ABDK CONSULTING

SMART CONTRACT AUDIT

AAVE

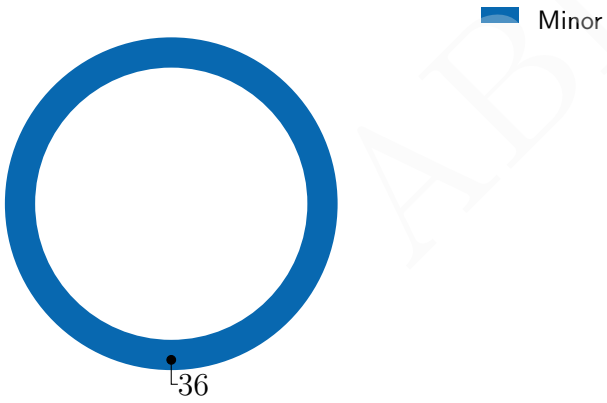
ParaswapAdapter

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
12th May 2021

We've been asked to review Aave smart contracts related to the Paraswap Adapter, given in a Github repo.
At some point we were also given the business logic. We have found only minor issues.



Findings

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

ID	Severity	Category	Status
CVF-28	Minor	Procedural	Opened
CVF-29	Minor	Bad naming	Opened
CVF-30	Minor	Bad datatype	Opened
CVF-31	Minor	Bad datatype	Opened
CVF-32	Minor	Bad datatype	Opened
CVF-33	Minor	Procedural	Opened
CVF-34	Minor	Flaw	Opened
CVF-35	Minor	Suboptimal	Opened
CVF-36	Minor	Suboptimal	Opened

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	May 10, 2021	D. Khovratovich	Initial Draft
0.2	May 10, 2021	D. Khovratovich	Minor revision
1.0	May 11, 2021	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 audited the Aave Github repository at pull request 55. Concretely, the following files were audited:

- BaseParaSwapAdapter.sol;
- BaseParaSwapSellAdapter.sol; ParaSwapLiquiditySwapAdapter.sol.

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** Opened
- **Category** Suboptimal
- **Source** ParaSwapLiquiditySwapAdapter.sol

Recommendation Should be "ô.6.0" according to a common best practice.

Listing 1:

```
2 solidity 0.6.12;
```

3.2 CVF-2

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** ParaSwapLiquiditySwapAdapter.sol

Description We didn't review this file.

Listing 2:

```
6 {ILendingPoolAddressesProvider} from '../interfaces/
    ↪ ILendingPoolAddressesProvider.sol';
```

3.3 CVF-3

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** ParaSwapLiquiditySwapAdapter.sol

Description It is unclear what version of the file is used.

Listing 3:

```
7 {IERC20} from '../dependencies/openzeppelin/contracts/IERC20.sol
    ↪ ';
```

3.4 CVF-4

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** ParaSwapLiquiditySwapAdapter.sol

Recommendation It is a good practice to put a comment into an empty block to explain why it is empty.

Listing 4:

```
17 ) public BaseParaSwapSellAdapter(addressesProvider) {}
```

3.5 CVF-5

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

Recommendation Probably the length of these arrays should be checked as well.

Listing 5:

```
45 uint256 flashLoanAmount = amounts[0];
   uint256 premium = premiums[0];
```

3.6 CVF-6

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** ParaSwapLiquiditySwapAdapter.sol

Recommendation These arguments should have type "IERC20".

Listing 6:

```
88  address assetToSwapFrom,
    address assetToSwapTo,

149 address assetToSwapFrom,
150 address assetToSwapTo,
```

3.7 CVF-7

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** ParaSwapLiquiditySwapAdapter.sol

Recommendation The type of the "aToken" variable should be "IERC20".

Listing 7:

```
97  address aToken = _getReserveData(assetToSwapFrom).aTokenAddress;

153 address aToken = _getReserveData(assetToSwapFrom).aTokenAddress;
```

3.8 CVF-8

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

Recommendation Two calls look like waste of gas.

Listing 8:

```
123 IERC20(assetToSwapTo).safeApprove(address(LENDING_POOL), 0);
    IERC20(assetToSwapTo).safeApprove(address(LENDING_POOL),
    ↪ amountReceived);

175 IERC20(assetToSwapTo).safeApprove(address(LENDING_POOL), 0);
    IERC20(assetToSwapTo).safeApprove(address(LENDING_POOL),
    ↪ amountReceived);

188 IERC20(assetToSwapFrom).safeApprove(address(LENDING_POOL), 0);
    IERC20(assetToSwapFrom).safeApprove(address(LENDING_POOL),
    ↪ flashLoanAmount.add(premium));
```

3.9 CVF-9

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** ParaSwapLiquiditySwapAdapter.sol

Description We didn't review the "deposit" function.

Listing 9:

```
125 LENDING_POOL.deposit(assetToSwapTo, amountReceived, msg.sender,
    ↪ 0);
```

3.10 CVF-10

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

Description The expression "amountToSwap.add(premium)" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 10:

```
162 require(balance >= amountToSwap.add(premium), '
    ↪ INSUFFICIENT_ATOKEN_BALANCE');

183 amountToSwap.add(premium),
```

3.11 CVF-11

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

Recommendation These lines should probably be moved to the "executeOperation" function, as it is responsible for repaying the flash loan.

Listing 11:

```
187 // Repay flash loan
IERC20(assetToSwapFrom).safeApprove(address(LENDING_POOL), 0);
IERC20(assetToSwapFrom).safeApprove(address(LENDING_POOL),
    ↪ flashLoanAmount.add(premium));
```

3.12 CVF-12

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

Recommendation Should be "0.6.0" according to a common best practice.

Listing 12:

```
2 solidity 0.6.12;
```

3.13 CVF-13

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** BaseParaSwapSellAdapter.sol

Description We did't review these files.

Listing 13:

```
6 {PercentageMath} from '../protocol/libraries/math/PercentageMath
    ↪ .sol';

8 {ILendingPoolAddressesProvider} from '../interfaces/
    ↪ ILendingPoolAddressesProvider.sol';
```

3.14 CVF-14

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** BaseParaSwapSellAdapter.sol

Description It is unclear what version of this file is used.

Listing 14:

```
9 {IERC20} from '../dependencies/openzeppelin/contracts/IERC20.sol
    ↪ ';
```

3.15 CVF-15

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** BaseParaSwapSellAdapter.sol

Recommendation It is a good practice to put a comment into an empty block to explain why it is empty.

Listing 15:

```
21 ) public BaseParaSwapAdapter(addressesProvider) {
   }
```

3.16 CVF-16

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** BaseParaSwapSellAdapter.sol

Recommendation The type of this argument should be "IParaSwapAugustus".

Listing 16:

```
38 address augustus,
```

3.17 CVF-17

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** BaseParaSwapSellAdapter.sol

Recommendation These arguments should have type "IERC20".

Listing 17:

```
39 address assetToSwapFrom,
40 address assetToSwapTo,
```

3.18 CVF-18

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** BaseParaSwapSellAdapter.sol

Description Phantom overflow is possible here, i.e. a situation when the final result would fit into the destination type, while some intermediary calculations overflow.

Listing 18:

```
53 .mul(fromAssetPrice.mul(10**toAssetDecimals))
   .div(toAssetPrice.mul(10**fromAssetDecimals))
   .percentMul(PercentageMath.PERCENTAGE_FACTOR -
     ↪ MAX_SLIPPAGE_PERCENT);
```

3.19 CVF-19

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

Recommendation Multiplying the numerator by '10̂toAssetDecimals', and the denominator by '10̂fromAssetDecimals' is suboptimal and prone to phantom overflow. Just multiply by '10̂(toAssetDecimals - fromAssetDecimals)' or divide by '10̂(fromAssetDecimals - toAssetDecimals)' depending on whether the 'toAssetDecimals' value is greater than than the 'fromAssetDecimals' or now. In case the 'to' and 'from' assets have the same number of decimals (quite common case) no exponentiation is needed at all.

Listing 19:

```
53 .mul(fromAssetPrice.mul(10**toAssetDecimals))
   .div(toAssetPrice.mul(10**fromAssetDecimals))
```

3.20 CVF-20

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

Description Here a multiplication is performed after a division, which could lead to precision degradation.
Recommendation Consider performing a division only once at the very end of the calculation.

Listing 20:

```
54 .div(toAssetPrice.mul(10**fromAssetDecimals))
   .percentMul(PercentageMath.PERCENTAGE_FACTOR -
     ↪ MAX_SLIPPAGE_PERCENT);
```

3.21 CVF-21

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

Recommendation This check is redundant. IT would be better to let the token smart contract to check this, as there could be exotic contracts that allow transferring more tokens than the sender has.

Listing 21:

```
61 require(balanceBeforeAssetFrom >= amountToSwap, '
    ↪ INSUFFICIENT_BALANCE_BEFORE_SWAP');
```

3.22 CVF-22

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

Recommendation Two calls here look like waste of gas.

Listing 22:

```
65 IERC20(assetToSwapFrom).safeApprove(tokenTransferProxy, 0);
IERC20(assetToSwapFrom).safeApprove(tokenTransferProxy,
    ↪ amountToSwap);
```

3.23 CVF-23

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

Recommendation This code looks redundant, as the caller may do this.

Listing 23:

```
68  if (fromAmountOffset != 0) {
      require(fromAmountOffset >= 4 &&
70      fromAmountOffset <= swapCalldata.length.sub(32),
        'FROM_AMOUNT_OFFSET_OUT_OF_RANGE');
      assembly {
        mstore(add(swapCalldata, add(fromAmountOffset, 32)),
          ↪ amountToSwap)
      }
    }
```

3.24 CVF-24

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

Description It is possible to get the returned data as a "bytes" array in plain solidity. No need for assembly.

Listing 24:

```
76  (bool success,) = augustus.call(swapCalldata);

80      returndatacopy(0, 0, returndatasize())
```

3.25 CVF-25

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

Recommendation Should be "^0.6.0" according to a common best practice.

Listing 25:

```
2  solidity 0.6.12;
```

3.26 CVF-26

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

Recommendation Should be "ô.6.0" according to a common best practice.

Listing 26:

```
2 solidity 0.6.12;
```

3.27 CVF-27

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** BaseParaSwapAdapter.sol

Description It is unclear what version of these files are used.

Listing 27:

```
5 {SafeMath} from '../dependencies/openzeppelin/contracts/SafeMath
    ↪ .sol';
  {IERC20} from '../dependencies/openzeppelin/contracts/IERC20.sol
    ↪ ';
  {IERC20Detailed} from '../dependencies/openzeppelin/contracts/
    ↪ IERC20Detailed.sol';
  {SafeERC20} from '../dependencies/openzeppelin/contracts/
    ↪ SafeERC20.sol';
  {Ownable} from '../dependencies/openzeppelin/contracts/Ownable.
    ↪ sol';
```

3.28 CVF-28

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** BaseParaSwapAdapter.sol

Description We didn't review these files.

Listing 28:

```
10 {ILendingPoolAddressesProvider} from '../interfaces/
   ↪ ILendingPoolAddressesProvider.sol';
{DataTypes} from '../protocol/libraries/types/DataTypes.sol';
{IPriceOracleGetter} from '../interfaces/IPriceOracleGetter.sol
   ↪ ';
{IERC20WithPermit} from '../interfaces/IERC20WithPermit.sol';
{FlashLoanReceiverBase} from '../flashloan/base/
   ↪ FlashLoanReceiverBase.sol';
```

3.29 CVF-29

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** BaseParaSwapAdapter.sol

Recommendation Events are usually named via nouns, such as "Swap".,

Listing 29:

```
38 event Swapped(address indexed fromAsset, address indexed toAsset
   ↪ , uint256 fromAmount, uint256 receivedAmount);
```

3.30 CVF-30

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** BaseParaSwapAdapter.sol

Recommendation The return type of this function should be "uint8" to match the type of the ERC-20 "decimals" property.

Listing 30:

```
59 function _getDecimals(address asset) internal view returns (
   ↪ uint256) {
```

3.31 CVF-31

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** BaseParaSwapAdapter.sol

Recommendation The type of the "asset" argument should be "IERC20Detailed".

Listing 31:

```
59 function _getDecimals(address asset) internal view returns (
   ↪ uint256) {
```

3.32 CVF-32

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** BaseParaSwapAdapter.sol

Recommendation The type of the "reserveAToken" argument should be "IERC20WithPermit".

Listing 32:

```
81 address reserveAToken,
```

3.33 CVF-33

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** BaseParaSwapAdapter.sol

Description We didn't review the "withdraw" function.

Listing 33:

```
102 LENDING_POOL.withdraw(reserve, amount, address(this));
```

3.34 CVF-34

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** BaseParaSwapAdapter.sol

Description Despite the comment, not all the signature params are checked.

Listing 34:

```
107 * If signature params are set to 0, then permit won't be called.
```

3.35 CVF-35

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

Recommendation The conversion of 'signature.deadline' to 'uint256' type is redundant, as the 'deadline' fields of the 'PermitSignature' structure is already 'uint256'.

Listing 35:

```
113 !(uint256(signature.deadline) == uint256(signature.v) && uint256
    ↪ (signature.deadline) == 0);
```

3.36 CVF-36

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

Description This function allows rescuing normal ERC-20 tokens, but cannot rescue ether, NFT and exotic ERC-20 tokens, such as those that charge a fee on top of the transfer amount.
Recommendation Consider allowing an owner to execute arbitrary transaction on behalf of the smart contract.

Listing 36:

```
121 function rescueTokens(IERC20 token) external onlyOwner {
```