



Security Review For Elara Finance



Collaborative Audit Prepared For: **Elara Finance**
Lead Security Expert(s): **Kvar**
thekmj
Date Audited: **May 29 - June 5, 2026**

Introduction

Elara lets approved users mint eUSD, a USD-denominated treasury asset, by depositing supported stablecoins, redeem when they want out, and stake for yield through sElUSD shares.

Collateral is held in external treasury wallets, while the Vault coordinates all flows. Staking yield comes from real NAV growth: an operator periodically syncs on-chain supply to the protocol's net asset value, after fees. The system is built for regulated use with KYC on mint and redeem, account controls, Chainlink pricing with peg protection, and upgradeable contracts under role-based access control. Initial targeted yields are in the mid-teens.

Scope

Repository: TrueFi-Protocol/elara-contracts-v2

Audited Commit: 463d6af3037952fb3febbbbbb446646302329b37

Final Commit: f15e36fa114ab47c0e2b3b6e406f0b9200727aaa

Files:

- contracts/base/ElaraBase.sol
- contracts/nav/adapters/AaveV3Adapter.sol
- contracts/nav/adapters/ConvexAdapter.sol
- contracts/nav/adapters/CurveStableAdapter.sol
- contracts/nav/adapters/ERC4626Adapter.sol
- contracts/nav/adapters/MorphoBlueAdapter.sol
- contracts/nav/adapters/PendleAdapter.sol
- contracts/nav/adapters/UniswapV3Adapter.sol
- contracts/nav/adapters/UniswapV4Adapter.sol
- contracts/nav/NAVAggregator.sol
- contracts/oracle/PriceOracle.sol
- contracts/staking/ElUSDStake.sol
- contracts/token/ElUSD.sol
- contracts/token/sElUSD.sol
- contracts/treasury/TreasuryAdapter.sol
- contracts/vault/KYCRegistry.sol
- contracts/vault/TokenRegistry.sol

- contracts/vault/Vault.sol
- interfaces/base/IElaraBase.sol
- interfaces/nav/INAVAggregator.sol
- interfaces/nav/IStrategyAdapter.sol
- interfaces/oracle/IPriceOracle.sol
- interfaces/staking/IEIUSDStake.sol
- interfaces/token/IEIUSD.sol
- interfaces/token/IsEIUSD.sol
- interfaces/treasury/ITreasuryAdapter.sol
- interfaces/vault/IKeyringCredentials.sol
- interfaces/vault/IKYRegistry.sol
- interfaces/vault/ITokenRegistry.sol
- interfaces/vault/IVault.sol

Final Commit Hash

f15e36fa114ab47c0e2b3b6e406f0b9200727aaa

Findings

Each issue has an assigned severity:

- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
3	5	4

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue H-1: AaveV3Adapter computes position value incorrectly, leading to incorrect yield distribution [RESOLVED]

Source: <https://github.com/sherlock-audit/2026-05-elara-finance-full-adapter-audit-may-29th-2026/issues/39>

Vulnerability Detail

`AaveV3Adapter::getPositionValue()` floors the supplied/borrowed net to 0 for each market.

This is only correct if the position is actually bad debt. However, in Aave a wallet can supply in market A and borrow in market B. In that case the NAV can be overstated.

For example:

1. The wallet supplies \$1000 worth of USDC and \$500 worth of WETH.
2. The wallet borrows \$1000 worth of WETH. This can still be healthy, because the wallet has \$1500 worth of collateral.
3. `getPositionValue()` computes \$1000 for the USDC market. For the WETH market, it computes \$500 supplied - \$1000 borrowed and floors it to 0. So the adapter returns \$1000, while the real net value is \$500.

Impact

If this adapter is used and this case happens, NAV can be significantly overstated. When `Vault::syncNAV()` is called, the overstated amount will be incorrectly distributed as yield to `E1USDStake`. This can make the vault insolvent because more yield is distributed than actually exists. Later, when the debt is repaid, stakers will receive negative yield. This can unfairly benefit users who unstake after the inflated sync, while later stakers can absorb the loss.

Recommendation

`AaveV3Adapter::getPositionValue()` should not floor each market independently. It should sum all supplied value and all borrowed value across markets, then subtract total debt from total supply at the end.

Issue H-2: UniswapV3Adapter double-counts the LP position, will inflate the reported NAV [RESOLVED]

Source: <https://github.com/sherlock-audit/2026-05-elara-finance-full-adapter-audit-may-29th-2026/issues/40>

Summary

NAVAggregator._computePositionValues() loops over every (wallet * adapter) pair. UniswapV3Adapter ignores the wallet argument and always returns the full position value for its tokenId. With N registered wallets, one Uniswap V3 position is counted N times.

Vulnerability Detail

NAVAggregator._computePositionValues() sums adapter values per wallet:

```
for (uint256 i; i < walletLen; ++i) {
    address wallet = _wallets[i];
    for (uint256 j; j < adapterLen; ++j) {
        total += IStrategyAdapter(_adapters[j]).getPositionValue(wallet);
    }
}
```

UniswapV3Adapter.getPositionValue() ignores wallet entirely:

```
function getPositionValue(address /*wallet*/) external view override returns
    ↪ (uint256 valueUSD1e18) {
    (,,,,, int24 tickLower, int24 tickUpper, uint128 liquidity,,,,) =
    ↪ positionManager.positions(tokenId);
    if (liquidity == 0) return 0;
}
```

Hence for every wallet added, the same Uniswap V3 LP position will be added into the NAV multiple times

Impact

Uniswap V3 LP positions will be double-counted, the reported NAV will be inflated

Tool Used

Manual Review

Recommendation

Check ownership in `getPositionValue()` and return 0 if the wallet doesn't own the NFT:

```
function getPositionValue(address wallet) external view override returns (uint256
↪ valueUSD1e18) {
    if (positionManager.ownerOf(tokenId) != wallet) return 0;
    // ...
}
```

Issue H-3: yieldDelta ignores _burnDebt, leading to incorrect yield calculation [RESOLVED]

Source: <https://github.com/sherlock-audit/2026-05-elara-finance-full-adapter-audit-may-29th-2026/issues/43>

Vulnerability Detail

When calculating yieldDelta, the Vault does:

```
int256 yieldDelta = int256(netNav) - int256(e1USDSupplyNow);
```

It does not consider the existing _burnDebt.

Consider this scenario:

1. E1USDStake is empty and netNav - e1USDSupplyNow = -10.
2. There is no staked e1USD to burn, so the 10 is added to _burnDebt.
3. Nothing changes and _syncNAVCore() is called again.

The Vault calculates the same -10 and adds it to _burnDebt again, even though this loss was already accounted for.

There is also a problem when NAV recovers. NAV first has to reach e1USD.totalSupply() before yieldDelta becomes positive. However, reaching that point already means the actual loss has been recovered. Only after that does the Vault start repaying _burnDebt.

This means that, even if the duplicate debt issue does not happen, NAV must effectively recover 20 to clear an actual loss of 10.

Recommendation

Include the existing burn debt in the calculation:

```
uint256 burnDebt = _stakeContract.burnDebt();  
  
int256 yieldDelta = int256(netNav) - int256(e1USDSupplyNow) + int256(burnDebt);
```

This prevents the same loss from being added multiple times and allows recovered NAV to repay the existing burn debt.

Issue M-1: Pending fees are not accrued before a fee configuration change [RESOLVED]

Source: <https://github.com/sherlock-audit/2026-05-elara-finance-full-adapter-audit-may-29th-2026/issues/31>

Vulnerability Detail

`Vault.sol::addFee()`, `removeFee()`, and `setFeePercent()` do not accrue pending fees before changing the fee configuration. Since management fees are only accrued in `syncNAV()` using the current fee configuration for the entire elapsed period, fee changes can apply retroactively to the period before the change.

Impact

This can overcharge or undercharge fees depending on whether a fee was added, removed, or modified before the next `syncNAV()`.

Recommendation

Accrue pending fees before changing the fee configuration.

Discussion

Kvarr

This is not completely fixed.

```
function setManagementFee(uint256 newFeePercent) external override restricted
↪ nonReentrant {
    ...
    if (address(_navAggregator) != address(0) && address(_stakeContract) !=
↪ address(0) && elUSD.totalSupply() != 0)
    {
        _syncNAVCore(); // it does not always sync it
    }
    ...
    // _lastMgmtFeeTimestamp is not updated if it skips the sync
}
```

I have left comments in the code snippet to make it easier. Basically, if it skips syncing when the `managementFee` is set/changed, it will apply retroactively on later accrual and will accrue more or will set it to `block.timestamp` if it hasn't been set yet, which will accrue less.

I'd recommend updating `_lastMgmtFeeTimestamp` in `setManagementFee()` if it skipped syncing.

Issue M-2: E1USD blacklist functionality can be bypassed by staking in E1USDStake [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2026-05-elara-finance-full-adapter-audit-may-29th-2026/issues/33>

This issue has been acknowledged by the team but won't be fixed at this time.

Vulnerability Detail

E1USD has blacklist functionality, however sE1USD does not. A malicious actor can hide his E1USD the following way:

1. Have a large amount of E1USD, expecting to be blocked, or frontrunning a block.
2. Stake into E1USDStake.
3. The user is blacklisted from E1USD.
4. The user can now transfer his sE1USD shares to a non-blocked address and unstake them.

Impact

The E1USD blacklist functionality can be bypassed.

Recommendation

Implement similar blacklist functionality for sE1USD.

Issue M-3: Users can frontrun a negative yield sync by unstaking first, bypassing the loss socialization [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2026-05-elara-finance-full-adapter-audit-may-29th-2026/issues/35>

This issue has been acknowledged by the team but won't be fixed at this time.

Vulnerability Detail

When `syncNAV()` is called and there is negative yield, the vault calls `E1USDStake::decreaseYield()`. However, the covered amount is capped by `_totalAccountedE1USD`.

```
uint256 available = _totalAccountedE1USD;
covered = amount < available ? amount : available;
uint256 uncovered = amount - covered;

_totalAccountedE1USD = available - covered;

if (uncovered > 0) {
    _burnDebt += uncovered;
    emit IE1USDStake.LossUncovered(uncovered, _burnDebt);
}
```

Unstaking has no delay and no fee. This means users can frontrun a negative yield sync by unstaking first, reducing `_totalAccountedE1USD` below the loss amount.

Impact

Part of the loss becomes uncovered burn debt, even though it could have been completely absorbed by the staking pool if users had not unstaked right before the sync.

Recommendation

Add an unstaking delay or an unstaking fee.

Issue M-4: Native ETH is not supported, even though NAVAggregator attempts to support it [RESOLVED]

Source: <https://github.com/sherlock-audit/2026-05-elara-finance-full-adapter-audit-may-29th-2026/issues/37>

Vulnerability Detail

Wallets are supposed to be able to hold native ETH. NAVAggregator::_computeWalletBalances() tries to implement this by using wallet.balance when the token is address(0), but it then prices the asset through PriceOracle.getPrice(address(0)).

```
if (tc.token == address(0)) {
    balance = wallet.balance;
} else {
    balance = IERC20(tc.token).balanceOf(wallet);
}

if (balance == 0) continue;

uint256 price = _oracle.getPrice(tc.token);
```

However, PriceOracle rejects address(0) in all feed configuration functions:

```
if (token == address(0)) revert IPriceOracle.InvalidAddress(token);
```

So no native ETH price feed can be configured. As a result, registering native ETH in NAVAggregator causes getNAV() to revert whenever a tracked wallet has a non-zero ETH balance.

Impact

Native ETH is not actually supported, even though NAVAggregator attempts to support it.

Recommendation

Allow address(0) to be configured in PriceOracle for native ETH.

Issue M-5: MorphoBlueAdapter is unusable [RESOLVED]

Source: <https://github.com/sherlock-audit/2026-05-elara-finance-full-adapter-audit-may-29th-2026/issues/38>

Vulnerability Detail

MorphoBlueAdapter assumes this function exists on the deployed Morpho contract:

```
morpho.expectedSupplyAssets(params, wallet)
```

However, Morpho does not have that function. This function is inside MorphoBalancesLib. This library function is not part of Morpho, it is helper code that is intended to be used by integrators.

Impact

MorphoBlueAdapter cannot get the position value, because it will revert. Thus the adapter is unusable.

Recommendation

Remove the local IMorpho interface and import Morpho's real IMorpho plus MorphoBalance sLib, then use the library function instead.

Issue L-1: Yield distribution should check whether there is meaningful sE1USD supply [RESOLVED]

Source: <https://github.com/sherlock-audit/2026-05-elara-finance-full-adapter-audit-may-29th-2026/issues/32>

Vulnerability Detail

`syncNAV()` can distribute positive yield to `E1USDStake` without checking whether there is meaningful `sE1USD` supply. If the supply is zero or very low, most of the yield can be absorbed by the virtual share offset, causing an inflated share price without actually distributing that yield to real users.

Recommendation

Consider checking whether there is meaningful `sE1USD` total supply before distributing yield.

Issue L-2: The recipient KYC restriction can be bypassed by transferring E1USD [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2026-05-elara-finance-full-adapter-audit-may-29th-2026/issues/34>

This issue has been acknowledged by the team but won't be fixed at this time.

Vulnerability Detail

`Vault::mint()` enforces KYC checks for `msg.sender` and `recipient`. However, E1USD transfers do not check KYC. A KYC'd user can mint to themselves and then transfer the E1USD to a non-KYC address, bypassing the recipient KYC restriction.

Recommendation

Check KYC also in `E1USD.sol::_update()`.

Issue L-3: syncOrphanedDonations() should repay _burnDebt first. [RESOLVED]

Source: <https://github.com/sherlock-audit/2026-05-elara-finance-full-adapter-audit-may-29th-2026/issues/36>

Vulnerability Detail

syncOrphanedDonations() adds donated eUSD into _totalAccountedElUSD, but does not use it to repay _burnDebt. Even if donated eUSD could cover the debt, staking/unstaking remain blocked until future positive yield repays it through syncNAV().

Recommendation

syncOrphanedDonations() should repay _burnDebt first.

Discussion

Kvarr

```
if (debt > 0) {
    uint256 debtRepayment = orphaned < debt ? orphaned : debt;
    _burnDebt -= debtRepayment;
    emit IEUSDStake.BurnDebtPaid(debtRepayment, _burnDebt);
}
```

When orphaned eUSD repays _burnDebt, the corresponding tokens must be burned because they already exist in totalSupply. In contrast, when the Vault repays burn debt in the vault, no tokens are burned because it repays the debt by minting less eUSD.

Also, when you burn the eUSD, update _totalAccountedElUSD with the new eUSD balance.

Issue L-4: PendleAdapter: Unclaimed YT/LP yield and rewards are excluded from the NAV [RESOLVED]

Source: <https://github.com/sherlock-audit/2026-05-elara-finance-full-adapter-audit-may-29th-2026/issues/41>

Vulnerability Detail

`PendleAdapter.getPositionValue()` values only the principal token balances (PT, YT, LP) using TWAP oracle rates. Unclaimed yield accrued to YT holders and pending LP rewards are not included.

Although the NAV will be counted once the yield is claimed, this behavior is not documented on the Pendle Adapter (unlike the case for Convex/Uniswap adapter).

Recommendation

Document such behavior.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.