessionSignature()`, the `sessionNonce` is incremented but the `SessionNonceIncremented` event is not emitted.

This deviates from the logic in `_incrementSessionNonce()` which emits the event:

```
function _incrementSessionNonce() private {
    sessionNonce++;

    emit SessionNonceIncremented(sessionNonce);
}
```

Recommendation

Consider using `_incrementSessionNonce()` instead of manually incrementing the `sessionNonce` variable in `Account::__validateSessionSignature()`

Remediation: Fixed in commit b6f9f41

[I-06] Unused error

Description

The error `Account__SendFromOwnerFeeNotAllowed` is declared in the `Account` contract but is not used by the contract.

Recommendation

Remove the error declaration from the `Account` contract

Remediation: Fixed in commit b6148a1