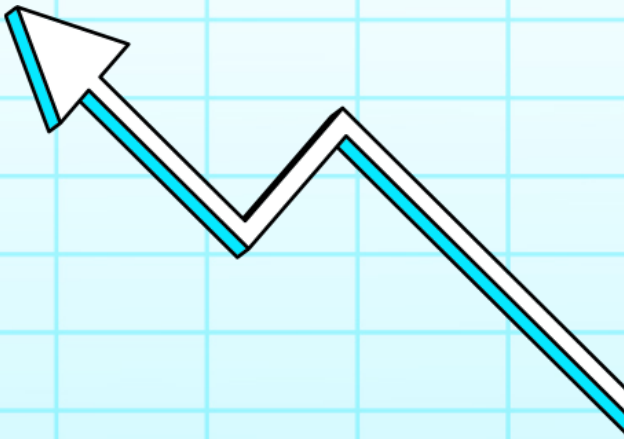
shieldify



Colb Finance

Vault Extended

SECURITY REVIEW

Date: 21 September 2025

CONTENTS

1. **About Shieldify** 3
2. **Disclaimer** 3
3. **About Colb Finance - Vault Extended** 3
4. **Risk classification** 3
   - 4.1 Impact 3
   - 4.2 Likelihood 4
5. **Security Review Summary** 4
   - 5.1 Protocol Summary 4
   - 5.2 Scope 4
6. **Findings Summary** 4
7. **Findings** 5

1. About Shieldify

Positioned as the first hybrid Web3 Security company, Shieldify shakes things up with a unique subscription-based auditing model that entitles the customer to unlimited audits within its duration, as well as top-notch service quality thanks to a disruptive 6-layered security approach. The company works with very well-established researchers in the space and have secured multiple millions in TVL across protocols, also can audit codebases written in Solidity, Vyper, Rust, Cairo, Move and Go.

Learn more about us at shieldify.org.

2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Shieldify Security against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

3. About Colb Finance - Vault Extended

Colb is the first native non-custodial tokenization solution that enables peerless access to Swiss-grade wealth management strategies, pre-IPO opportunities, and premium investment funds. It offers a bankruptcy remote Trust structure and native ownership of real-world assets, all on-chain. Colb reduces the entrance threshold to such investments by removing constraints such as the minimum investment amount to get exposure to them. The protocol is designed with security at its core, boasting compliance with Swiss regulations and DeFi composability. Colb envisions a future rooted in transparency where every individual has equitable access to premium RWA investments.

The single Vault contract seamlessly manages the full lifecycle of the Pre-IPO investment vault, from configuration, deposits, token minting, to withdrawals — with admin control and future upgradeability built-in.

4. Risk Classification

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

4.1 Impact

- **High** - results in a significant risk for the protocol's overall well-being. Affects all or most users
- **Medium** - results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** - losses will be limited but bearable - and covers vectors similar to griefing attacks that can be easily repaired

4.2 Likelihood

- **High** – almost certain to happen and highly lucrative for execution by malicious actors
- **Medium** – still relatively likely, although only conditionally possible
- **Low** – requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

5. Security Review Summary

The security review lasted 5 days with a total of 40 hours dedicated to the audit by the Shieldify team.

Overall, the code is very well-written. The audit report contributed by identifying one Medium and three Low severity issues, mainly a potential vault tokens trap, several incorrect event emissions and centralization concern about the mint vs. reimburse.

The Colb Finance team has done a great job with their test suite and provided support and responses to all of the questions that the Shieldify researchers had.

5.1 Protocol Summary

Project Name	**Colb Finance - Vault Extended**
Repository	SmartContracts
Type of Project	RWA, Pre-IPO, Vault
Audit Timeline	5 days
Review Commit Hash	d569a17dfdf96fc029d2664beafa3dd1a54495fc
Fixes Review Commit Hash	a78b1f472e11015bb5dfc54698dc9536a512df7a

5.2 Scope

The following smart contracts were in the scope of the security review:

File	**nSLOC**
contracts/vault/Vault.sol	483
contracts/vault/VaultFactory.sol	129
Total	**612**

6. Findings Summary

The following number of issues have been identified, sorted by their severity:

- **Medium** issues: **1**
- **Low** issues: **3**
- **Info** issues: **1**

ID	Title	Severity	Status
[M-01]	Vault Tokens Can Be Trapped If User Is Removed from Whitelist	Medium	Acknowledged
[L-01]	Incorrect Amount Emitted in `emitCancelDeposit()` Event During `withdraw()` Function	Low	Fixed
[L-02]	Vault ID Used Before Emitting Event - Off By One Risk	Low	Fixed
[L-03]	Operator Has Full Control Over Mint vs. Reimburse - Relying Entirely on Ratio Bounds	Low	Acknowledged
[I-01]	Missing Validation on `investmentEnd()` in `VaultFactory` Allows Dead-On-Arrival Vaults	Info	Fixed

7. Findings

[M-01] Vault Tokens Can Be Trapped If User Is Removed from Whitelist

Severity

Medium Risk

Description

When `onlyWhitelistTransfer == true`, the `_beforeTokenTransfer()` hook blocks transfers **to or from** any address not on the whitelist.

If a user is removed from the whitelist after claiming tokens, they can no longer:

- Transfer their vault tokens
- Call `requestWithdraw()`, which internally transfers tokens to `address(this)`

This results in a permanent lock of the user's funds.

Location of Affected Code

File: contracts/vault/Vault.sol#L734

```
function _beforeTokenTransfer(...) internal override {
    // code
    if (onlyWhitelistTransfer) {
        bool can = (whitelist.contains(from) || from == address(0) ||
           from == address(this))
                && (whitelist.contains(to) || to == address(0) || to ==
                   address(this));
        if (!can) revert OnlyWhitelistUserCanTransfer();
    }
}
```

File: contracts/vault/Vault.sol#L491

```
function requestWithdraw(uint256 amount) external whenNotPaused {
    address user = msg.sender;
    uint256 userBalance = balanceOf(user);

    if (userBalance < amount) revert InsufficientAmount();

    _transfer(user, address(this), amount);

    uint256 index = nextWithdrawalId++;
    withdraws[index] = WithdrawInfo(
        user,
        uint32(block.timestamp),
        false,
        true,
        amount
    );

    usersWithdraw[user].push(index);
    withdrawToProcess.push(index);

    if (usersWithdraw[user].length >= MAX_PENDING_REQUESTS)
        revert TooManyWithdrawals();

    factory.emitRequestWithdraw(user, amount, index);
}
```

Recommendation

Exclude `address(this)` from `from`-side whitelist check to allow `requestWithdraw()` even for non-whitelisted users

Team Response

Acknowledged.

[L-01] Incorrect Amount Emitted in `emitCancelDeposit()` Event During `withdraw()` Function

Severity

Low Risk

Description

In the `Vault.withdraw(uint256 amount)` function, when a user withdraws tokens before the investment period ends, the contract processes multiple individual deposits to fulfil the total withdrawal amount. However, during this process, the contract emits the `emitCancelDeposit()` event using the total withdrawal amount (`amount`) for every deposit ID that is partially or fully used to fulfil the withdrawal.

This results in event logs that overstate the withdrawn amount per deposit, leading to inaccurate off-chain indexing or broken data consistency in systems relying on event logs (e.g. analytics dashboards or custom dApps).

Importantly, this bug does not affect internal contract accounting or on-chain correctness. It only impacts off-chain consumers of event logs.

Example scenario that shows the bug:

1. A user, Alice, has 3 deposits of 33 tokens each.
2. Alice calls `withdraw(90)` – the contract deducts 30 tokens from each deposit.
3. The function emits:

```
emitCancelDeposit(..., ..., depositIdX, 90);
```

For each deposit ID, `90` tokens are reported instead of the actual `30` deducted from that specific deposit.

This creates the false appearance (in off-chain logs) that Alice was refunded 90 tokens per deposit, totalling 270 tokens.

If off-chain systems rely on these logs for state sync or accounting (e.g., dashboards, reward calculation scripts), they may assume incorrect values and produce invalid results.

Location of Affected Code:

File: contracts/vault/Vault.sol#L563

```
/// @dev Only allowed before the investment period ends. Early withdrawal
    fee may apply.
/// @param amount The amount of deposited tokens to withdraw.
function withdraw(uint256 amount) external whenNotPaused {
    // code

    for (uint256 i = count; i > 0; i--) {
        uint256 id = userIds[i - 1];
        DepositInfo storage deposit = deposits[id];

        if (deposit.amount == 0 || deposit.processed || deposit.
           reimboursed)
            continue;

        uint256 depositAmount = deposit.amount;
```

```
        uint256 amountToDeduct = remainder > depositAmount
            ? depositAmount
            : remainder;

        deposit.amount = depositAmount - amountToDeduct;
        remainder -= amountToDeduct;

        if (deposit.amount == 0) {
            deposit.reimboursed = true;
            // find process index based on the deposit id
            uint256 processId = _findIndex(depositToProcess, id);
            // remove the deposit process id
            _removeAtIndex(depositToProcess, processId);
            // remove the fully processed deposit from user's array
            removeUserDeposit(user, id);
        }

        factory.emitCancelDeposit(msg.sender, deposit.sender, id, amount)
           ; //@note amount is the entire user withdrawal request, not
           the per-deposit deduction (amountToDeduct).

        if (remainder == 0) break;
    }

    // code
}
```

Recommendation

Update the emitted value in the loop to reflect the actual amount deducted from each deposit:

```
- factory.emitCancelDeposit(msg.sender, deposit.sender, id, amount);
+ factory.emitCancelDeposit(msg.sender, deposit.sender, id,
   amountToDeduct);
```

Team Response

Fixed.

[L-02] Vault ID Used Before Emitting Event - Off By One Risk

Severity

Low Risk

Description

In the `deploy()` function of the `VaultFactory` contract, the vault ID is incremented after being used to store the new vault address in the `vaults` mapping.

However, the `VaultDeployed` event is emitted using the incremented value (`vaultId`), which leads to a mismatch between:

- The `vaultId` value **emitted** in the event, and
- The key in the `vaults` mapping under which the new vault address is stored.

This introduces off-chain data integrity issues for indexers, dashboards or UIs relying on `vaultId` – vault address mapping via the event logs. Nothing major.

Location of Affected Code

File: contracts/vault/VaultFactory.sol#L151

```
function deploy(
    string calldata name,
    string calldata symbol,
    address token,
    uint256 minDepositByUser,
    uint256 minTotalDeposit,
    uint256 maxDepositByUser,
    uint256 maxTotalDeposit,
    uint256 investmentEnd
) external onlyRole(OPERATOR_ROLE) returns (address) {
    // code

    uint256 _vaultId = vaultId;

    vaults[_vaultId] = address(newVault);
    isVault[address(newVault)] = true;
    vaultId++;

    emit VaultDeployed(msg.sender, address(newVault), vaultId); //@note
       emits incremented ID

    return address(newVault);
}
```

Recommendation

Emit the correct vault ID from the event, using the unincremented `_vaultId` value:

```
emit VaultDeployed(msg.sender, address(newVault), _vaultId);  // correct
   ID
```

Team Response

Fixed.

[L-03] Operator Has Full Control Over Mint vs. Reimburse – Relying Entirely on Ratio Bounds

Severity

Low Risk

Description

The `processDeposit()` function allows the operator to manually input both `amountMinted` and `amountReimbursed` for a deposit. While ratio bounds (`minRatio` and `maxRatio`) are enforced, the contract does not check that the sum of `amountMinted + amountReimbursed` matches the original deposit amount.

This opens the door for value mismatches where:

- Some deposit value remains unaccounted (ghost value).
- The user receives fewer tokens than expected.

Location of Affected Code:

File: contracts/vault/Vault.sol#L411

```
function processDeposit(
    uint256 processIndex,
    uint256 amountMinted,
    uint256 amountReimbursed
) external onlyRole(factory.OPERATOR_ROLE()) whenNotPaused {
    if (block.timestamp <= investmentEnd)
        revert InvestmentPeriodNotFinish();
    // If we process during a batch, the calculations would be incorrect
    if (inBatchProcess) revert BatchNotFinished();

    uint256 indexDeposit = depositToProcess[processIndex];
    DepositInfo storage deposit = deposits[indexDeposit];
    uint256 amountDeposited = deposit.amount;

    // ratio calculation prevents errors in share distribution
    uint256 currentRatio = (amountMinted * BASE_RATIO) / amountDeposited;

    if (currentRatio < minRatio) revert UnderflowRatio();
    if (currentRatio > maxRatio) revert OverflowRatio();

    _mint(address(this), amountMinted);
    _processDeposit(indexDeposit, amountMinted, amountReimbursed);
    _removeAtIndex(depositToProcess, processIndex);
}
```

Recommendation

Enforce a value-preserving invariant:

```
require(amountMinted + amountReimbursed == deposit.amount, "Value
   mismatch");
```

Team Response

Acknowledged.

[I-01] Missing Validation on `investmentEnd()` in `VaultFactory` Allows Dead-On-Arrival Vaults

Severity

Informational Risk

Description

The `deploy()` function in `VaultFactory` accepts `investmentEnd` as a constructor argument:

```
function deploy(
    // code
    uint256 investmentEnd
) external onlyRole(OPERATOR_ROLE) returns (address)
```

However, there is no check that `investmentEnd > block.timestamp`.

This allows possible misconfiguration and creation of vaults where **the investment period has already ended**, making them:

- Unusable on deployment
- Unavailable for deposits or withdrawals
- Requiring manual remediation

Location of Affected Code

File: contracts/vault/VaultFactory.sol#L151

```
function deploy(
    string calldata name,
    string calldata symbol,
    address token,
    uint256 minDepositByUser,
    uint256 minTotalDeposit,
    uint256 maxDepositByUser,
    uint256 maxTotalDeposit,
    uint256 investmentEnd
) external onlyRole(OPERATOR_ROLE) returns (address) {
    if (!allowedTokens[token]) revert InvalidConfiguration();

    // create data to call initialize method
    bytes memory data = abi.encodeCall(
        IVault.initialize,
        (
            name,
            symbol,
            whitelist,
            token,
            minDepositByUser,
            minTotalDeposit,
            maxDepositByUser,
            maxTotalDeposit,
            investmentEnd
        )
    );
    // use beacon proxy and not clone to deploy a vault
    BeaconProxy newVault = new BeaconProxy(address(vaultBeaconProxy),
       data);

    uint256 _vaultId = vaultId;

    vaults[_vaultId] = address(newVault);
    isVault[address(newVault)] = true;
    vaultId++;

    emit VaultDeployed(msg.sender, address(newVault), vaultId);

    return address(newVault);
}
```

Recommendation

Add a simple validation check to ensure the investment period is in the future:

```
if (investmentEnd <= block.timestamp) revert InvalidConfiguration();
```

Team Response

Fixed.

shieldify

Thank you!