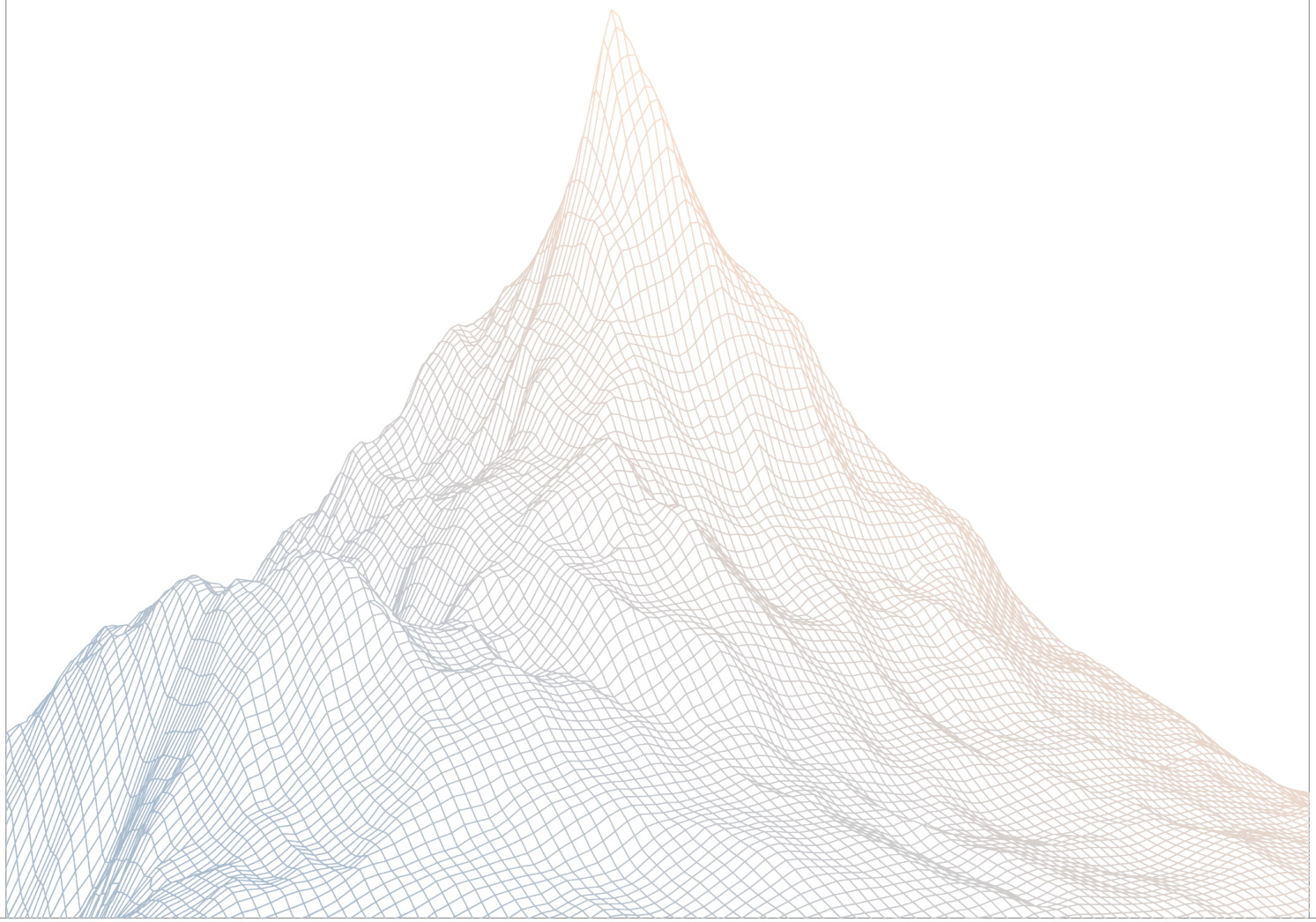
Zenith

Trove Markets

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: January 6th to January 7th, 2026

AUDITED BY: ether_sky
zzykxx

Contents

1 Introduction

1.1 About Zenith

Zenith assembles auditors with proven track records: finding critical vulnerabilities in public audit competitions.

Our audits are carried out by a curated team of the industry's top-performing security researchers, selected for your specific codebase, security needs, and budget.

Learn more about us at https://zenith.security.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2

Executive Summary

2.1 About Trove Markets

Trove is the first perp DEX for collectibles, RWAs, equities, and prediction markets. You can 5x short a Birkin or 3x long Charizard's right now. Trove creates accurate pricing data for illiquid markets (through heavy data aggregation and a weighted price discovery method), and converts this data into indexes which become real tradable perpetuals market.

2.2 Scope

The engagement involved a review of the following targets:

Target	sale-a
Repository	https://github.com/P123D456/sale-a
Commit Hash	`340b00281d1aa1a29297c8d3b4946bf0944775fe`
Files	`src/Sale.sol`

Target	Mitigation Review
Repository	https://github.com/P123D456/sale-a
Commit Hash	`9c34802647b338cca932472b5aa1e26336fb5690`
Files	`src/Sale.sol`

2.3 Audit Timeline

January 6, 2026	Audit start
January 7, 2026	Audit end
January 7, 2026	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	0
Informational	3
Total Issues	**3**

3

Findings Summary

ID	Description	Status
I-1	Refunding would be impossible if the owner withdraws the tokens	Acknowledged
I-2	The NativeSupportedSet event is not used	Resolved
I-3	Use Ownable2StepUpgradeable instead of OwnableUpgradeable	Resolved

4

Findings

4.1 Informational

A total of 3 informational findings were identified.

[I-1] Refunding would be impossible if the owner withdraws the tokens

SEVERITY: Informational	IMPACT: Informational
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- Sale.sol

Description:

The `owner` can `withdraw` tokens.

- Sale.sol#L182

```
function withdraw(address token, uint256 amount)
    external nonReentrant onlyOwner whenSaleEnded {
    if (token == NATIVE) payable(_msgSender()).sendValue(amount);
    else IERC20(token).safeTransfer(_msgSender(), amount);
}
```

If so, the `refund` could revert due to a lack of `tokens`.

Recommendations:

The `owner` can transfer tokens to enable `refunds`. However, for the `native token`, this is not possible because the `receive` and `fallback` functions revert. Consider updating the `receive` function, or documenting this limitation instead.

Trove: Acknowledged. All users will be refunded before withdrawing the assets.

[I-2] The NativeSupportedSet event is not used

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- Sale.sol

Description:

The `NativeSupportedSet` event is defined but not used.

- Sale.sol#L27

```
event NativeSupportedSet(bool isNativeSupported);
```

Recommendations:

Remove this event.

Trove: Resolved with `@583b060f17 ...`

Zenith: Verified.

[I-3] Use `Ownable2StepUpgradeable` instead of `OwnableUpgradeable`

`SEVERITY:` Informational	`IMPACT:` Informational
`STATUS:` Resolved	`LIKELIHOOD:` Low

Target

- Sale.sol

Description:

The `Sale.sol` contract uses Openzeppelin's OwnableUpgradable contract to manage ownership, which exposes the transferOwnership() function as the way to change owner.

The function takes as input the new owner and assigns the input varibale directly to the storage variable `$._owner`, in case the specified new owner is the wrong address this can result in the ownership of the contract being lost forever.

Recommendations:

It's best practice to use the Ownable2StepUpgradeable contract instead, which exposes two functions to change ownership:

- transferOwnership(): sets the pending owner to the specified input address.
- acceptOwnership(): called by the current pending owner to claim ownership of the contract.

This prevents the possibility of losing ownership of the contract if the wrong address is passed to transferOwnership().

Trove: Resolved with `@9c34802647 ...`

Zenith: Verified.