hexens × MANTLE

AUG.24

SECURITY REVIEW REPORT FOR MANTLE

CONTENTS

- It's not possible to use BoringVault.manage() with any value because the contract will revert on receive()
- Missing array parameters length check

ABOUT HEXENS

Hexens is a cybersecurity company that strives to elevate the standards of security in Web 3.0, create a safer environment for users, and ensure mass Web 3.0 adoption.

Hexens has multiple top-notch auditing teams specialized in different fields of information security, showing extreme performance in the most challenging and technically complex tasks, including but not limited to: Infrastructure Audits, Zero Knowledge Proofs / Novel Cryptography, DeFi and NFTs. Hexens not only uses widely known methodologies and flows, but focuses on discovering and introducing new ones on a day-to-day basis.

In 2022, our team announced the closure of a $4.2 million seed round led by IOSG Ventures, the leading Web 3.0 venture capital. Other investors include Delta Blockchain Fund, Chapter One, Hash Capital, ImToken Ventures, Tenzor Capital, and angels from Polygon and other blockchain projects.

Since Hexens was founded in 2021, it has had an impressive track record and recognition in the industry: Mudit Gupta - CISO of Polygon Technology - the biggest EVM Ecosystem, joined the company advisory board after completing just a single cooperation iteration. Polygon Technology, 1inch, Lido, Hats Finance, Quickswap, Layerswap, 4K, RociFi, as well as dozens of DeFi protocols and bridges, have already become our customers and taken proactive measures towards protecting their assets.

SCOPE

The analyzed resources are located on:

https://github.com/Se7en-Seas/cMETH-boring-vault/tree/v0.1.2

The issues described in this report were fixed in the following commit:

https://github.com/Se7en-Seas/cMETH-boring-vault.git
branch: origin/audit-fix

AUDITING DETAILS

STARTED
19.08.2024

DELIVERED
02.09.2024

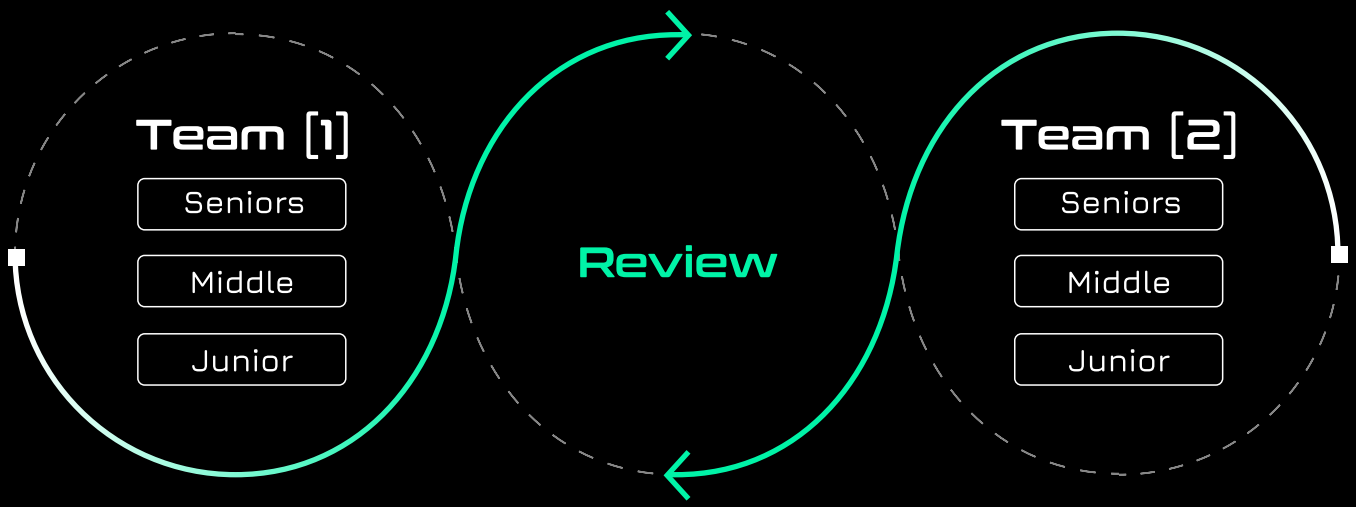
Review Led by

TRUNG DINH

Senior Security Researcher | Hexens

HEXENS METHODOLOGY

Hexens methodology involves 2 teams, including multiple auditors of different seniority, with at least 5 security engineers. This unique cross-checking mechanism helps us provide the best quality in the market.



SEVERITY STRUCTURE

The vulnerability severity is calculated based on two components

- Impact of the vulnerability
- Probability of the vulnerability

Impact	Probability			
	rare	unlikely	likely	very likely
Low/Info	Low/Info	Low/Info	Medium	Medium
Medium	Low/Info	Medium	Medium	High
High	Medium	Medium	High	Critical
Critical	Medium	High	Critical	Critical

SEVERITY CHARACTERISTICS

Smart contract vulnerabilities can range in severity and impact, and it's important to understand their level of severity in order to prioritize their resolution. Here are the different types of severity levels of smart contract vulnerabilities:

Critical

Vulnerabilities with this level of severity can result in significant financial losses or reputational damage. They often allow an attacker to gain complete control of a contract, directly steal or freeze funds from the contract or users, or permanently block the functionality of a protocol. Examples include infinite mints and governance manipulation.

High

Vulnerabilities with this level of severity can result in some financial losses or reputational damage. They often allow an attacker to directly steal yield from the contract or users, or temporarily freeze funds. Examples include inadequate access control integer overflow/underflow, or logic bugs.

Medium

Vulnerabilities with this level of severity can result in some damage to the protocol or users, without profit for the attacker. They often allow an attacker to exploit a contract to cause harm, but the impact may be limited, such as temporarily blocking the functionality of the protocol. Examples include uninitialized storage pointers and failure to check external calls.

Low

Vulnerabilities with this level of severity may not result in financial losses or significant harm. They may, however, impact the usability or reliability of a contract. Examples include slippage and front-running, or minor logic bugs.

Informational

Vulnerabilities with this level of severity are regarding gas optimizations and code style. They often involve issues with documentation, incorrect usage of EIP standards, best practices for saving gas, or the overall design of a contract. Examples include not conforming to ERC20, or disagreement between documentation and code.

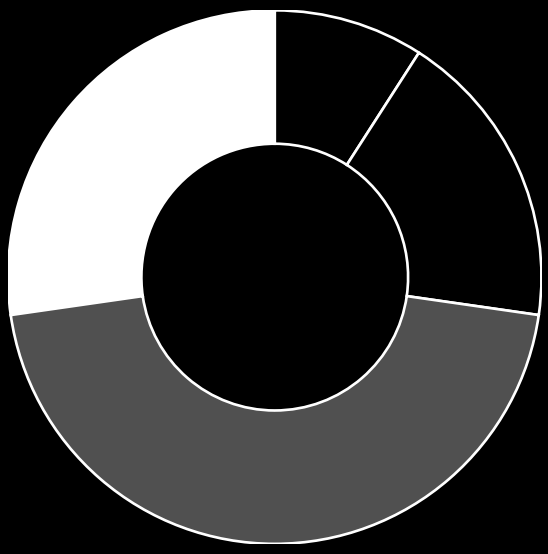
ISSUE SYMBOLIC CODES

Every issue being identified and validated has its unique symbolic code assigned to the issue at the security research stage. Cause of the vulnerability reporting flow design, some of the rejected issues could be missing.

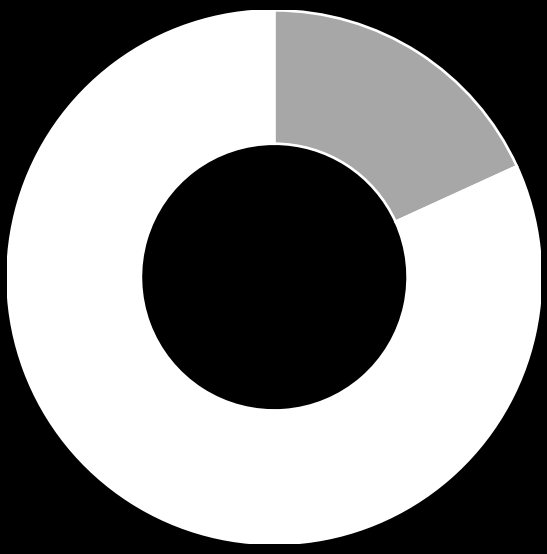
FINDINGS SUMMARY

Severity	Number of Findings
Critical	0
High	1
Medium	2
Low	5

Total: 11



- High
- Medium
- Low
- Informational



- Fixed
- Acknowledged

WEAKNESSES

This section contains the list of discovered weaknesses.

MANTLE-1

USER CAN ADD AND REMOVE USERS TO SANCTIONSLIST AND BLOCK THE UPDATE FUNCTION

SEVERITY: High

PATH:

src/lists/SanctionsList.sol#L20-L32

REMEDIATION:

See description.

STATUS: Fixed

DESCRIPTION:

In **SanctionList.sol**, the functions from lines 20 to 32 lack access control, unlike the similar contract **Blocklist.sol**, which uses the **onlyOwner** modifier for adding and removing entries.

Because of this absence of access control, an attacker could potentially add addresses to the sanction list, blocking transfers for those addresses.

```
function addToSanctionsList(address[] memory newSanctions) public {
    for (uint256 i = 0; i < newSanctions.length; i++) {
        sanctionedAddresses[newSanctions[i]] = true;
    }
    emit SanctionedAddressesAdded(newSanctions);
}


function removeFromSanctionsList(address[] memory removeSanctions) public {
    for (uint256 i = 0; i < removeSanctions.length; i++) {
        sanctionedAddresses[removeSanctions[i]] = false;
    }
    emit SanctionedAddressesRemoved(removeSanctions);
}
```

Add the access control modifier to the functions as follows:

```
- function addToSanctionsList(address[] memory newSanctions) public  {
+ function addToSanctionsList(address[] memory newSanctions) public
onlyOwner {
```

and

```
- function removeFromSanctionsList(address[] memory removeSanctions) public
{
+ function removeFromSanctionsList(address[] memory removeSanctions) public
onlyOwner {
```

Here is the test to verify the issue:

```
function testSetSanctionsListSuccess() public {
        // create user
        address attacker = makeAddr("attacker");
        vm.startPrank(attacker);

        // Get SanctionList contract used by l1cmETH
        IISanctionsList myContract = IISanctionsList(sanctionList);

        // add address to a list
        MemberAddresses = [address(1)];

        // add To SanctionsList
        myContract.addToSanctionsList(MemberAddresses);
        myContract.addToBlockList(MemberAddresses);

        //check if its sanctioned
        assertTrue(
            l1cmETH.isSanctionedInL1cmEth(MemberAddresses[0]),
            "Adddress not found in sanctions list"
        );

        // remove any sanction aswell
        myContract.removeFromSanctionsList(MemberAddresses);

        //check if its sanctioned
        assertFalse(
            l1cmETH.isSanctionedInL1cmEth(MemberAddresses[0]),
            "Adddress found in sanctions list"
        );


    }
```

Added for convenience in L1cmETH.sol

```
function isSanctionedInL1cmEth(address addr_) public returns(bool) {
        return _isSanctioned(addr_);
    }
```

MANTLE-12

OWNER CAN CLAIM MORE REWARDS THAN THEY'RE SUPPOSED TO

SEVERITY: Medium

PATH:

src/base/Roles/AccountantWithRateProviders.sol#L221-L232

src/base/Roles/AccountantWithRateProviders.sol#L234-L245

REMEDIATION:

Add _calculateFeesOwed to updateManagementFee &. updatePerformanceFee.

STATUS: Acknowledged, see commentary

DESCRIPTION:

The functions:

- **resetHighwaterMark** => calls **_calculateFeesOwed** before updating
- **updateExchangeRate** => calls **_calculateFeesOwed** before updating
- **updateManagementFee** => does not accrue fees
- **updatePerformanceFee** => does not accrue fees

Unlike the functions **updateExchangeRate()** or **resetHighwaterMark()**, the functions **updateManagementFee()** and **updatePerformanceFee()** don't accrue the fee for the owner before changing the fee configuration. This can lead to an issue when the subsequent **calculateFeeOwed()** function is triggered, as it will use the new fee configuration to calculate the fee for the owner, potentially resulting in a larger fee accrued for the owner than expected if the updated fee is bigger than the old one.

```
    function updateManagementFee(uint16 managementFee) external requiresAuth
{
        if (managementFee > 0.2e4) {
            revert AccountantWithRateProviders__ManagementFeeTooLarge();
        }
        uint16 oldFee = accountantState.managementFee;
        accountantState.managementFee = managementFee;
        emit ManagementFeeUpdated(oldFee, managementFee);
    }
```

```
function updatePerformanceFee(uint16 performanceFee) external requiresAuth {
    if (performanceFee > 0.5e4) {
        revert AccountantWithRateProviders__PerformanceFeeTooLarge();
    }
    uint16 oldFee = accountantState.performanceFee;
    accountantState.performanceFee = performanceFee;
    emit PerformanceFeeUpdated(oldFee, performanceFee);
}
```

_calculateFeesOwed is not called, so it doesn't use the old values to calculate the fees.

```
    function updateManagementFee(uint16 managementFee) external requiresAuth
{
        if (managementFee > 0.2e4) {
            revert AccountantWithRateProviders__ManagementFeeTooLarge();
        }
        uint16 oldFee = accountantState.managementFee;
        accountantState.managementFee = managementFee;
        emit ManagementFeeUpdated(oldFee, managementFee);
    }
```

Running the test:
forge test -vv --match-test testClaimMoreFees
Add the following test function to **AccountantWithRateProvidersTest.t.sol**:

```
function testClaimMoreFees() external {
        deal(address(WEETH), address(boringVault), 1e18);


        // Test with unchanged Management fee:
        accountant.updateManagementFee(0.01e4);

        skip(1 days / 24);
        uint96 new_exchange_rate = uint96(1.0005e18);
        accountant.updateExchangeRate(new_exchange_rate);

        skip(1 days / 24);
        new_exchange_rate = uint96(1.0015e18);
        accountant.updateExchangeRate(new_exchange_rate);


        vm.startPrank(address(boringVault));
        WEETH.safeApprove(address(accountant), 1e18);
        uint256 initialBalance = WEETH.balanceOf(address(payout_address));
        accountant.claimFees(WEETH);
        vm.stopPrank();

        uint256 afterBalance = WEETH.balanceOf(address(payout_address));
        uint256 normalFeesReceived = (afterBalance - initialBalance);
        console.log("Fees received", normalFeesReceived);


        // Test with changed Management fee:
        skip(1 days / 24);
        new_exchange_rate = uint96(1.0005e18);
        accountant.updateExchangeRate(new_exchange_rate);

         skip(1 days / 24);
        new_exchange_rate = uint96(1.0015e18);
        accountant.updateManagementFee(0.2e4); // vurnability here
        accountant.updateExchangeRate(new_exchange_rate);
```

```
        vm.startPrank(address(boringVault));
        WEETH.safeApprove(address(accountant), 1e18);
        initialBalance = WEETH.balanceOf(address(payout_address));
        accountant.claimFees(WEETH);
        vm.stopPrank();

        afterBalance = WEETH.balanceOf(address(payout_address));
        uint256 moreFeesReceived = (afterBalance - initialBalance);

        console.log("Fees received", moreFeesReceived);
        console.log("Received", moreFeesReceived / normalFeesReceived,
"times more fees.");
    }
```

Output:

```
logs:
Fees received 1100396267186392
Fees received 23108321610914267
Received 21 times more fees.
```

Commentary from the client:

“ - This issue will have no impact on cmETH as the performance and management fees will be zero. Also trust assumption wise the entity capable of changing the fees is the same entity that can change the strategists merkle roots, and we already have the trust assumption that this entity will not setup malicious merkle trees(which is far more damaging than abusing fee logic), so this issue also does not expand trust assumptions.”

DOS IN L2MESSAGINGSTATUS.SETEXCHANGERATEFOR() DUE TO AN INCORRECT ENCODED FUNCTION SELECTOR

SEVERITY: Medium

PATH:

src/L2MessagingStatus.sol#L151-L152

REMEDIATION:

See description.

STATUS: Fixed

DESCRIPTION:

The **L2MessagingStatus.setExchangeRateFor()** function is designed to set a new exchange rate for the contract on another chain. However, the function mistakenly encodes the selector for a non-existent function, **setIsExchangeRate()**, instead of the intended **setExchangeRate()** function. As a result, the message sent by **setExchangeRateFor()** cannot be processed on the destination chain, leading to a DoS vulnerability.

```
function setExchangeRateFor(uint32 eid, uint256 rate) external payable
onlyRole(MANAGER_ROLE) {
    bytes memory lzPayload = abi.encode(block.timestamp,
bytes4(keccak256("setIsExchangeRate(uint256)")), rate);
```

Consider modifying the function L2MessagingStatus.setExchangeRateFor() to:

```
function setExchangeRateFor(uint32 eid, uint256 rate) external payable
onlyRole(MANAGER_ROLE) {
-    bytes memory lzPayload = abi.encode(block.timestamp,
bytes4(keccak256("setIsExchangeRate(uint256)")), rate);
+    bytes memory lzPayload = abi.encode(block.timestamp,
bytes4(keccak256("setExchangeRate(uint256)")), rate);
```

INCOMPLETE EVENT EMISSION

SEVERITY: Low

PATH:

src/base/Roles/DelayedWithdraw.sol#L139-L144

src/base/Roles/DelayedWithdraw.sol#L257-L262

REMEDIATION:

Consider adding the completionWindow in the SetupWithdrawalsInAsset event to ensure all relevant information is captured.

STATUS: Acknowledged

DESCRIPTION:

The **SetupWithdrawalsInAsset** event is emitted every time the **OWNER_ROLE** sets up new withdrawal settings, including **withdrawDelay**, **completionWindow**, **withdrawFee**, and **maxLoss** for a specific asset. However, the **SetupWithdrawalsInAsset** event does not include the **completionWindow** parameter, which could lead to inconsistencies or difficulties in tracking withdrawal settings accurately.

```
event SetupWithdrawalsInAsset(
        address indexed asset,
        uint64 withdrawDelay,
        uint16 withdrawFee,
        uint16 maxLoss
    );
```

```
function setupWithdrawAsset(
        ERC20 asset,
        uint32 withdrawDelay,
        uint32 completionWindow,
        uint16 withdrawFee,
        uint16 maxLoss
    ) external requiresAuth {
      ...

        emit SetupWithdrawalsInAsset(
            address(asset),
            withdrawDelay,
            withdrawFee,
            maxLoss
        );
      }
```

INCORRECT MAXIMUM LOSS CALCULATION IN WITHDRAWAL COMPLETION

SEVERITY: Low

PATH:

src/base/Roles/DelayedWithdraw.sol#L542-L604

REMEDIATION:

To have the maximum loss be a percentage of the actual value, the calculation should be:

```
if (maxRate.mulDivDown(1e4 - maxLoss, 1e4) > minRate)
    revert DelayedWithdraw__MaxLossExceeded();
```

STATUS: Acknowledged, see commentary

DESCRIPTION:

The withdrawal mechanism in **DelayedWithdraw** allows the user to specify the maximum loss for a withdrawal as it takes the minimum between the exchange rate at creation and completion.

It should be that the maximum loss is a percentage of the actual value that the user is willing to give up.

However this is not the case, in the current calculation the specified **maxLoss** is used in **minRate * (1.0 + maxLoss) / 1.0** and so it is actually smaller than expected. For example the **MAX_LOSS = 0.5e4** constant is not actually 50%, but rather 33% of the **maxRate**.

```
// Make sure minRate * maxLoss is greater than or equal to maxRate.
if (minRate.mulDivDown(1e4 + maxLoss, 1e4) < maxRate)
    revert DelayedWithdraw__MaxLossExceeded();
```

For example, if the user sets the **maxLoss** to 10% (**0.1e4**), then the completion would fail if with **maxRate = 1.0, minRate = 0.9** since **0.9 * (1.1 / 1.0) = 0.99 < 1.0**.

A more extreme example with a **maxLoss** of 50% (**0.5e4**) and with **maxRate = 1.0, minRate = 0.5**, then **0.5 * (1.5 / 1.0) = 0.75 < 1.0**.

Commentary from the client:

" - Acknowledged, while the current implementation does not really represent the maxLoss as a percentage of the actual value, the important point is that the current implementation does protect users from excessive losses when withdrawing from the boring vault. Also cmETH's share price will be constant at 1, so a maxLoss of zero can be used."

INCORRECT ROLE ASSIGNMENT FOR SETISTRANSFERPAUSEDFOR() FUNCTION

SEVERITY: Low

PATH:

src/L2MessagingStatus.sol#L132

src/L1MessagingStatus.sol#L112

REMEDIATION:

We recommend change the access modifier from MANAGER_ROLE to onlyPauserUnpauserRole.

STATUS: Acknowledged

DESCRIPTION:

Protocol implements pause logic with **PAUSER** and **UNPAUSER** roles and use it for pausing mechanism. The pause logic is a critical part of the protocol's security, allowing authorized entities to halt operations in emergency situations. However, the **setIsTransferPausedFor()** is currently restricted to users with the **MANAGER_ROLE**, rather than the **PAUSER** or **UNPAUSER** roles, which are specifically designated for controlling the pause functionality.

This could lead to a scenario where in case of an emergency **PAUSER** doesn't have the ability to pause that function.

```
function setIsTransferPausedFor(uint32 eid, bool isPaused) external payable
onlyRole(MANAGER_ROLE) {
        ...
    }
```

MANTLE-13

LACK OF UPPER LIMIT CHECK FOR CRUCIAL PARAMETERS ALLOWS FREEZING OF ASSETS

SEVERITY: Low

PATH:

src/base/Roles/DelayedWithdraw.sol#L269-L280

src/base/Roles/DelayedWithdraw.sol#L236-L263

REMEDIATION:

Sensible upper limits should be checked before setting the value.

STATUS: Acknowledged, see commentary

DESCRIPTION:

In the DelayedWithdraw contract, **withdrawDelay** represents the delay in seconds before a requested withdrawal can be completed. While this variable is modifiable, the corresponding setter function lacks an upper-bound check for the values it accepts. Consequently, **withdrawDelay** can be set to any arbitrary **uint32** value, which could be excessively high, potentially preventing the unlocking of user assets.

```
function changeWithdrawDelay(
    ERC20 asset,
    uint32 withdrawDelay
) external requiresAuth {
    WithdrawAsset storage withdrawAsset = withdrawAssets[asset];
    if (!withdrawAsset.allowWithdraws)
        revert DelayedWithdraw__WithdrawsNotAllowed();

    withdrawAsset.withdrawDelay = withdrawDelay;

    emit WithdrawDelayUpdated(address(asset), withdrawDelay);
}
```

```
function setupWithdrawAsset(
    ERC20 asset,
    uint32 withdrawDelay,
    uint32 completionWindow,
    uint16 withdrawFee,
    uint16 maxLoss
) external requiresAuth {
    WithdrawAsset storage withdrawAsset = withdrawAssets[asset];

    if (withdrawFee > MAX_WITHDRAW_FEE)
        revert DelayedWithdraw__WithdrawFeeTooHigh();
    if (maxLoss > MAX_LOSS) revert DelayedWithdraw__MaxLossTooLarge();

    if (withdrawAsset.allowWithdraws)
        revert DelayedWithdraw__AlreadySetup();
    withdrawAsset.allowWithdraws = true;
    withdrawAsset.withdrawDelay = withdrawDelay;
    withdrawAsset.completionWindow = completionWindow;
    withdrawAsset.withdrawFee = withdrawFee;
    withdrawAsset.maxLoss = maxLoss;
    ...
}
```

Commentary from the client:

“ - Acknowledged, this issue really does not expand trust assumptions, and even if these checks were added, if the strategist does not move the assets into the DelayedWithdraw contract then withdraws are frozen, so I don't feel like these checks add a ton of guarantees, but they do add extra restrictions which make the logic more complex/inflexible.”

LACK OF ZERO ADDRESS CHECK FOR PAYOUTADDRESS IN ACCOUNTANTWITHRATEPROVIDERS CONTRACT

SEVERITY: Low

PATH:

src/base/Roles/AccountantWithRateProviders.sol#L251-L255

REMEDIATION:

Zero address check should be added to this function.

STATUS: Acknowledged, see commentary

DESCRIPTION:

In the **AccountantWithRateProviders** contract, the **updatePayoutAddress()** function doesn't have any checks to ensure that the new payoutAddress is not zero. This may create a risk where the admin/owner mistakenly sets the **payoutAddress** to zero, causing a loss of fees for the protocol.

```
function updatePayoutAddress(address payoutAddress) external requiresAuth {
      address oldPayout = accountantState.payoutAddress;
      accountantState.payoutAddress = payoutAddress;
      emit PayoutAddressUpdated(oldPayout, payoutAddress);
  }
```

Commentary from the client:

“ - Acknowledged, cmETH will not be taking fees using the AccountantWithRateProvider, so there would never be calls to claimFees, and thus no issue if the payout address was zero.”

CONSOLE.LOG STATEMENTS IN ACCOUNTANTWITHRATEPROVIDERS WERE NOT REMOVED

SEVERITY: Informational

PATH:

src/base/Roles/AccountantWithRateProviders.sol#L12
src/base/Roles/AccountantWithRateProviders.sol#L461

REMEDIATION:

The redundant logging code should be removed.

STATUS: Acknowledged, see commentary

DESCRIPTION:

```
console.log("minimum assets: ", minimumAssets);
```

Commentary from the client:

“ - Acknowledged, we will fix this in the future when we have a larger change to make to the Accountant, but in its current state, the only downsides are its a bit ugly, and some extra gas will be wasted.”

IT'S NOT POSSIBLE TO USE BORINGVAULT.MANAGE() WITH ANY VALUE BECAUSE THE CONTRACT WILL REVERT ON RECEIVE()

SEVERITY: Informational

PATH:

src/lib/TransparentUpgradeableProxy.sol#L101-L106
src/base/BoringVault.sol#L111
src/base/BoringVault.sol#L39-L65

REMEDIATION:

Refactor the proxy to not expose any external functions to users. Hide admin functions under the fallback.

STATUS: Acknowledged, see commentary

DESCRIPTION:

Due to the implementation of **TransparentUpgradeableProxy**, it's not possible to transfer funds to **BoringVault** (it's planned to be deployed as an implementation of the proxy).

As you can see in the code snippet below, **receive()** of the proxy will always revert. Although **BoringVault** also has a **receive()**, it'll never be called.

The two **manage()** functions in **BoringVault** don't have a **payable** keyword, so they can't accept value by themselves.

The only way to add funds to the contract remains the deprecated **selfdestruct**, the use of which is not encouraged and is a subject to change.

```
    /**
     * @dev just nee to implement receive to forbid transfer value
     */
    receive() external payable virtual {
        revert();
    }
```

```
    receive() external payable {}
```

```
    /**
     * @notice Allows manager to make an arbitrary function call from this contract.
     * @dev Callable by MANAGER_ROLE.
     */
    function manage(address target, bytes calldata data, uint256 value)
        external
        requiresAuth
        returns (bytes memory result)
    {
        result = target.functionCallWithValue(data, value);
    }

    /**
     * @notice Allows manager to make arbitrary function calls from this contract.
     * @dev Callable by MANAGER_ROLE.
     */
    function manage(address[] calldata targets, bytes[] calldata data, uint256[] calldata values)
        external
        requiresAuth
        returns (bytes[] memory results)
    {
        uint256 targetsLength = targets.length;
        results = new bytes[](targetsLength);
        for (uint256 i; i < targetsLength; ++i) {
            results[i] = targets[i].functionCallWithValue(data[i], values[i]);
        }
    }
```

Commentary from the client:

“ - Acknowledged, this is an important point to bring up. There are currently no plans for cmETH to handle native eth, but if in the future it does need to, I have some out of scope code that would allow us to support it.”

MISSING ARRAY PARAMETERS LENGTH CHECK

SEVERITY: Informational

PATH:

src/base/BoringVault.sol#L55

REMEDIATION:

Consider adding a check to ensure that targets.length, data.length, and values.length are equal.

STATUS: Acknowledged, see commentary

DESCRIPTION:

There are 3 arrays as **manage()** function parameters that require the same length but are missing these length checks:

```
function manage(address[] calldata targets, bytes[] calldata data, uint256[]
calldata values)// @note need to  require check for lenght equality
        external
        requiresAuth
        returns (bytes[] memory results)
    {
        uint256 targetsLength = targets.length;
        results = new bytes[](targetsLength);
        for (uint256 i; i < targetsLength; ++i) {
                    results[i]  =  targets[i].functionCallWithValue(data[i],
values[i]);
        }
    }
```

Commentary from the client:

“ - Acknowledged, these array length checks are actually done in the ManagerWithMerkleVerification.”


hexens
MANTLE