ABDK CONSULTING

SMART CONTRACT AUDIT

SENSE. Part II

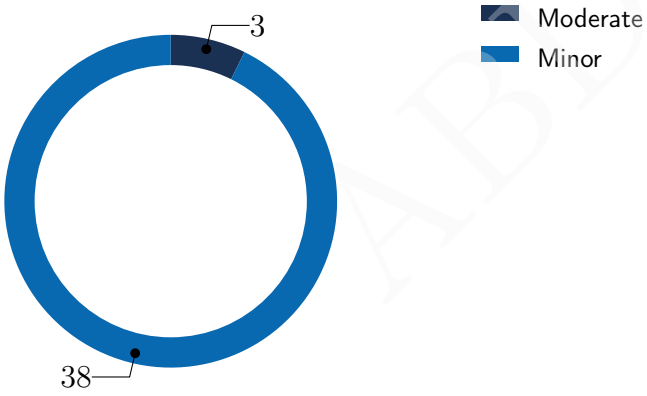
Solidity

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
18th March 2022

We've been asked to review a pull request as a fix to our earlier report. We found 3 moderate and a few less important issues.



Findings

ID	Severity	Category	Status
CVF-1	Minor	Procedural	Fixed
CVF-2	Minor	Readability	Fixed
CVF-3	Minor	Suboptimal	Opened
CVF-4	Minor	Readability	Fixed
CVF-5	Minor	Bad datatype	Info
CVF-6	Minor	Suboptimal	Fixed
CVF-7	Minor	Unclear behavior	Info
CVF-8	Minor	Procedural	Info
CVF-9	Minor	Suboptimal	Fixed
CVF-10	Minor	Flaw	Info
CVF-11	Minor	Suboptimal	Fixed
CVF-12	Minor	Bad datatype	Fixed
CVF-13	Minor	Bad naming	Fixed
CVF-14	Minor	Unclear behavior	Fixed
CVF-15	Minor	Readability	Fixed
CVF-16	Minor	Bad datatype	Info
CVF-17	Minor	Bad datatype	Info
CVF-18	Minor	Suboptimal	Opened
CVF-19	Minor	Bad datatype	Info
CVF-20	Minor	Bad naming	Fixed
CVF-21	Moderate	Suboptimal	Info
CVF-22	Minor	Bad datatype	Fixed
CVF-23	Minor	Readability	Fixed
CVF-24	Minor	Bad datatype	Info
CVF-25	Minor	Documentation	Fixed
CVF-26	Minor	Procedural	Info
CVF-27	Minor	Bad datatype	Fixed

ID	Severity	Category	Status
CVF-28	Minor	Readability	Fixed
CVF-29	Moderate	Flaw	Info
CVF-30	Minor	Bad naming	Info
CVF-31	Minor	Bad datatype	Info
CVF-32	Minor	Overflow/Underflow	Fixed
CVF-33	Minor	Procedural	Fixed
CVF-34	Minor	Procedural	Fixed
CVF-35	Minor	Suboptimal	Fixed
CVF-36	Minor	Suboptimal	Fixed
CVF-37	Minor	Procedural	Info
CVF-38	Minor	Procedural	Info
CVF-39	Minor	Procedural	Info
CVF-40	Minor	Procedural	Fixed
CVF-41	Moderate	Flaw	Fixed

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	March 18, 2022	D. Khovratovich	Initial Draft
0.2	March 18, 2022	D. Khovratovich	Minor revision
1.0	March 18, 2022	D. Khovratovich	Release

Contact

D. Khovratovich

khovratovich@gmail.com

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.
We have reviewed the contracts at the pull request:

- core/adapters/compound/CAdapter.sol
- core/adapters/lido/WstETHAdapter.sol
- core/adapters/BaseAdapter.sol
- core/adapters/BaseFactory.sol
- core/adapters/CropFactory.sol
- core/external/balancer/Pool.sol
- core/external/FixedMath.sol
- core/tokens/Claim.sol
- core/Divider.sol
- core/Periphery.sol
- fuse/oracles/Target.sol
- fuse/oracles/Zero.sol
- utils/libs/Errors.sol
- utils/libs/Levels.sol

The fixes were provided at several pull requests:

- pull 181
- pull 184

2.1 About ABDK

ABDK Consulting, established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function. The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

2.2 Disclaimer

Note that the performed audit represents current best practices and smart contract standards which are relevant at the date of publication. After fixing the indicated issues the smart contracts should be re-audited.

2.3 Methodology

The methodology is not a strict formal procedure, but rather a collection of methods and tactics that combined differently and tuned for every particular project, depending on the project structure and and used technologies, as well as on what the client is expecting from the audit. In current audit we use:

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows code best practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.

- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places and that their visibility scopes and access levels are relevant. At this phase we understand overall system architecture and how different parts of the code are related to each other.

- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and is done properly. At this phase we understand user roles and permissions, as well as what assets the system ought to protect.

- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check that code actually does what it is supposed to do, that algorithms are optimal and correct, and that proper data types are used. We also check that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

3 Detailed Results

3.1 CVF-1

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** Levels.sol

Recommendation Should be "^0.8.0".
Client Comment Resolved. We decided to drop the requirement down to >= 0.7.0 since it's a general util contract that doesn't rely on any recent sol features.

Listing 1:

```
+pragma solidity ^0.8.6;
```

3.2 CVF-2

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** Levels.sol

Recommendation Consider using bit masks instead of bit indexes to make the code easier to read, such as: _INIT_BIT = 0x1; _ISSUE_BIT = 0x2; _COMBINE_BIT = 0x4; etc.
Client Comment Resolved. Suggestion adopted.

Listing 2:

```
+uint256 private constant _INIT_BIT = 0;
+uint256 private constant _ISSUE_BIT = 1;
+uint256 private constant _COMBINE_BIT = 2;
+uint256 private constant _COLLECT_BIT = 3;
+uint256 private constant _REDEEM_ZERO_BIT = 4;
+uint256 private constant _REDEEM_ZERO_HOOK_BIT = 5;
```

3.3 CVF-3

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Levels.sol

Recommendation != 0 would be more efficient and shorter. Addition comment: Should be ==0.
Client Comment Not sure we understand this suggestion entirely. What's the suggested alternative? :)

Listing 3:

```
13 +return level & (2**_INIT_BIT) != 2**_INIT_BIT;

17 +return level & (2**_ISSUE_BIT) != 2**_ISSUE_BIT;

21 +return level & (2**_COMBINE_BIT) != 2**_COMBINE_BIT;

25 +return level & (2**_COLLECT_BIT) != 2**_COLLECT_BIT;

29 +return level & (2**_REDEEM_ZERO_BIT) != 2**_REDEEM_ZERO_BIT;

33 +return level & (2**_REDEEM_ZERO_HOOK_BIT) != 2**
     ↪ _REDEEM_ZERO_HOOK_BIT;
```

3.4 CVF-4

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** Divider.sol

Recommendation Should be 'adapter'.
Client Comment Resolved. We now have an adapter meta where we store this variable, and the comment no longer has this inconsitency.

Listing 4:

```
49 +/// @notice adaper -> max amount of Target allowed to be issued
```

3.5 CVF-5

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** Divider.sol

Recommendation The type of this argument should be "Adapter".
Client Comment Kept as is. Differences in preferred style

Listing 5:

```
88  address adapter,

162 address adapter,
```

3.6 CVF-6

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Divider.sol

Description Conversions to the "uint256" type are redundant, as the "level()" function returns "uint128" with is implicitly convertible to "uint256".
Client Comment Resolved. We're caching this in the adapter meta as well, and have now set the datatype so that it packs more cleanly.

Listing 6:

```
170 +uint256 level = uint256(Adapter(adapter).level());

255 +uint256 level = uint256(Adapter(adapter).level());

309 +uint256 level = uint256(Adapter(adapter).level());

411 +uint256 level = uint256(Adapter(adapter).level());
```

3.7 CVF-7

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** Divider.sol

Recommendation In case the user already has some Claim, it would be simpler to collect earnings on that Claim first, then add collected Target to the provided Target amount and use it to issue more Zero and Claim. Also, in case the current scale is less than the current max scale, the user will be able to collect some Target right after issuing Claim, so it would be logical to add this addition Target amount to the Target amount provided by the user. See this memo for details: https://hackmd.io/@abdk/SkyOO6cdY#Issue
Client Comment Kept as is. Minimize changes before launch

Listing 7:

```
222 +lscales[adapter][maturity][msg.sender] = lscales[adapter][
    ↪ maturity][msg.sender] == 0
+     ? scale
+     : _reweightLScale(
+         adapter,
+         maturity,
+         Claim(series[adapter][maturity].claim).balanceOf(msg.
    ↪ sender),
+         uBal,
+         msg.sender,
230 +         scale
+     );
```

3.8 CVF-8

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** Divider.sol

Recommendation This calculation should be moved into the "then" branch of the conditional statement below.
Client Comment Kept as is. Unclear how this would work since the conditional requires zBal

Listing 8:

```
347 +uint256 zShare = FixedMath.WAD - series[adapter][maturity].tilt
    ↪ ;
```

3.9 CVF-9

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Divider.sol

Recommendation Consider using utility functions such as "fmul" and "fdiv" for A * B / C expressions.
Client Comment Resolved. Suggestion adopted.

Listing 9:

```
352 +tBal = (uBal * zShare) / series[adapter][maturity].mscale;

525 +     (uBal * FixedMath.WAD) /
    +     series[adapter][maturity].maxscale -
    +     (uBal * zShare) /
    +     series[adapter][maturity].mscale;
```

3.10 CVF-10

- **Severity** Minor
- **Status** Info
- **Category** Flaw
- **Source** Divider.sol

Description There is not explicit check that "_divider" is not zero, thus the "init" function could be called several times.
Recommendation Consider adding such check.
Client Comment Kept as is. The impact on the actual functioning of the protocol seems minimal

Listing 10:

```
817 +require(divider == address(0));
     divider = _divider;
```

3.11 CVF-11

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** BaseAdapter.sol

Description Why changing to 16 bits if only two values are used?
Client Comment Resolved. Adapter parameter storage was re-worked.

Listing 11:

```
 48 +uint16 mode; // 0 for monthly, 1 for weekly
```

3.12 CVF-12

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** BaseAdapter.sol

Recommendation This should be a named constant.
Client Comment Resolved. Suggestion adopted.

Listing 12:

```
161 +return 31; // Returns '31' by default, which enables all
    ↪ Divider lifecycle methods
```

3.13 CVF-13

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** BaseAdapter.sol

Description The semantics of the returned values is unclear.
Recommendation Consider giving descriptive names to the returned values and/or adding a documentation comment.
Client Comment Resolved. Suggestion adopted.

Listing 13:

```
217 +address,
    +address,
    +uint256
```

3.14 CVF-14

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** WstETHAdapter.sol

Description The denominator "100e18" is unusual and there is no good reason for it to be so high.
Recommendation Consider using the standard denominator 1e18 (the "SLIPPAGE_TOLERANCE" value should be changed to 0.005e18 in such case). Also, consider optimizing the expression like this: minDy - minDy / 200
Client Comment Resolved. Suggestion adopted.

Listing 14:

```
106 +(minDy * (100e18 - SLIPPAGE_TOLERANCE)) / 100e18
```

3.15 CVF-15

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** Errors.sol

Recommendation Consider using Solidity errors instead of string constants: https://docs.soliditylang.org/en/v0.8.11/structure-of-a-contract.html#errors
Client Comment Resolved. Suggestion adopted.

Listing 15:

```
5 library Errors {
```

3.16 CVF-16

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** Periphery.sol

Recommendation The type of this mapping should be "mapping (Adapter => Factory)".
Client Comment Kept as is. Differences in preferred style

Listing 16:

```
60 +mapping(address => address) public factory; // adapter ->
    ↪ factory
```

3.17 CVF-17

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** Periphery.sol

Recommendation The type of this argument should be "SpaceFactoryLike".
Client Comment Kept as is. Differences in preferred style

Listing 17:

```
71 +address _spaceFactory,
```

3.18 CVF-18

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Periphery.sol

Description The "decimals" property is used by UI to render token amounts is a human-friendly way. Use of this property in smart contracts is discouraged.
Recommendation Consider treating all token amounts as integers.

Listing 18:

```
 93  uint256 stakeDecimals = ERC20(stake).decimals();

501 +require(cBal * 10**(18 - ERC20(claim).decimals()) > 1e12,
    ↪ Errors.SwapTooSmall);

649 +uint256 tBase = 10**ERC20(Adapter(adapter).getTarget()).
    ↪ decimals();

734 +uint256 acceptableError = ERC20(claim).decimals() < 9 ? 1 : 1
    ↪ e10 / 10**(18 - ERC20(claim).decimals());
```

3.19 CVF-19

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** Periphery.sol

Recommendation The type of the "f" argument should be "Factory".
Client Comment Kept as is. Differences in preferred style

Listing 19:

```
120 +function onboardAdapter(address f, address target) external
    ↪ returns (address adapterClone) {
```

3.20 CVF-20

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** Periphery.sol

Description The semantics of the returned values is unclear.
Recommendation Consider giving descriptive names to the returned values and/or adding documentation comments.
Client Comment Resolved. Suggestion adopted.

Listing 20:

```
275 +uint256,
    +uint256,
    +uint256

298 +uint256,
    +uint256,
300 +uint256

378 +uint256,
    +uint256,
380 +uint256

556 +uint256,
    +uint256,
    +uint256
```

3.21 CVF-21

- **Severity** Moderate
- **Status** Info
- **Category** Suboptimal
- **Source** Periphery.sol

Description This will revert in case Claim has more than 18 decimals.
Recommendation Consider checking like this: tBal * 1e18 > 10**(12 + ERC20(claim).decimals())
Client Comment Comment added, that the contracts won't work with tokens using more than 18 decimals is accepted.

Listing 21:

```
501 +require(cBal * 10**(18 - ERC20(claim).decimals()) > 1e12,
    ↪ Errors.SwapTooSmall);
```

3.22 CVF-22

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Periphery.sol

Recommendation The "1e12" value should be a named constant.
Client Comment Resolved. Suggestion adopted.

Listing 22:

```
501 +require(cBal * 10**(18 - ERC20(claim).decimals()) > 1e12,
    ↪ Errors.SwapTooSmall);
```

3.23 CVF-23

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** Periphery.sol

Recommendation It would be more readable to render the "1e12" value as "0.000001e18".
Client Comment Resolved. Suggestion adopted.

Listing 23:

```
501 +require(cBal * 10**(18 - ERC20(claim).decimals()) > 1e12,
    ↪ Errors.SwapTooSmall);
```

3.24 CVF-24

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** Periphery.sol

Recommendation There should be named constants for the valid modes.
Client Comment Kept as is.

Listing 24:

```
573 +if (mode == 0) {
```

3.25 CVF-25

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** Periphery.sol

Description The comment is confusing without further explanation.
Recommendation Consider removing it.
Client Comment Resolved. Suggetion adopted.

Listing 25:

```
651 +tBal.fmul(zeroiBal.fdiv(Adapter(adapter).scale().fmul(
    ↪ targetiBal, tBase) + zeroiBal, FixedMath.WAD), tBase); //
    ↪ ABDK formula
```

3.26 CVF-26

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** Periphery.sol

Recommendation According to our formula, it should be the max scale rather than the current scale. However, as the issue logic doesn't collect the difference between the current scale and the max scale, this formula is probably correct.
Client Comment Kept as is. Minimize changes before launch.

Listing 26:

```
651 +tBal.fmul(zeroiBal.fdiv(Adapter(adapter).scale().fmul(
    ↪ targetiBal, tBase) + zeroiBal, FixedMath.WAD), tBase); //
    ↪ ABDK formula
```

3.27 CVF-27

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Periphery.sol

Recommendation The "1.e10" value should be a named constant.
Client Comment Resolved. Suggestion adopted.

Listing 27:

```
734 +uint256 acceptableError = ERC20(claim).decimals() < 9 ? 1 : 1
    ↪ e10 / 10**(18 - ERC20(claim).decimals());
```

3.28 CVF-28

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** Periphery.sol

Recommendation It would be more readable to render the "1e10" value as "0.00000001e18".
Client Comment Resolved. Suggestion adopted.

Listing 28:

```
734 +uint256 acceptableError = ERC20(claim).decimals() < 9 ? 1 : 1
    ↪ e10 / 10**(18 - ERC20(claim).decimals());
```

3.29 CVF-29

- **Severity** Moderate
- **Status** Info
- **Category** Flaw
- **Source** Periphery.sol

Description This will revert in case claim has more than 18 decimals.
Recommendation Consider handling such case properly.
Client Comment Comment added, that the contracts won't work with tokens using more than 18 decimals is accepted.

Listing 29:

```
734 +uint256 acceptableError = ERC20(claim).decimals() < 9 ? 1 : 1
    ↪ e10 / 10**(18 - ERC20(claim).decimals());
```

3.30 CVF-30

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** Periphery.sol

Recommendation Events are usually named via nouns, such as "FactoryChange" or "Factory", and "Swap".
Client Comment Kept as is. Differences in preferred style.

Listing 30:

```
828 +event FactoryChanged(address indexed adapter, bool indexed isOn
    ↪ );

831 +event Swapped(
```

3.31 CVF-31

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** Periphery.sol

Recommendation The types of these parameters should be "IERC20".
Client Comment Kept as is. Differences in preferred style.

Listing 31:

```
834 +address assetIn,
    +address assetOut,
```

3.32 CVF-32

- **Severity** Minor
- **Status** Fixed
- **Category** Overflow/Underflow
- **Source** FixedMath.sol

Description Phantom overflow is possible in these functions.
Recommendation Consider using a non-overflowing "muldiv" function.
Client Comment Resolved. New solmate audited math lib used internally instead now.

Listing 32:

```
23 +function fmul(uint256 x, uint256 y) internal pure returns (
   ↪ uint256 z) {

41 +function fmulUp(uint256 x, uint256 y) internal pure returns (
   ↪ uint256 z) {

59 +function fdiv(uint256 x, uint256 y) internal pure returns (
   ↪ uint256 z) {

75 +function fdivUp(uint256 x, uint256 y) internal pure returns (
   ↪ uint256 z) {
```

3.33 CVF-33

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** FixedMath.sol

Recommendation Putting "WAD - 1" into brackets would allow compiler to recognize this subexpression as a compile-time constant.
Client Comment Resolved. New solmate audited math lib used internally instead now.

Listing 33:

```
42 +z = x * y + WAD - 1;
```

3.34 CVF-34

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** FixedMath.sol

Recommendation Compiler is probably smart enough to not perform a division by zero check when the denominator is a compile-time constant.
Client Comment Resolved. New solmate audited math lib used internally instead now.

Listing 34:

```
43 +unchecked {
```

3.35 CVF-35

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** FixedMath.sol

Recommendation Brackets are redundant here.
Client Comment Resolved. New solmate audited math lib used internally instead now.

Listing 35:

```
44 +z /= (WAD);
```

3.36 CVF-36

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** FixedMath.sol

Recommendation Brackets are redundant here.
Client Comment Resolved. New solmate audited math lib used internally instead now.

Listing 36:

```
60 +z = (x * WAD) / y;
```

3.37 CVF-37

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** CAdapter.sol

Recommendation This interface should be moved to a separate file named "IWETH.sol". Also, consider using the interface file from WETH.
Client Comment Kept as is. Differences in preferred style.

Listing 37:

```
11 +interface IWETH {
```

3.38 CVF-38

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** CAdapter.sol

Recommendation This interface should be moved to a file named "CETHTokenInterface.sol".
Client Comment Kept as is. Differences in preferred style

Listing 38:

```
49 +interface CETHTokenInterface {
```

3.39 CVF-39

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** CAdapter.sol

Description Hardcoding addresses makes it harder to test contracts.
Recommendation Consider passing the address as a constructor argument and storing in an immutable variable.
Client Comment Kept as is. Differences in preferred style

Listing 39:

```
74 +address public constant WETH = 0
     ↪ xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;
```

3.40 CVF-40

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** CAdapter.sol

Recommendation This logic could be moved to a utility function.
Client Comment Resolved. Suggestion adopted.

Listing 40:

```
98 +return uDecimals >= 8 ? exRate / 10**(uDecimals - 8) : exRate *
     ↪ 10**(8 - uDecimals);

104 +return uDecimals >= 8 ? exRate / 10**(uDecimals - 8) : exRate *
     ↪ 10**(8 - uDecimals);
```

3.41 CVF-41

- **Severity** Moderate
- **Status** Fixed
- **Category** Flaw
- **Source** CAdapter.sol

Description Symbols are not unique, so this check is unreliable.
Recommendation Consider passing the cETH address to the constructor and storing in an immutable variable.
Client Comment Resolved. Still a constant, but the address is now used to check rather than the symbol.

Listing 41:

```
179 +return keccak256(abi.encodePacked(ERC20(target).symbol())) ==
     ↪ keccak256(abi.encodePacked("cETH"));
```