ABDK CONSULTING

SMART CONTRACT AUDIT

YIELD

yieldspace-tv

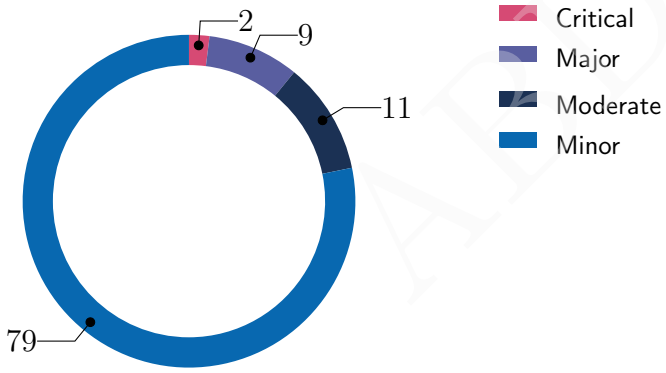
Solidity

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
29th July 2022

We've been asked to review 11 files in a Github repository. We found 2 critical, 9 major, and a few less important issues. All identified critical issues have been fixed.



Findings

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

ID	Severity	Category	Status
CVF-28	Minor	Bad datatype	Info
CVF-29	Minor	Bad datatype	Info
CVF-30	Major	Unclear behavior	Info
CVF-31	Moderate	Unclear behavior	Fixed
CVF-32	Minor	Bad datatype	Info
CVF-33	Moderate	Unclear behavior	Fixed
CVF-34	Minor	Unclear behavior	Info
CVF-35	Minor	Unclear behavior	Info
CVF-36	Major	Documentation	Info
CVF-37	Minor	Unclear behavior	Info
CVF-38	Minor	Documentation	Fixed
CVF-39	Major	Unclear behavior	Fixed
CVF-40	Minor	Unclear behavior	Fixed
CVF-41	Minor	Suboptimal	Info
CVF-42	Minor	Unclear behavior	Info
CVF-43	Minor	Suboptimal	Fixed
CVF-44	Minor	Bad datatype	Info
CVF-45	Minor	Overflow/Underflow	Info
CVF-46	Minor	Unclear behavior	Fixed
CVF-47	Minor	Unclear behavior	Info
CVF-48	Minor	Unclear behavior	Info
CVF-49	Critical	Flaw	Fixed
CVF-50	Minor	Suboptimal	Info
CVF-51	Minor	Suboptimal	Fixed
CVF-52	Minor	Documentation	Fixed
CVF-53	Moderate	Flaw	Fixed
CVF-54	Minor	Readability	Fixed
CVF-55	Moderate	Unclear behavior	Info
CVF-56	Major	Unclear behavior	Info
CVF-57	Minor	Suboptimal	Fixed

ID	Severity	Category	Status
CVF-58	Moderate	Flaw	Info
CVF-59	Minor	Documentation	Fixed
CVF-60	Major	Suboptimal	Fixed
CVF-61	Moderate	Unclear behavior	Info
CVF-62	Minor	Suboptimal	Info
CVF-63	Minor	Suboptimal	Fixed
CVF-64	Minor	Procedural	Fixed
CVF-65	Minor	Suboptimal	Fixed
CVF-66	Minor	Suboptimal	Info
CVF-67	Minor	Suboptimal	Info
CVF-68	Minor	Procedural	Info
CVF-69	Minor	Suboptimal	Fixed
CVF-70	Critical	Flaw	Fixed
CVF-71	Minor	Readability	Info
CVF-72	Minor	Bad datatype	Info
CVF-73	Minor	Overflow/Underflow	Info
CVF-74	Minor	Suboptimal	Info
CVF-75	Minor	Procedural	Info
CVF-76	Minor	Bad datatype	Info
CVF-77	Minor	Procedural	Info
CVF-78	Minor	Bad datatype	Info
CVF-79	Minor	Suboptimal	Info
CVF-80	Minor	Procedural	Info
CVF-81	Minor	Documentation	Info
CVF-82	Minor	Documentation	Info
CVF-83	Minor	Bad datatype	Info
CVF-84	Minor	Procedural	Info
CVF-85	Minor	Procedural	Info
CVF-86	Minor	Bad naming	Info
CVF-87	Minor	Documentation	Info

ID	Severity	Category	Status
CVF-88	Minor	Documentation	Fixed
CVF-89	Minor	Procedural	Info
CVF-90	Minor	Unclear behavior	Info
CVF-91	Minor	Unclear behavior	Info
CVF-92	Major	Unclear behavior	Info
CVF-93	Minor	Bad datatype	Info
CVF-94	Minor	Unclear behavior	Info
CVF-95	Major	Flaw	Info
CVF-96	Minor	Procedural	Info
CVF-97	Minor	Bad naming	Info
CVF-98	Minor	Bad naming	Info
CVF-99	Minor	Procedural	Info
CVF-100	Minor	Bad naming	Info
CVF-101	Minor	Bad naming	Info

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	July 29, 2022	D. Khovratovich	Initial Draft
0.2	July 29, 2022	D. Khovratovich	Minor revision
1.0	July 29, 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 commit a8f47b0b:

- interfaces/IERC20Like.sol
- interfaces/IERC4626.sol
- interfaces/IEToken.sol
- interfaces/IPool.sol
- interfaces/IYVToken.sol
- Pool/Modules/PoolEuler.sol
- Pool/Modules/PoolNonTv.sol
- Pool/Modules/PoolYearnVault.sol
- Pool/Pool.sol
- Pool/PoolErrors.sol
- Pool/PoolEvents.sol
- Pool/PoolImports.sol
- YieldMath.sol

The fixes were provided in a new pull request.

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** PoolNonTv.sol

Recommendation Should be "^0.8.0" or "^0.8.13" in case there is something special about this particular version. Also relevant for the next files: YieldMath.sol, PoolEvents.sol, PoolErrors.sol, Pool.sol, PoolYearnVault.sol, PoolImports.sol, PoolEuler.sol, IYVToken.sol, IEToken.sol, IERC4626.sol, IERC20Like.sol.

Listing 1:

```
2 pragma solidity >=0.8.13;
```

3.2 CVF-2

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

Recommendation The types of these arguments could be more specific.

Listing 2:

```
34 address base_,
   address fyToken_,
```

3.3 CVF-3

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** PoolNonTv.sol

Description This value is calcualted every time.
Recommendation Consider calculating once and storing in an internal variable.
Client Comment No thank you. Some small gas inefficiency is worth the trade off of introducing some relatively significant complexity as this fn is used by the constructor and therefore may not access immutables so there would be a lot of new code added to the core contract for the gas savings on just the PoolEuler.sol contract.

Listing 3:

```
54 return uint256(10**baseDecimals);
```

3.4 CVF-4

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

Description We didn't review these files.

Listing 4:

```
13 import {Exp64x64} from "./Exp64x64.sol";
   import {Math64x64} from "./Math64x64.sol";
```

3.5 CVF-5

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** YieldMath.sol

Description This looks like an internal document, referring to it from public source code looks odd.
Recommendation Consider moving the content to a proper whitepaper and referring this whitepaper instead.
Client Comment No thank you. It's a list of links that includes the whitepaper and other useful links.

Listing 5:

```
45  /// https://docs.google.com/spreadsheets/d/14
      ↪ K_McZhlgSXQfi6nFGwDvDh4BmOu6_Hczi_sFreFfOE/

72              /* https://docs.google.com/spreadsheets/d/14
                   ↪ K_McZhlgSXQfi6nFGwDvDh4BmOu6_Hczi_sFreFfOE/

196     /* https://docs.google.com/spreadsheets/d/14
           ↪ K_McZhlgSXQfi6nFGwDvDh4BmOu6_Hczi_sFreFfOE/

279     /* https://docs.google.com/spreadsheets/d/14
           ↪ K_McZhlgSXQfi6nFGwDvDh4BmOu6_Hczi_sFreFfOE/

402     /* https://docs.google.com/spreadsheets/d/14
           ↪ K_McZhlgSXQfi6nFGwDvDh4BmOu6_Hczi_sFreFfOE/
```

3.6 CVF-6

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

Description The "pow(x, y, z)" function not just calculates x^(y/z) but also normalizes the result to fit into 64.64 fixed point number, i.e. it actually calculates: x^(y/z) * (2^63)^(1 - y/z)
Recommendation Consider mentioning this fact in comments.

Listing 6:

```
89      // za = c/μ  * (normalizedSharesReserves ** a)
96      // ya = fyTokenReserves ** a
110     // zxa = c/μ  * zx ** a
120 // result = fyTokenReserves - (sum ** (1/a))
219         // za = c/μ  * (normalizedSharesReserves ** a)
226         // ya = fyTokenReserves ** a
229         // yxa = (fyTokenReserves + x) ** a   # x is aka △y
304     // za = c/μ  * (normalizedSharesReserves ** a)
311     // ya = fyTokenReserves ** a
325     // zxa = c/μ  * zx ** a
333 // result = fyTokenReserves - (sum ** (1/a))
417 // za = c/μ * (normalizedSharesReserves ** a)
421 // ya = fyTokenReserves ** a
424 // yxa = (fyTokenReserves - x) ** aβ
```

3.7 CVF-7

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

Description The "sum" value is not checked to fit into 128 bits.
Recommendation Consider adding appropriate check.

Listing 7:

```
117 sum = za + ya - zxa;
```

3.8 CVF-8

- **Severity** Moderate
- **Status** Fixed
- **Category** Overflow/Underflow
- **Source** YieldMath.sol

Description Underflow is possible here.

Listing 8:

```
117 sum = za + ya - zxa;
```

3.9 CVF-9

- **Severity** Moderate
- **Status** Info
- **Category** Overflow/Underflow
- **Source** YieldMath.sol

Description Overflow is possible when converting "sum" to "uint128".
Client Comment Duplicate of 92.

Listing 9:

```
123 (fyTokenOut = uint256(fyTokenReserves) - uint256(uint128(sum).
    ↪ pow(ONE, a))) <= MAX,
```

3.10 CVF-10

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

Recommendation The value "1e12" should be a named constant.
Client Comment "error guards" removed, added additional documentation.

Listing 10:

```
127 fyTokenOut = fyTokenOut < MAX - 1e12 ? fyTokenOut + 1e12 : MAX;
    ↪ // Add error guard, ceiling the result at max
```

3.11 CVF-11

- **Severity** Major
- **Status** Fixed
- **Category** Unclear behavior
- **Source** YieldMath.sol

Description Despite the comment, this line not only ceils the result at max, but also increases the result by 1e12.
Recommendation Consider explaining this behavior.
Client Comment "error guards" removed, added additional documentation.

Listing 11:

```
127 fyTokenOut = fyTokenOut < MAX - 1e12 ? fyTokenOut + 1e12 : MAX;
    ↪ // Add error guard, ceiling the result at max
```

3.12 CVF-12

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

Description Brackets around "zaYaYxa" are redundant.
Recommendation Consider removing them.

Listing 12:

```
236 int128(uint128((zaYaYxa).divu(uint128(c.div(mu)))).pow(uint128(
    ↪ ONE), uint128(a)))
```

3.13 CVF-13

- **Severity** Moderate
- **Status** Fixed
- **Category** Overflow/Underflow
- **Source** YieldMath.sol

Description Overflow is possible when converting to "int128".
Recommendation Consider using safe conversion.

Listing 13:

```
236          int128(uint128((zaYaYxa).divu(uint128(c.div(mu)))).pow(
                 ↪ uint128(ONE), uint128(a)))

239 require(rightTerm <= int128(sharesReserves), "YieldMath: Rate
    ↪ underflow");

431       int128(uint128(zaYaYxa.divu(uint128(c.div(mu)))).pow(uint128
              ↪ (ONE), uint128(a)))

434 require(subtotal >= int128(sharesReserves), "YieldMath: Rate
    ↪ underflow");
```

3.14 CVF-14

- **Severity** Major
- **Status** Fixed
- **Category** Unclear behavior
- **Source** YieldMath.sol

Description Despite the comment, this line not only floors the result at zero, but also decreases the result by 1e12.
Recommendation Consider explaining this behavior.
Client Comment "error guards" removed,added additional documentation.

Listing 14:

```
340 result = result > 1e12 ? result - 1e12 : 0; // Subtract error
    ↪ guard, flooring the result at zero
```

3.15 CVF-15

- **Severity** Moderate
- **Status** Fixed
- **Category** Overflow/Underflow
- **Source** YieldMath.sol

Description Underflow is possible here.
Recommendation Consider adding an appropriate underflow check.

Listing 15:

```
425 uint256 yxa = (fyTokenReserves - fyTokenOut).pow(a, ONE);
```

```
436 return uint128(subtotal) - sharesReserves;
```

3.16 CVF-16

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

Description The body of this block is not properly indented.
Recommendation Consider indenting.

Listing 16:

```
473 unchecked {
```

3.17 CVF-17

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

Recommendation The value "1e18" should be a named constant.
Client Comment No thank you, 1e18 is recognizable enough.

Listing 17:

```
484 uint256 result_ = uint256(uint128(sum).pow(ONE, uint128(a))) * 1
    ↪ e18 / totalSupply;
```

3.18 CVF-18

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

Recommendation Events are usually named via nouns, such as "Fees".
Client Comment No thank you. Never seen that using nouns before, actually.

Listing 18:

```
9  event FeesSet(uint16 g1Fee);
```

3.19 CVF-19

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

Description The names of these events looks odd.
Recommendation Consider using more conventional names.
Client Comment No thank you. These events are described in pool life cycle haiku in Pool.sol.

Listing 19:

```
12 event gg();

15 event gm();
```

3.20 CVF-20

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

Recommendation The maturity parameter should probably be indexed as it is used to identify asssets involved.
Client Comment No thank you. The maturity is the same for all Trade events emitted by this contract.

Listing 20:

```
19     uint32 maturity,

36 event Trade(uint32 maturity, address indexed from, address
   ↪ indexed to, int256 base, int256 fyTokens);
```

3.21 CVF-21

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

Description The comment about "fyYVDAI" is confusing. It is unclear why one could suggest that it is "fyYVDAI".
Recommendation Consider removing the comment or clarifying it.

Listing 21:

```
83 /// The fyToken for the corresponding base token. It's not
     ↪ fyYVDAI, it's still fyDAI. Even though we convert base
```

3.22 CVF-22

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

Recommendation "Whent" -> "When".

Listing 22:

```
89 /// Whent these are deposited into a tokenized vault they become
     ↪  shares.
```

3.23 CVF-23

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

Description In ERC-20 the "decimals" property is used by UI to render token amounts in a human-readable way. Using this property in smart contracts is discouraged.
Recommendation Consider treating all token amounts as integers.
Client Comment We use them to calculate prices (because we need to know how much is one unit), and to calculate the scalingFactor.
On the prices Mikhail is right, and is the same idea behind that article I wrote about oracle prices. Unfortunately, the yieldspace-tv formulas use prices, and not integer amounts, so no chance.
On using decimals to calculate the scalingFactor, it's actually true that we are making a minor mistake, because there we are assuming that 1 unit == $1, and scaling our inputs to YieldMath so that the losses due to precision stay in a certain range. However, with ETH being $1000, we are allowing x1000 the precision losses in monetary terms. Fine for this release, but we should fix it in a later release. Either we take the scalingFactor as a constructor parameter, which is error-prone, or we swap to solmate pow and do full-precision math, removing the scalingFactor altogether.

Listing 23:

```
94  uint256 public immutable baseDecimals;

182     baseDecimals = baseToken_.decimals();
```

3.24 CVF-24

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

Recommendation As "IERC4626" extends "IERC20", it would be more logical to declare this variable as "IERC4626", so it could be used as both, "IERC20" and "IERC4626" without casting.
Client Comment No thank you. We feel it would be more confusing to use IERC4626 with non-4626 compliant tokens as is the case with all of our currently deployed pools (PoolEuler.sol / PoolNonTv.sol).

Listing 24:

```
97  /// @dev For most of this contract, only the ERC20 functionality
        ↪ of the shares token is required. As such, shares
    /// are cast as "IERC20Like" and when that 4626 functionality is
        ↪ needed, they are recast as IERC4626.

100 IERC20Like public immutable sharesToken;
```

3.25 CVF-25

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** Pool.sol

Recommendation "wei" -> "way".
Client Comment No thank you. This is the wei.

Listing 25:

```
99 /// This wei, modules for non-4626 compliant base tokens can
   ↪ import this contract and override 4626 specific fn's.
```

3.26 CVF-26

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

Description This comment suggest that the "ts" value is always 1 / seconds in 10 years, while actually it is a constructor argument.
Recommendation Consider rephrasing the comment.

Listing 26:

```
102 /// Time stretch == 1 / seconds in 10 years (64.64)
    int128 public immutable ts;
```

3.27 CVF-27

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** Pool.sol

Description These "not 64.64" remarks are confusing, as they give no clue about the actual number format of the variables.
Recommendation Consider explaining the actual number format.
Client Comment No thank you. This presents a clear difference between the 64bit and non-64bit numbers which many non-mathemeticians struggle with upon initial review.

Listing 27:

```
108 /// Pool's maturity date (not 64.64)

111 /// Used to scale up to 18 decimals (not 64.64)
```

3.28 CVF-28

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

Recommendation The type of this argument should be "IERC20".
Client Comment No thanks, we use the address directly in the constructor.

Listing 28:

```
152 address sharesToken_, //     address of shares token
```

3.29 CVF-29

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

Recommendation The type of this argument should be "IFYToken".
Client Comment No thanks, we use the address directly in the constructor.

Listing 29:

```
153 address fyToken_, //  address of fyToken
```

3.30 CVF-30

- **Severity** Major
- **Status** Info
- **Category** Unclear behavior
- **Source** Pool.sol

Description There is no range check for this argument.
Recommendation Consider adding appropriate checks.
Client Comment No thank you. We will include such checks off-chain as part of the deployment testing harness.

Listing 30:

```
154 int128 ts_, //         time stretch (64.64)
```

3.31 CVF-31

- **Severity** Moderate
- **Status** Fixed
- **Category** Unclear behavior
- **Source** Pool.sol

Description Here a "uint32" value is checked to be greater than the "uint32" maximum value, which could never be true.
Recommendation Consider using a safe conversion function instead.

Listing 31:

```
168 if ((maturity = uint32(IFYToken(fyToken_).maturity())) > type(
    ↪ uint32).max) revert MaturityOverflow();
```

3.32 CVF-32

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

Recommendation The value "10000" should be a named constant.
Client Comment No thank you, this is by design.

Listing 32:

```
196 if ((g1Fee = g1Fee_) > 10000) revert InvalidFee(g1Fee_);

1236 return uint256(g1Fee_).fromUInt().div(uint256(10000).fromUInt())
    ↪ ;

1252 return int128(YieldMath.ONE).div(uint256(g1Fee_).fromUInt().div(
    ↪ uint256(10000).fromUInt()));

1385 if (g1Fee_ > 10000) {
```

3.33 CVF-33

- **Severity** Moderate
- **Status** Fixed
- **Category** Unclear behavior
- **Source** Pool.sol

Description The returned value is ignored.
Recommendation Consider reverting in case false was returned.

Listing 33:

```
203 baseToken_.approve(sharesToken_, type(uint256).max);
```

3.34 CVF-34

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

Description A user may want to set the maximum remainder amount.
Recommendation Consider adding such argument.
Client Comment Acknowledged. We will discuss this internally and consider updating in the future.

Listing 34:

```
248 /// @param remainder Wallet receiving any surplus base.

289     address remainder,

339     address remainder,

365 /// @param remainder Wallet receiving any surplus base.
```

3.35 CVF-35

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

Description It is unclear why this limit is needed. Usually, the more shares a user will bet for his fyToken, the better.
Recommendation Consider removing this limit.
Client Comment Acknowledged. We will discuss this internally and consider updating in the future.

Listing 35:

```
250 /// @param maxRatio Maximum ratio of shares to fyToken in the
    ↪ pool (fp18).

342     uint256 maxRatio

368 /// @param maxRatio Maximum ratio of shares to fyToken in the
    ↪ pool (fp18).
```

3.36 CVF-36

- **Severity** Major
- **Status** Info
- **Category** Documentation
- **Source** Pool.sol

Description Actually, the amount of base found and the amount of base used for the mint could be different, as there could be remainder returned to the caller. Actually, the found amount is returned while the used amount would probably be more interesting.
Recommendation Consider returning the actually used amount and rephrasing the comment.
Client Comment No thank you. It would require extra gas and it was requested by the front end team. We may address it later.

Listing 36:

```
251 /// @return baseIn The amount of base found in the contract that
    ↪  was used for the mint.

284 /// @return baseIn The amount of base found that was used for
    ↪ the mint.

334 /// @return baseIn The amount of base found that was used for
    ↪ the mint.

369 /// @return baseIn The amount of base found that was used for
    ↪ the mint.
```

3.37 CVF-37

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

Description A user may want to limit the minimum amount of liquidity minted.
Recommendation Consider adding such argument.
Client Comment Acknowledged. We will discuss this internally and consider updating in the future.

Listing 37:

```
253 /// @return lpTokensMinted The amount of LP tokens minted.

350         uint256 lpTokensMinted

383         uint256 lpTokensMinted
```

3.38 CVF-38

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

Description Not only "supply > 0" check is skipped, but also "supply == 0" check is added.
Recommendation Consider mentioning this fact in the comment.

Listing 38:

```
277 /// @dev This is the exact same as mint() but with auth added
    ↪ and skip the supply > 0 check.
```

3.39 CVF-39

- **Severity** Major
- **Status** Fixed
- **Category** Unclear behavior
- **Source** Pool.sol

Description These arguments doesn't make sense for the initial minting.
Recommendation Consider removing them.

Listing 39:

```
290 uint256 minRatio,
    uint256 maxRatio
```

3.40 CVF-40

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

Description In case "realFYTokenCached_" is zero, "minRatio" and "maxRatio" are ignored, while such situation could be thought as an infinity ratio. It seems weird, that "maxRatio" set to a finite value will effectively forbid finite ratios above this value, but will not forbid infinite ratio.
Recommendation Consider requiring "maxRatio" to be "infinite", when "realFYTokenCached" is zero. "Infinite" could be coded as type(uint256).max.

Listing 40:

```
398 if (realFYTokenCached_ != 0) {

575 if (realFYTokenCached_ != 0) {
```

3.41 CVF-41

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Pool.sol

Description The expression "uint256(cache.sharesCached).wdiv(realFYTokenCached_)" is calculated twice.
Recommendation Consider calculating once and reusing.
Client Comment Acknowledged. We will discuss this internally and consider updating in the future.

Listing 41:

```
400     uint256(cache.sharesCached).wdiv(realFYTokenCached_) <
            ↪ minRatio ||
        uint256(cache.sharesCached).wdiv(realFYTokenCached_) >
            ↪ maxRatio
) revert SlippageDuringMint((uint256(cache.sharesCached) * 1e18)
    ↪ / realFYTokenCached_, minRatio, maxRatio);
```

3.42 CVF-42

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

Description The "wdiv" value is calculated with rounding down, so the actual returned value could be less than the precise value. Thus, if the returned value equals to "maxRatio", the precise value could be greater than "maxRatio".
Recommendation Consider rounding up.
Client Comment Acknowledged. We will discuss this internally and consider updating in the future.

Listing 42:

```
401 uint256(cache.sharesCached).wdiv(realFYTokenCached_) > maxRatio
```

3.43 CVF-43

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

Description Here, the logic of the "wdiv" function is reimplemented.
Recommendation Consider using the function.

Listing 43:

```
402 ) revert SlippageDuringMint((uint256(cache.sharesCached) * 1e18)
    ↪ / realFYTokenCached_, minRatio, maxRatio);
```

3.44 CVF-44

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

Recommendation The value "1e18" should be a named constant.
Client Comment No thank you.

Listing 44:

```
402 ) revert SlippageDuringMint((uint256(cache.sharesCached) * 1e18)
    ↪ / realFYTokenCached_, minRatio, maxRatio);
```

3.45 CVF-45

- **Severity** Minor
- **Status** Info
- **Category** Overflow/Underflow
- **Source** Pool.sol

Description Phantom overflow is possible here, i.e. a situation when the final calculation result would fit into the destination type, while some intermediary calculation overflows.
Recommendation Consider using the "muldiv" function as described here: https://xn--2-umb.com/21/muldiv/
Client Comment Acknowledged. We will discuss this internally and consider updating in the future.

Listing 45:

```
402     ) revert SlippageDuringMint((uint256(cache.sharesCached) * 1
          ↪ e18) / realFYTokenCached_, minRatio, maxRatio);

415     lpTokensMinted = (supply * sharesIn) / cache.sharesCached;

432     sharesIn = sharesToSell + ((cache.sharesCached +
          ↪ sharesToSell) * lpTokensMinted) / supply;

1170 currentCumulativeRatio_ = cumulativeRatioLast + (fyTokenCached *
       ↪  timeElapsed).rdiv(_mulMu(sharesCached));

1279 return (_getSharesBalance() * _getCurrentSharePrice()) / 10**
       ↪ baseDecimals;
```

3.46 CVF-46

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

Description In case realFYTokenCached_ < fyTokenToBuy, this will revert.
Recommendation Consider handling this case separately.

Listing 46:

```
430 lpTokensMinted = (supply * (fyTokenToBuy + fyTokenIn)) / (
      ↪ realFYTokenCached_ - fyTokenToBuy);
```

3.47 CVF-47

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

Description Here, the number of shares to be taken from the caller is calculating wounded down, i.e. towards the user.
Recommendation Consider calculating rounding up, i.e. towards the pool.
Client Comment Acknowledged. We will discuss this internally and consider updating in the future.

Listing 47:

```
432 sharesIn = sharesToSell + ((cache.sharesCached + sharesToSell) *
    ↪ lpTokensMinted) / supply;
```

3.48 CVF-48

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

Description The caller may want to set the minimum output amounts.
Recommendation Consider adding such arguments.
Client Comment Acknowledged. We will discuss this internally and consider updating in the future.

Listing 48:

```
500         uint256 baseOut,
            uint256 fyTokenOut

535 ) external virtual override returns (uint256 lpTokensBurned,
    ↪ uint256 baseOut) {

560         uint256 baseOut,
            uint256 fyTokenOut
```

3.49 CVF-49

- **Severity** Critical
- **Status** Fixed
- **Category** Flaw
- **Source** Pool.sol

Description Here "baseOut" is guaranteed to be zero.
Recommendation Should be "sharesOut" instead.

Listing 49:

```
591 (cache.sharesCached - baseOut.u128()) * scaleFactor_, //
    ↪ Cache, minus virtual burn
```

3.50 CVF-50

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Pool.sol

Description Here token amounts in arguments are multiplied by "scaleFactor" and the resulting amount is divided by "scaleFactor". This looks like an unnecessary complication, as in theory this shouldn't affect the final result.
Recommendation Consider removing redundant operations.
Client Comment Acknowledged. We would remove the scale factor if we move to full precision 'pow' later on.

Listing 50:

```
591             (cache.sharesCached - baseOut.u128()) * scaleFactor_, //
                    ↪         Cache, minus virtual burn
                (cache.fyTokenCached - fyTokenOut.u128()) * scaleFactor_
                    ↪ , //  Cache, minus virtual burn
                fyTokenOut.u128() * scaleFactor_, //
                    ↪                                 Sell the virtual fyToken
                    ↪ obtained

600         scaleFactor_;

731         sharesBalance * scaleFactor_,
            fyTokenBalance * scaleFactor_,
            sharesOut * scaleFactor_,

740     scaleFactor_;

845         sharesBalance * scaleFactor_,
            fyTokenBalance * scaleFactor_,
            fyTokenOut * scaleFactor_,

854     scaleFactor_;

945         sharesBalance * scaleFactor_,
            fyTokenBalance * scaleFactor_,
            sharesIn * scaleFactor_,

954     scaleFactor_;

1042        sharesBalance * scaleFactor_,
            fyTokenBalance * scaleFactor_,
            fyTokenIn * scaleFactor_,

1050    ) / scaleFactor_;
```

3.51 CVF-51

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

Description It looks reasonable to use "tradeToBase" after maturity, as there are no fyTokens anymore.
Recommendation Consider just ignoring the "tradeToBase" flag after maturity instead of reverting.
Client Comment Fixed, now we don't allow any minting, or burnForBase, after maturity.

Listing 51:

```
594 maturity — uint32(block.timestamp), //
    ↪ This can't be called after maturity
```

3.52 CVF-52

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

Recommendation Consider adding a comment explaining why "lpTokensBurnoed" is subtracted here, similar to the comment in "_mint".

Listing 52:

```
607 (cache.fyTokenCached — fyTokenOut — lpTokensBurned).u128(),
```

3.53 CVF-53

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

Description If the updated "fyTokenCached" will be less than the totalSupply, his could lock the pool.
Recommendation Consider adding an explicit "require" statement to prevent such situation.

Listing 53:

```
607     (cache.fyTokenCached — fyTokenOut — lpTokensBurned).u128(),

806     cache.fyTokenCached — fyTokenOut,

912 _update(sharesBalance, cache.fyTokenCached — fyTokenOut, cache.
    ↪ sharesCached, cache.fyTokenCached);
```

3.54 CVF-54

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

Description it is unclear what "correct" means here.
Recommendation Consider rephrasing.

Listing 54:

```
669 /// The trader needs to have transferred in the correct amount
    ↪ of fyTokens in advance.
```

3.55 CVF-55

- **Severity** Moderate
- **Status** Info
- **Category** Unclear behavior
- **Source** Pool.sol

Description Both, "wrap" and "unwrap" could do rounding and most probably they round towards the wrapper contract, i.e. agains the pool. So unwrap(wrap(x)) could be less than x, thus the caller may actually receive less than "baseOut" base tokens.
Recommendation Consider doing "sharesOut = sharesToken.withdraw(baseOut, to, this)" at the beginning of the function to send exactly "baseOut" base tokens and calcualte the amount of shares burned.
Client Comment We acknowledge this. It is not as easy as using previewWithdraw since none of the current vault tokens support this. We are designing a solution around storing a "buffer" of base tokens in the contract to make up for the very small (<0.01%) difference but we will not be implementing that solution before we deploy the pool.

Listing 55:

```
683 uint128 sharesOut = _wrapPreview(baseOut).u128();

701 _unwrap(to);
```

3.56 CVF-56

- **Severity** Major
- **Status** Info
- **Category** Unclear behavior
- **Source** Pool.sol

Description This transition from base to shares potentially rounds down, thus the resulting amount of shares could be not enough to redeem the "baseOut" amount of base tokens.
Recommendation Consider using "previewWithdraw" instead of "previewDeposit".
Client Comment We acknowledge this. It is not as easy as using previewWithdraw since none of the current vault tokens support that fn. We are designing a solution around storing a "buffer" of base tokens in the contract to make up for the very small (<0.01%) difference but we will not be implementing that solution before we deploy the pool.

Listing 56:

```
713 _wrapPreview(baseOut).u128(),
```

3.57 CVF-57

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

Description This argument is redundant, as a user implicitly specifies the maximum base amount to be spent, by sending this amount to the contract before the call.
Recommendation Consider removing this argument.
Client Comment Fixed, removed logic and the related custom errors. Left the param in for backwards compatability and a natspec saying it does nothing.

Listing 57:

```
777 /// @param max Maximum amount of base token that will be paid
    ↪ for the trade.
```

3.58 CVF-58

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

Description This function doesn't return the unused base tokens, so they remain unaccounted and could be taken by anybody.
Recommendation Consider returning them to the caller.
Client Comment Acknowledged. This is known. It generally will be dust amounts. Furthermore, updating this would break backwards compatibility.

Listing 58:

```
779 function buyFYToken(
```

3.59 CVF-59

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

Description The purpose of this check is unclear.
Recommendation Consider adding a comment.
Client Comment Fixed. Updated error name (the comment was already clear).

Listing 59:

```
856 uint128 newSharesMulMu = _mulMu(sharesBalance + sharesIn).u128()
    ↪ ;
if ((fyTokenBalance - fyTokenOut) < newSharesMulMu) {
    revert InsufficientFYTokenBalance(fyTokenBalance -
        ↪ fyTokenOut, newSharesMulMu);
}

956 uint128 newSharesMulMu = _mulMu(sharesBalance + sharesIn).u128()
    ↪ ;
if ((fyTokenBalance - fyTokenOut) < newSharesMulMu) {
    revert InsufficientFYTokenBalance(fyTokenBalance -
        ↪ fyTokenOut, newSharesMulMu);
}
```

3.60 CVF-60

- **Severity** Major
- **Status** Fixed
- **Category** Suboptimal
- **Source** Pool.sol

Description This call is redundant, it is needed only to check agains "min".
Recommendation Just move the check after the "_upwrap" call and use the value returned by "_unwrap".

Listing 60:

```
1009 baseOut = _unwrapPreview(sharesOut).u128();
```

3.61 CVF-61

- **Severity** Moderate
- **Status** Info
- **Category** Unclear behavior
- **Source** Pool.sol

Description Here "fyTokenCached_" is multiplied by "timeElapsed" and the product is calculated as a "uint104" value, then converted to "uint256". Thus, in case the product doesn't fit into "uint104" the transaction will be reverted.
Recommendation Consider converting "fyTokenCached_" to "uint256" first, and then multiplying by "timeElapsed".
Client Comment Acknowledged. We would like to rewrite the contract to do a better job with types of less than 256 bits. Basically we want to upcast whenever we load from storage and then downcast (safely) before we store new values.But we will not get this done in time for deployment this quarter.

Listing 61:

```
1209 newCumulativeRatioLast += uint256(fyTokenCached_ * timeElapsed).
     ↪ rdiv(_mulMu(sharesCached_));
```

3.62 CVF-62

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Pool.sol

Description There will be four SSTORE operations into the same slot.
Recommendation Consider refactoring to perform only one SSTORE.
Client Comment Acknowledged. We will discuss this internally and consider updating in the future.

Listing 62:

```
1212 blockTimestampLast = blockTimestamp;
     cumulativeRatioLast = newCumulativeRatioLast;

1218 sharesCached = newSharesCached;
     fyTokenCached = newFYTokenCached;
```

3.63 CVF-63

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

Recommendation This could be simplified using the "divu" function: return uint256(g1Fee_).divu(10000);

Listing 63:

```
1236 return uint256(g1Fee_).fromUInt().div(uint256(10000).fromUInt())
     ↪ ;
```

3.64 CVF-64

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

Recommendation The value "uint256(10000).fromUInt()" could be precomputed.

Listing 64:

```
1236 return uint256(g1Fee_).fromUInt().div(uint256(10000).fromUInt())
     ↪ ;
```

3.65 CVF-65

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

Recommendation This could be simplified using the "divu" function: return uint256(10000).divu(g1Fee_);

Listing 65:

```
1252 return int128(YieldMath.ONE).div(uint256(g1Fee_).fromUInt().div(
     ↪ uint256(10000).fromUInt()));
```

3.66 CVF-66

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Pool.sol

Description The expression "10**baseDecimals" is calculated every time.
Recommendation Consider calculating once and storing in an immutable variable.

Listing 66:

```
1279 return (_getSharesBalance() * _getCurrentSharePrice()) / 10**
     ↪ baseDecimals;

1293 uint256 scalar = 10**baseDecimals;
```

3.67 CVF-67

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** Pool.sol

Recommendation This could be simplified using the "divu" function: return (_getCurrentSharePrice() * scaleFactor).divu(1e18);
Client Comment Some small gas inefficiency is worth the trade off of introducing some relatively significant complexity.

Listing 67:

```
1306 return ((_getCurrentSharePrice() * scaleFactor)).fromUInt().div(
     ↪ uint256(1e18).fromUInt());
```

3.68 CVF-68

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

Description The returned values are referred in the comment by names, while actually they don't have names.
Recommendation Consider naming them.
Client Comment Acknowledged. We will discuss this internally and consider updating in the future.

Listing 68:

```
1310 /// @return Cached shares token balance.
     /// @return Cached virtual FY token balance which is the actual
        ↪ balance plus the pool token supply.
     /// @return Timestamp that balances were last cached.
     /// @return g1Fee  This is a fp4 number where 10_000 is 1.

1319             uint104,
1320             uint104,
                 uint32,
                 uint16
```

3.69 CVF-69

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

Recommendation This could be simplified using the "mulu" function: product = mu.mulu (amount);

Listing 69:

```
1359 product = (amount * (mu.mul(uint256(1e18).fromUInt()).toUInt()))
        ↪  / 1e18;
```

3.70 CVF-70

- **Severity** Critical
- **Status** Fixed
- **Category** Flaw
- **Source** Pool.sol

Description The surplus shares balance is calculated, but base tokens are attempted to be transferred without unwrapping. This makes the function useless.

Listing 70:

```
1367 retrieved = _getSharesBalance() - sharesCached; // Cache can
       ↪ never be above balances
     baseToken.safeTransfer(to, retrieved);
```

3.71 CVF-71

- **Severity** Minor
- **Status** Info
- **Category** Readability
- **Source** Pool.sol

Description This double conversion looks ugly.
Recommendation Consider using "IERC20" instead of "IERC20Like" for simplicity and readability.
Client Comment Both baseToken and sharesToken are of the type IERC20Like which may introduce unnecessary complexity so it or may not make sense to remove IERC20Like entirely.

Listing 71:

```
1397 return IERC20(address(baseToken));
```

3.72 CVF-72

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

Recommendation The types of these arguments could be more specific.

Listing 72:

```
38 address base_,
   address fyToken_,
```

3.73 CVF-73

- **Severity** Minor
- **Status** Info
- **Category** Overflow/Underflow
- **Source** PoolYearnVault.sol

Description Phantom overflow is possible here.
Recommendation Consider using the "muldiv" function as described here: https://xn–2-umb.com/21/muldiv/
Client Comment Acknowledged. We will discuss this internally and consider updating in the future.

Listing 73:

```
69 shares = base_ * 10**baseDecimals / _getCurrentSharePrice();

86 base_ = shares * _getCurrentSharePrice() / 10**baseDecimals;
```

3.74 CVF-74

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** PoolYearnVault.sol

Description The expression "10**baseDecimals" is calculated every time.
Recommendation Consider calculating once and storing in an immutable variable.
Client Comment Acknowledged. We will discuss this internally and consider updating in the future.

Listing 74:

```
69 shares = base_ * 10**baseDecimals / _getCurrentSharePrice();

86 base_ = shares * _getCurrentSharePrice() / 10**baseDecimals;
```

3.75 CVF-75

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

Description We didn't review these files.
Client Comment Out of scope.

Listing 75:

```
7  import {CastU256U128} from "@yield-protocol/utils-v2/contracts/
     ↪ cast/CastU256U128.sol";
   import {CastU256U104} from "@yield-protocol/utils-v2/contracts/
     ↪ cast/CastU256U104.sol";
   import {CastU256I256} from "@yield-protocol/utils-v2/contracts/
     ↪ cast/CastU256I256.sol";
10 import {CastU128U104} from "@yield-protocol/utils-v2/contracts/
     ↪ cast/CastU128U104.sol";
   import {CastU128I128} from "@yield-protocol/utils-v2/contracts/
     ↪ cast/CastU128I128.sol";

13 import {Exp64x64} from "../Exp64x64.sol";

16 import {WDiv} from "@yield-protocol/utils-v2/contracts/math/WDiv
     ↪ .sol";
   import {RDiv} from "@yield-protocol/utils-v2/contracts/math/RDiv
     ↪ .sol";

22 import {ERC20Permit} from "@yield-protocol/utils-v2/contracts/
     ↪ token/ERC20Permit.sol";
   import {AccessControl} from "@yield-protocol/utils-v2/contracts
     ↪ /access/AccessControl.sol";
   import {ERC20, IERC20Metadata as IERC20Like, IERC20} from "
     ↪ @yield-protocol/utils-v2/contracts/token/ERC20.sol";
   import {MinimalTransferHelper} from "@yield-protocol/utils-v2/
     ↪ contracts/token/MinimalTransferHelper.sol";
```

3.76 CVF-76

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

Recommendation The argument types could be more specific.

Listing 76:

```
38 address euler_, // The main Euler contract address
   address eToken_,
40 address fyToken_,
```

3.77 CVF-77

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

Recommendation Consider putting this comment into the empty function body.
Client Comment No thank you.

Listing 77:

```
48 /// **This function is intentionally empty to overwrite the Pool
   ↪ ._approveSharesToken fn.**
   /// This is normally used by Pool.constructor give max approval
   ↪ to sharesToken, but Euler tokens require approval
50 /// of the main Euler contract — not of the individual
   ↪ sharesToken contracts. The required approval is given
   ↪ above
   /// in the constructor.
```

3.78 CVF-78

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

Recommendation The "1e18" value should be a named constant.
Client Comment No thank you.

Listing 78:

```
65 return IEToken(address(sharesToken)).convertBalanceToUnderlying
   ↪ (1e18);
```

3.79 CVF-79

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** PoolEuler.sol

Description The decimals adjustment is only done one way: the case when base token has more than 18 decimals is not supported.
Recommendation Consider supporting this case or just don't adjust decimals at all.
Client Comment Disagree, there are a number of major tokens with less than 18 decimals (USDC, WBTC), while it is rare for a token to have more than 18 (YAM).

Listing 79:

```
69  /// The decimals of the shares amount returned is adjusted to
      ↪ match the decimals of the baseToken

71      return (sharesToken.balanceOf(address(this)) / scaleFactor).
          ↪ u104();

90  /// @return shares The amount of shares that would be returned
      ↪ from depositing (converted to base decimals).

92      shares = IEToken(address(sharesToken)).
          ↪ convertUnderlyingToBalance(assets) / scaleFactor;

113 /// decimals via the overridden _getSharesBalance(). Therefore,
      ↪ this _unwrapPreview() expects to receive share
    /// amounts which have already been converted to base decimals.
      ↪ However, the eToken convertBalanceToUnderlying()
    /// used in this fn requires share amounts in 18 decimals so we
      ↪ scale the shareAmount back up to fp18 and pass

121     assets = IEToken(address(sharesToken)).
          ↪ convertBalanceToUnderlying(sharesInBaseDecimals *
          ↪ scaleFactor);
```

3.80 CVF-80

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

Description We didn't review these files.
Client Comment Ok.

Listing 80:

```
3 import "@yield-protocol/utils-v2/contracts/token/IERC20Metadata.
    ↪ sol";
import "@yield-protocol/utils-v2/contracts/token/IERC20.sol";
```

3.81 CVF-81

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** IYVToken.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 Acknowledged. We will discuss this internally and consider updating in the future.

Listing 81:

```
10 function deposit(uint256 _amount, address _recipient) external
    ↪ returns (uint256);

21 function withdraw(uint256 _amount, address _recipient) external
    ↪ returns (uint256);
```

3.82 CVF-82

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** IYVToken.sol

Description The number format of the returned value is unclear.
Recommendation Consider documenting.
Client Comment Acknowledged. We will discuss this internally and consider updating in the future.

Listing 82:

```
14 function pricePerShare() external view returns (uint256);
```

3.83 CVF-83

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

Recommendation The returned type should be more specific.
Client Comment Acknowledged. We will discuss this internally and consider updating in the future.

Listing 83:

```
18 function token() external view returns (address);
```

3.84 CVF-84

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

Recommendation Should be "^0.8.0".

Listing 84:

```
2 pragma solidity >= 0.8.0;
```

3.85 CVF-85

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

Description We didn't review these files.
Client Comment Ok.

Listing 85:

```
3 import "@yield-protocol/utils-v2/contracts/token/IERC20.sol";
  import "@yield-protocol/utils-v2/contracts/token/IERC2612.sol";

6 import {IERC20Metadata} from "@yield-protocol/utils-v2/
    ↪ contracts/token/ERC20.sol";
```

3.86 CVF-86

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

Description The semantics of the returned values is unclear.
Recommendation Consider giving them descriptive names and/or adding documentation comments.
Client Comment Acknowledged. We will discuss this internally and consider updating in the future.

Listing 86:

```
11 function burn(address baseTo, address fyTokenTo, uint256
     ↪ minRatio, uint256 maxRatio) external returns (uint256,
     ↪ uint256, uint256);
   function burnForBase(address to, uint256 minRatio, uint256
     ↪ maxRatio) external returns (uint256, uint256);
   function buyBase(address to, uint128 baseOut, uint128 max)
     ↪ external returns(uint128);
   function buyBasePreview(uint128 baseOut) external view returns(
     ↪ uint128);
   function buyFYToken(address to, uint128 fyTokenOut, uint128 max)
     ↪  external returns(uint128);
   function buyFYTokenPreview(uint128 fyTokenOut) external view
     ↪ returns(uint128);

28 function init(address to, address remainder, uint256 minRatio,
     ↪ uint256 maxRatio) external returns (uint256, uint256,
     ↪ uint256);

30 function mint(address to, address remainder, uint256 minRatio,
     ↪ uint256 maxRatio) external returns (uint256, uint256,
     ↪ uint256);

32 function mintWithBase(address to, address remainder, uint256
     ↪ fyTokenToBuy, uint256 minRatio, uint256 maxRatio) external
     ↪  returns (uint256, uint256, uint256);

36 function sellBase(address to, uint128 min) external returns(
     ↪ uint128);
   function sellBasePreview(uint128 baseIn) external view returns(
     ↪ uint128);
   function sellFYToken(address to, uint128 min) external returns(
     ↪ uint128);
   function sellFYTokenPreview(uint128 fyTokenIn) external view
     ↪ returns(uint128);

41 function ts() external view returns(int128);
```

3.87 CVF-87

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** IPool.sol

Description The number format of the returned values is unclear.
Recommendation Consider documenting.

Listing 87:

```
17 function currentCumulativeRatio() external view returns (uint256
     ↪ currentCumulativeRatio_, uint256 blockTimestampCurrent);
   function cumulativeRatioLast() external view returns (uint256);

20 function g1() external view returns(int128);
   function g2() external view returns(int128);
   function getC() external view returns (int128);
   function getCurrentSharePrice() external view returns (uint256);

31 function mu() external returns (int128);

35 function scaleFactor() external view returns(uint96);

41 function ts() external view returns(int128);
```

3.88 CVF-88

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

Description The number format of "g1Fee" is unclear.
Recommendation Consider documenting.
Client Comment It's in the natspec: "This is a fp4 number where 10_000 is 1."

Listing 88:

```
24 function getCache() external view returns (uint104 baseCached,
     ↪ uint104 fyTokenCached, uint32 blockTimestampLast, uint16
     ↪ g1Fee_);

40 function setFees(uint16 g1Fee_) external;
```

3.89 CVF-89

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

Description We didn't review these files.

Listing 89:

```
3 import "@yield-protocol/utils-v2/contracts/token/IERC20Metadata.
    ↪ sol";
import "@yield-protocol/utils-v2/contracts/token/IERC20.sol";
```

3.90 CVF-90

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

Description The "subAccountId" argument is "uint", but in the comments only values 0..255 are described. Does this mean that values above 255 are invalid?
Recommendation Consider clarifying.
Client Comment Acknowledged. We will discuss this internally and consider updating in the future.

Listing 90:

```
20 /// @param subAccountId 0 for primary, 1-255 for a sub-account.

28 /// @param subAccountId 0 for primary, 1-255 for a sub-account
```

3.91 CVF-91

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

Recommendation This function should return the number of eTokens issued.
Client Comment Acknowledged. We will discuss this internally and consider updating in the future.

Listing 91:

```
23 function deposit(uint subAccountId, uint amount) external;
```

3.92 CVF-92

- **Severity** Major
- **Status** Info
- **Category** Unclear behavior
- **Source** IEToken.sol

Recommendation This function should accept an additional argument to specify the minimum amount of eTokens to be issued.
Client Comment Out of scope.

Listing 92:

```
23 function deposit(uint subAccountId, uint amount) external;
```

3.93 CVF-93

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

Recommendation The return type of this function could be more specific.
Client Comment Acknowledged. We will discuss this internally and consider updating in the future.

Listing 93:

```
25 function underlyingAsset() external view returns (address);
```

3.94 CVF-94

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

Recommendation This function should return the number of eTokens burned.
Client Comment Acknowledged. We will discuss this internally and consider updating in the future.

Listing 94:

```
30 function withdraw(uint subAccountId, uint amount) external;
```

3.95 CVF-95

- **Severity** Major
- **Status** Info
- **Category** Flaw
- **Source** IEToken.sol

Recommendation This function should accept an additional argument to specify the maximum amount of eTokens to be burned.
Client Comment Out of scope.

Listing 95:

```
30 function withdraw(uint subAccountId, uint amount) external;
```

3.96 CVF-96

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** IERC4626.sol

Description We didn't review these files.
Client Comment Ok.

Listing 96:

```
3 import "@yield-protocol/utils-v2/contracts/token/IERC20Metadata.
    ↪ sol";
import "@yield-protocol/utils-v2/contracts/token/IERC20.sol";
```

3.97 CVF-97

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IERC4626.sol

Description This interface doesn't declare all the functions specified in ERC-4626.
Recommendation Consider renaming the interface to "IERC4626Like".all declaring all the ERC-4626 functions. The interface also doesn't declare ERC-4626 events. Consider declaring them.
Client Comment No thank you.

Listing 97:

```
6 interface IERC4626 is IERC20, IERC20Metadata {
```

3.98 CVF-98

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IERC4626.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 Acknowledged. We will discuss this internally and consider updating in the future.

Listing 98:

```
7 function asset() external returns (IERC20);
  function convertToAssets(uint256 shares) external view returns (
    ↪ uint256);
  function convertToShares(uint256 assets) external view returns (
    ↪ uint256);
```

3.99 CVF-99

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** IERC20Like.sol

Description We didn't review these files.
Client Comment Ok.

Listing 99:

```
3 import "@yield-protocol/utils-v2/contracts/token/IERC20Metadata.
    ↪ sol";
  import "@yield-protocol/utils-v2/contracts/token/IERC20.sol";
```

3.100 CVF-100

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IERC20Like.sol

Description This function is not a part of the ERC-20 standard.
Recommendation Consider renaming the interface to "IERC20Mintable" or something like this.

Listing 100:

```
7 function mint(address receiver, uint256 shares) external;
```

3.101 CVF-101

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** IERC20Like.sol

Description The argument name ‘shares” looks odd, as in ERC-20 asset units are usually called “tokens”.
Recommendation Consider renaming the argument to “value” or “amount”.

Listing 101:

```
7 function mint(address receiver, uint256 shares) external;
```