Spectra Security Review

Version 2.0

28.06.2024

Conducted by:

MaslarovK, Security Researcher

radev-eth, Security Researcher

Table of Contents

1 About MaslarovK

MaslarovK is an independent security researcher from Bulgaria. He has secured various protocols through private audits and public contests - Secured ~$5M in TVL.

2 About radev.eth

radev_eth is an independent security researcher from Bulgaria. He is an experinced blockchain developer, also proficient in security with various top 10 finished in public contests.

3 Disclaimer

Audits are a time, resource, and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can show the presence of vulnerabilities **but not their absence**.

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** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that's not so critical.

4.2 Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

4.3 Actions required by severity level

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

5 Executive summary

Overview

Project Name	Spectra
Repository	https://github.com/perspectivefi/gmx-market-token-4626-wrapper
Commit hash	85fda51624c6f1a793d4806998783ad3e968f05c
Resolution	3c550d4d73b61a55c505268587e1f53d1646233b
Documentation	N/A
Methods	Manual review & testing

Scope

src/gmx-v2/SpectraWrappedGM.sol
src/utils/Spectra4626Wrapper.sol

Issues Found

Critical risk	0
High risk	0
Medium risk	2
Low risk	4
Informational	0

6 Findings

6.1 Medium risk

6.1.1 Chainlink oracle will return the wrong price if the aggregator hits `minAnswer`

Severity: *Medium risk*

Description: The `SpectraWrappedGM.sol#_getTokenPrice()` function retrieves the latest price (`latestAnswer`) from the Chainlink price feed (ChainlinkFeedRegistry) for the specified token.

```
    function _getTokenPrice(
        address token
    ) internal view returns (IReader.PriceProps memory price) {
        (, int256 latestAnswer, , , ) = IPriceFeed(
            IDataStore(dataStore).getAddress(_priceFeedKey(token))
        ).latestRoundData();

        // ... code ...
    }
```

However, Chainlink aggregators have a built in circuit breaker if the price of an asset goes outside of a predetermined price band. The result is that if an asset experiences a huge drop in value (i.e. LUNA crash) the price of the oracle will continue to return the minPrice instead of the actual price of the asset. This would allow user to continue borrowing with the asset but at the wrong price.

6.1.1.1 Real-World Example

During the LUNA crash, the Chainlink oracle continued to return the `minAnswer` instead of the actual market price, allowing users to borrow against LUNA at an inflated price, leading to significant losses for the protocol. The detailed incident report can be found here.

Recommendation: To prevent this issue, the `SpectraWrappedGM.sol` contract should include additional checks to validate the returned price from the Chainlink oracle. Specifically, it should compare the `latestAnswer` against predefined `minimum and maximum price bounds` and revert if the answer falls outside these bounds

Here is an example implementation of the `SpectraWrappedGM.sol#_getTokenPrice()` function with the added checks:

```
    function _getTokenPrice(
        address token
    ) internal view returns (IReader.PriceProps memory price) {
        (, int256 latestAnswer, , , ) = IPriceFeed(
            IDataStore(dataStore).getAddress(_priceFeedKey(token))
        ).latestRoundData();

+       // Define the min and max price bounds
+       int256 minPrice = ...; // Set appropriate minPrice
+       int256 maxPrice = ...; // Set appropriate maxPrice

+       // Revert if the answer is outside of the bounds
+       require(latestAnswer >= minPrice && latestAnswer <= maxPrice, "Price out of
    bounds");

        uint256 multipler = IOracle(oracle).getPriceFeedMultiplier(
            dataStore,
            token
        );
        uint256 adjustedPrice = uint256(latestAnswer).mulDiv(
            multipler,
            FLOAT_PRECISION
        );
        return IReader.PriceProps({min: adjustedPrice, max: adjustedPrice});
    }
```

Resolution: Aknowledged

6.2 Medium risk

6.2.1 Insufficient oracle validation

Severity: *Medium risk*

Description: In the `SpectraWrappedGM.sol` contract, there is insufficient validation of the price data fetched from the Chainlink oracle. Specifically, there is no check on the freshness of the timestamp associated with the prices. If the Chainlink Off-Chain Reporting (OCR) mechanism fails to push an update in time, the contract may use outdated prices, which can lead to incorrect asset valuations. This is because, most prices are provided by an off-chain oracle archive via signed prices, but a Chainlink oracle is still used for index prices. These prices are insufficiently validated.

```
/// @audit latestAnswer missing validation
    function _getTokenPrice(address token) internal view returns (IReader.
    PriceProps memory price) {
        (, int256 latestAnswer, , , uint256 updatedAt, ) = IPriceFeed(
            IDataStore(dataStore).getAddress(_priceFeedKey(token))
        ).latestRoundData();

        uint256 multipler = IOracle(oracle).getPriceFeedMultiplier(dataStore,
    token);
        uint256 adjustedPrice = uint256(latestAnswer).mulDiv(multipler,
    FLOAT_PRECISION);
        return IReader.PriceProps({min: adjustedPrice, max: adjustedPrice});
    }
```

In scenarios where the Chainlink OCR fails to update the price feed promptly, the returned `latestAnswer` might be outdated. This can result in the protocol using stale prices, and users/depositors will get wrong values for their positions during depositing and withdrawing.

Recommendation: We recommend to implement a staleness threshold parameter to ensure the price data is recent. Here's an example version of the `SpectraWrappedGM.sol#_getTokenPrice()` function with the added staleness check:

```
+   // Define the staleness threshold/heartbeat period in seconds!


    function _getTokenPrice(
        address token
    ) internal view returns (IReader.PriceProps memory price) {
        (, int256 latestAnswer, , uint256 updatedAt, ) = IPriceFeed(
            IDataStore(dataStore).getAddress(_priceFeedKey(token))
        ).latestRoundData();

+       // Revert if the price data is stale
+       require(block.timestamp - updatedAt <= HEARTBEAT_PERIOD, "Price data is
    stale");

        uint256 multipler = IOracle(oracle).getPriceFeedMultiplier(
            dataStore,
            token
        );
        uint256 adjustedPrice = uint256(latestAnswer).mulDiv(
            multipler,
            FLOAT_PRECISION
        );
        return IReader.PriceProps({min: adjustedPrice, max: adjustedPrice});
    }
```

This ensures that the price data used by the protocol is within an acceptable time range, preventing the use of outdated prices and maintaining the integrity of the asset valuations.

Resolution: Fixed

6.3 Low Risk

6.3.1 Unclaimed rewards may become lost when updating rewards proxy in `Spectra4626Wrapper.sol`

Severity: *Low risk*

Description: In the `Spectra4626Wrapper.sol` contract, the `setRewardsProxy` function allows for updating the address of the rewards proxy.

```
    /// @dev See {ISpectra4626Wrapper-setRewardsProxy}.
    function setRewardsProxy(
        address newRewardsProxy
    ) public virtual restricted {
        // Note: address zero is allowed in order to disable the claim proxy
        _setRewardsProxy(newRewardsProxy);
    }

    /// @dev Updates the rewards proxy. Internal function with no access restriction
        .
    function _setRewardsProxy(address newRewardsProxy) internal virtual {
        Spectra4626WrapperStorage storage $ = _getSpectra4626WrapperStorage();
        address oldRewardsProxy = $._rewardsProxy;
        $._rewardsProxy = newRewardsProxy;
        emit RewardsProxyUpdated(oldRewardsProxy, newRewardsProxy);
    }
```

However, this function does not claim the rewards from the current rewards proxy before setting a new one. The absence of this step can lead to unclaimed rewards being lost when the rewards proxy address is updated.

If the rewards proxy is updated without claiming the pending rewards, any rewards that have accumulated in the current rewards proxy will be lost. This will result in financial loss for the users or the protocol.

Recommendation: The `Spectra4626Wrapper.sol#setRewardsProxy()` function should first call `Spectra4626Wrapper.sol#claimRewards()` (also this function should become public or create internal function for claiming rewards) before setting the new rewards proxy address to ensure that all rewards are claimed from the current proxy.

Resolution: Aknowledged

6.3.2 `pure` function accessing storage

Severity: *Low risk*

Description The `_getSpectra4626WrapperStorage` function uses inline assembly to access a storage slot. Despite this, it is marked as `pure`, which is incorrect as `pure` functions should not read or modify storage. This is currently allowed by the compiler due to a bug, bug it is expected to be disallowed in future Solidity versions.

```
function _getSpectra4626WrapperStorage()
        private
        pure
        returns (Spectra4626WrapperStorage storage $)
    {
        assembly {
            $.slot := Spectra4626WrapperStorageLocation
        }
    }
```

When the bug in the Solidity compiler is fixed, marking this function as `pure` will result in a compilation error. This will break the contract and require code changes to resolve the issue.

Resolution: Aknowledged

6.3.3 Setters Should Prevent Re-setting of the Same Value

Severity: *Low risk*

Description

```
/// @dev See {ISpectra4626Wrapper-setRewardsProxy}.
    function setRewardsProxy(
        address newRewardsProxy
    ) public virtual restricted {
        // Note: address zero is allowed in order to disable the claim proxy
        _setRewardsProxy(newRewardsProxy);
    }

    /// @dev Updates the rewards proxy. Internal function with no access restriction
        .
    function _setRewardsProxy(address newRewardsProxy) internal virtual {
        Spectra4626WrapperStorage storage $ = _getSpectra4626WrapperStorage();
        address oldRewardsProxy = $._rewardsProxy;
        $._rewardsProxy = newRewardsProxy;
        emit RewardsProxyUpdated(oldRewardsProxy, newRewardsProxy);
    }
```

Recommendation We recommend changing the `Spectra4626Wrapper.sol#_setRewardsProxy()` function as follows:

```
    function _setRewardsProxy(address newRewardsProxy) internal virtual {
        Spectra4626WrapperStorage storage $ = _getSpectra4626WrapperStorage();
        address oldRewardsProxy = $._rewardsProxy;
        $._rewardsProxy = newRewardsProxy;
        emit RewardsProxyUpdated(oldRewardsProxy, newRewardsProxy);
+       if (newRewardsProxy != oldRewardsProxy) {
            $._rewardsProxy = newRewardsProxy;
            emit RewardsProxyUpdated(oldRewardsProxy, newRewardsProxy);
+       }
    }
```

Resolution: Aknowledged

6.3.4 Setters Should Prevent Re-setting of the Same Value

Severity: *Low risk*

Description

```
/// @dev See {ISpectra4626Wrapper-setRewardsProxy}.
    function setRewardsProxy(
        address newRewardsProxy
    ) public virtual restricted {
        // Note: address zero is allowed in order to disable the claim proxy
        _setRewardsProxy(newRewardsProxy);
    }

    /// @dev Updates the rewards proxy. Internal function with no access restriction
        .
    function _setRewardsProxy(address newRewardsProxy) internal virtual {
        Spectra4626WrapperStorage storage $ = _getSpectra4626WrapperStorage();
        address oldRewardsProxy = $._rewardsProxy;
        $._rewardsProxy = newRewardsProxy;
        emit RewardsProxyUpdated(oldRewardsProxy, newRewardsProxy);
    }
```

Recommendation We recommend changing the `Spectra4626Wrapper.sol#_setRewardsProxy()` function as follows:

```
    function _setRewardsProxy(address newRewardsProxy) internal virtual {
        Spectra4626WrapperStorage storage $ = _getSpectra4626WrapperStorage();
        address oldRewardsProxy = $._rewardsProxy;
        $._rewardsProxy = newRewardsProxy;
        emit RewardsProxyUpdated(oldRewardsProxy, newRewardsProxy);
+       if (newRewardsProxy != oldRewardsProxy) {
            $._rewardsProxy = newRewardsProxy;
            emit RewardsProxyUpdated(oldRewardsProxy, newRewardsProxy);
+       }
    }
```

Resolution: Aknowledged

6.3.5 Missing Storage Gap in Upgradeable Contract

Severity: *Low risk*

Description

The `Spectra4626Wrapper.sol` contract is an upgradeable contract but does not include a `__gap[50]` storage variable at the end. This variable is recommended by OpenZeppelin to prevent storage layout issues when upgrading contracts.

See this link for a description of this storage variable. While some contracts may not currently be subclassed, adding the variable now protects against forgetting to add it in the future.

Recommendation

Add the `__gap[50]` storage variable to the end of the contract to reserve storage slots for future upgrades.