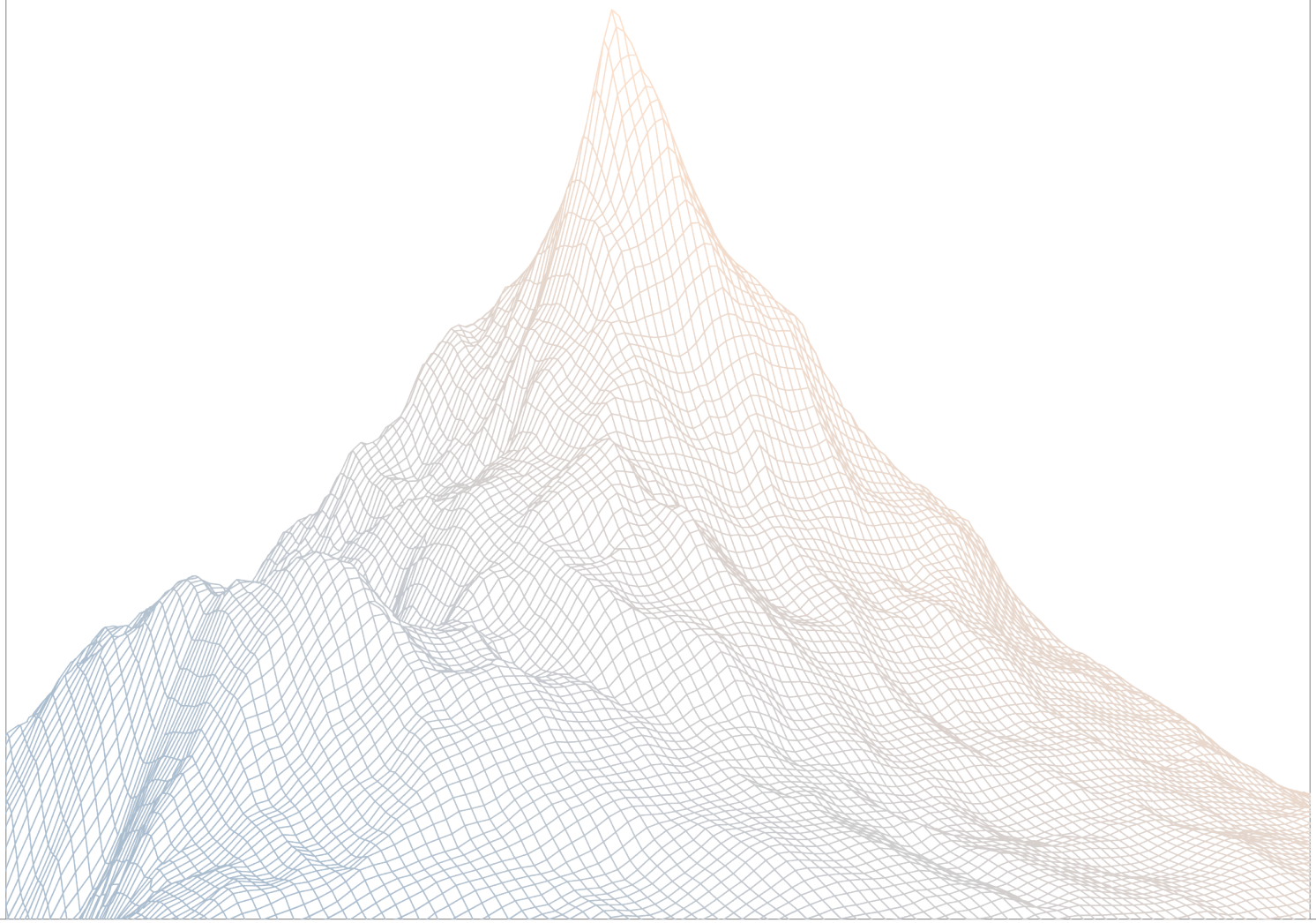
Zenith

Forge

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: September 4th to September 9th, 2025

AUDITED BY: J4X
Mario Poneder

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 Forge

ForgeYields develops the Starknet Vault Kit, an open-source framework for building ERC-4626 vaults and allocators on Starknet. It provides secure, modular infrastructure that other teams can use to launch and manage yield strategies.

2.2 Scope

The engagement involved a review of the following targets:

Target	starknet_vault_kit
Repository	https://github.com/ForgeYields/starknet_vault_kit
Commit Hash	`515fb28ad140f20211c8f9f3e9e15f986ca62865`

Files	`vault_allocator/src/vault_allocator/*`
	`vault_allocator/src/manager/manager.cairo`
	`vault/src/vault/*`
	`vault/src/redeem_request/*`

2.3 Audit Timeline

September 4, 2025	Audit start
September 9, 2025	Audit end
September 15, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	1
Low Risk	2
Informational	2
Total Issues	**5**

3

Findings Summary

ID	Description	Status
M-1	report() could intentionally be blocked by calling bring_liquidity()	Acknowledged
L-1	Redeem Fees are rounded down	Resolved
L-2	Insufficient role account separation	Acknowledged
I-1	report() will break for tokens blocking 0 value transfers	Resolved
I-2	Empty Merkle proofs are accepted	Resolved

4

Findings

4.1 Medium Risk

A total of 1 medium risk findings were identified.

[M-1] `report()` could intentionally be blocked by calling `bring_liquidity()`

SEVERITY: Medium	IMPACT: Medium
STATUS: Acknowledged	LIKELIHOOD: Medium

Target

- packages/vault/src/vault/vault.cairo#L775-L782

Description:

The `bring_liquidity()` function is currently not access-restricted.

```
/// Bring assets back from allocators to vault buffer
/// Used by allocators to return assets for redemptions or rebalancing
// @audit - This can be called by anyone, leaving aum in the allocator
    forever
fn bring_liquidity(
    ref self: ContractState, amount: u256,
) { // Amount of assets to bring back
    ERC20ABIDispatcher { contract_address: self.erc4626.asset() }
        .transfer_from(get_caller_address(),
    starknet::get_contract_address(), amount);
    self.buffer.write(self.buffer.read() + amount); // Increase buffer
    self.aum.write(self.aum.read() - amount); // Decrease deployed AUM
}
```

It allows a caller to manipulate the saved `aum` by donating to the contract. This leads to an issue as `report()` will revert if the saved `aum` differs too much from the one reported by the allocator.

```
// 1) Validate AUM change is within acceptable bounds
if (prev_aum.is_non_zero()) {
    let abs_diff = if (new_aum ≥ prev_aum) {
```

```
            new_aum - prev_aum
        } else {
            prev_aum - new_aum
        };
        // Calculate percentage change: (abs_diff * 1e18) / prev_aum
        let mut delta_ratio_wad = (abs_diff * WAD) / prev_aum;
        if ((abs_diff * WAD) % prev_aum).is_non_zero() {
            delta_ratio_wad += 1; // Round up for safety
        }
        if (delta_ratio_wad > self.max_delta.read()) {
            Errors::aum_delta_too_high(delta_ratio_wad, self.max_delta.read());
        }
    } else if (new_aum.is_non_zero()) {
        Errors::invalid_new_aum(new_aum);
    }
```

As a result, an attacker could call the `bring_liquidity()` function and donate enough to trigger the delta max, which will continuously block the allocator from calling report() and thus block all withdrawals.

Recommendations:

We recommend restricting `bring_liquidity()` so it can only be called by the allocator.

Forge: Acknowledged.

4.2 Low Risk

A total of 2 low risk findings were identified.

[L-1] Redeem Fees are rounded down

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

Target

- packages/vault/src/vault/vault.cairo#L508

Description:

The redemption fees are currently rounded down. This results in minimally less fees being charged than intended in the case of rounding.

```
let redeem_fees = if (owner == fees_recipient) {
    0
} else {
    self.redeem_fees.read()
};
let fee_shares = (shares * redeem_fees)
    / WAD; // Fee calculation:  shares * fee_rate / 1e18
```

Recommendations:

We recommend rounding the fees up.

Forge: Resolved with @abab34e402...

Zenith: Verified.

[L-2] Insufficient role account separation

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Medium

Target

- packages/vault_allocator/src/manager/manager.cairo#L79-L80
- packages/vault/src/vault/vault.cairo#L223-L226

Description:

Both the `Vault` and `Manager` contracts violate the principle of least privilege by granting multiple critical roles (`OWNER_ROLE`, `PAUSER_ROLE` and `ORACLE_ROLE`) to the same initial owner address during contract initialization. This creates a single point of failure and reduces the security posture of the entire system.

Recommendations:

It is recommended to implement role separation in the constructors, i.e. modify constructors to accept separate addresses for different roles.

Forge: We acknowledge the point, but won't change the constructors. Separating roles at deployment time does not materially improve security since the `OWNER` can grant/revoke roles and set role admins immediately after deployment.

Zenith: Acknowledged.

4.3 Informational

A total of 2 informational findings were identified.

[I-1] `report()` will break for tokens blocking 0 value transfers

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

Target

- packages/vault/src/vault/vault.cairo#L716

Description:

After the buffer has been used to satisfy withdrawals, the remaining buffer will be transferred back to the allocator.

```
// 6) Deploy remaining buffer if all epochs are handled
if (new_handled_epoch_len == new_epoch) {
    let alloc = self.vault_allocator.read();
    if (alloc.is_zero()) {
        Errors::vault_allocator_not_set();
    }
    // Deploy all remaining buffer to allocator
    ERC20ABIDispatcher { contract_address: self.erc4626.asset() }
        .transfer(alloc, remaining_buffer);
    self.aum.write(new_aum + remaining_buffer); // Update AUM to include
    deployed assets
    self.buffer.write(0); // Buffer is now empty
} else {
    self.aum.write(new_aum); // Keep buffer for pending redemptions
    self.buffer.write(remaining_buffer);
}
```

However, if the buffer was the exact amount needed for the withdrawals, this would lead to a zero value transfer. For most erc20 this won't be an issue as 0 transfers should be accepted per default, however for some special implementations calls will revert.

Recommendations:

We recommend skipping the transfer if `remaining_buffer == 0`

Forge: Resolved with @77e384c6e9...

Zenith: Verified.

[I-2] Empty Merkle proofs are accepted

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

Target

- packages/vault_allocator/src/manager/manager.cairo#L306
- OpenZeppelin/cairo-contracts/packages/merkle_tree/src/merkle_proof.cairo#L42-L50

Description:

The `_verify_manage_proof` function in the `Manager` contract does not enforce a minimum proof length before calling `merkle_proof::verify_pedersen`, and the OpenZeppelin Merkle proof verification function accepts empty proofs and returns `true` when `root == leaf_hash`, which is typically not an intended use case.

Recommendations:

It is recommended to implement explicit validation to ensure proofs are non-empty before performing Merkle verification, or to enforce a minimum proof length based on the expected Merkle tree depth.

Forge: Resolved with @9c84f50287....

Zenith: Verified..