CD SECURITY

AUDIT REPORT

Euler - HookTargetMarketStatus & DataStreamsVerifier

August 2025

Introduction

A time-boxed security review of the **Euler** protocol was done by **CD Security**, with a focus on the security aspects of the application's implementation.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource, and expertise-bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs, and on-chain monitoring are strongly recommended.

About **Euler**

This audit was specifically focused on two contracts - HookTargetMarketStatus & DataStreamsVerifier.

DataStreamsVerifier.sol

A base contract that provides Chainlink Data Streams verification functionality with the following features:

- Authorization Management: Restricts update functions to pre-authorized callers
- Version Validation: Ensures reports match expected version
- Fee Management: Automatically sets up LINK token approvals for reward managers
- Token Recovery: Allows contract owners to recover any ERC20 tokens sent to the contract

HookTargetMarketStatus.sol

A specialized contract that extends DataStreamsVerifier to manage market status based on Chainlink V8 reports. It's meant to be used as a hook target contract for select euler-vault-kit vaults:

- Market Status Management: Tracks and updates market status (closed/open/paused) from verified reports
- Feed ID Validation: Ensures reports match the contract's specific price feed
- Timestamp Validation: Validates report expiration times
- Fallback Protection: Only allows execution when market is open (this functionality is used by the vaults that enable the `HookTargetMarketStatus` as a hook target)
- Owner Controls: Allows manual market status updates

Additional documentation: https://docs.chain.link/data-streams

Severity classification

Severity	Impact: High	Impact: Medium	Impact: Low

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

Impact - the technical, economic, and reputation damage of a successful attack

Likelihood - the chance that a particular vulnerability gets discovered and exploited

Severity - the overall criticality of the risk

Security Assessment Summary

***review commit hash* - e3d31d4f39df5d0ab193f25a5c4ee4ff58d29f01**

***fixes review commit hash* - 5eed034a9119f5405ee045349da926fafff9e7ba**

Scope

The following folders were in scope of the audit:

- `DataStreamsVerifier.sol`
- `HookTargetMarketStatus.sol`

The following number of issues were found, categorized by their severity:

- Critical & High: 0 issues
- Medium: 0 issues
- Low & Info: 4 issues

Findings Summary

ID	Title	Severity	Status
[L-01]	Single-step ownership transfer in use	Low	Acknowledged
[I-01]	The `setMarketStatus` can give a false impression of successful execution	Informational	Fixed
[I-02]	Rename parameter name to `_authorizedUpdater`	Informational	Acknowledged
[I-03]	Optimize token payload encoding in function `verify`	Informational	Fixed

Detailed Findings

[L-01] Single-step ownership transfer in use

Description

The `DataStreamsVerifier` implements `Ownable` library which implements single-step ownership transfer. In the event of transferring the ownership to an invalid address, all functions protected by the access control will become permanently unavailable.

```
abstract contract DataStreamsVerifier is Ownable {
```

Recommendations

It is recommended to use `Ownable2Step` instead.

[I-01] The `setMarketStatus` can give a false impression of successful execution

Description

The `setMarketStatus` function never reverts and it always passes. However, it does emit `MarketStatusUpdated` whenever a certain condition is met. Such an execution flow can give a false impression of successful processing, as the function does not revert and it does not emit an alternative event when the aforementioned condition is not met.

```
    function _setMarketStatus(uint32 _marketStatus, uint64
_lastUpdatedTimestamp) internal {
        if (marketStatus != _marketStatus && lastUpdatedTimestamp <=
_lastUpdatedTimestamp) {
            marketStatus = _marketStatus;
            lastUpdatedTimestamp = _lastUpdatedTimestamp;
            emit MarketStatusUpdated(_marketStatus,
_lastUpdatedTimestamp);
        }
    }
```

Recommendations

Consider an alternative execution flow, when the function reverts or emits an additional event indicating that the aforementioned condition.

[I-02] Rename parameter name to `_authorizedUpdater`

Description

As part of the update in the PR here, _authorizedCaller was updated to `_authorizedUpdater`. However, this name change was not reflected in the DataStreamsVerifier contract with the parameter name and state variable name (AUTHORIZED_CALLER).

```
    /// @notice Initializes the contract with required parameters
    /// @param _authorizedCaller Address authorized to verify reports
    /// @param _verifierProxy Address of the verifier proxy contract
    /// @param _expectedVersion Expected version of the report
    constructor(address _authorizedCaller, address _verifierProxy, uint16
_expectedVersion) Ownable(msg.sender) {
        AUTHORIZED_CALLER = _authorizedCaller;
```

Recommendations

Rename the variables to authorized updater.

[I-03] Optimize token payload encoding in function `verify`

Description

Function _verify() in DataStreamsVerifier.sol encodes the token payload by checking whether the LINK_TOKEN address was set on contract deployment or not. However, the contract currently only supports the fee processing required for the report verification using LINK tokens. Native tokens are not possible since no msg.value is passed.

```
// Verify the report on-chain using Chainlink's verifier
        return VERIFIER_PROXY.verify(_rawReport, LINK_TOKEN == address(0)
? bytes("") : abi.encode(LINK_TOKEN));
```

Recommendations

Consider improving the code as follows:

```
// Verify the report on-chain using Chainlink's verifier
        return VERIFIER_PROXY.verify(_rawReport, abi.encode(LINK_TOKEN));
```