Report

v. 1.0

Customer

Exactly

Smart Contract Audit

Exactly Peripheral Contracts

26th September 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	26.09.23	A. Zveryanskaya	Initial Draft
0.2	26.09.23	A. Zveryanskaya	Minor revision
1.0	26.09.23	A. Zveryanskaya	Release
1.1	26.09.23	A. Zveryanskaya	Scope links updated, CVF-9 typo fixed
2.0	26.09.23	A. Zveryanskaya	Release

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting (Mikhail Vladimirov and Dmitry Khovratovich) 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.

Exactly is a decentralized, non-custodial and open-source protocol that provides an autonomous fixed and variable interest rate market enabling users to frictionlessly exchange the time value of their assets and completing the DeFi credit market.

3 Project scope

We were asked to review files:

- Original Code
- Diff Code
- Code with Fixes

contracts/: commit/165031cfa454fbb316d29e47305c09957a8dd47a		
EXA.sol	DebtManager.sol	Airdrop.sol
Market.sol		

4 Methodology

The methodology is not a strict formal procedure, but rather a selection of methods and tactics combined differently and tuned for each particular project, depending on the project structure and technologies used, as well as on client expectations from the audit.

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows best code 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 as well as 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 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 if code actually does what it is supposed to do, if that algorithms are optimal and correct, and if proper data types are used. We also make sure 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.

We classify issues by the following severity levels:

- **Critical issue** directly affects the smart contract functionality and may cause a significant loss.
- **Major issue** is either a solid performance problem or a sign of misuse: a slight code modification or environment change may lead to loss of funds or data. Sometimes it is an abuse of unclear code behaviour which should be double checked.
- **Moderate issue** is not an immediate problem, but rather suboptimal performance in edge cases, an obviously bad code practice, or a situation where the code is correct only in certain business flows.
- **Minor issues** contain code style, best practices and other recommendations.

5 Our findings

We found 11 moderate, and a few less important issues.

	Info	Fixed
Moderate	11	1
Minor	11	2

Fixed 3 out of 25 issues

6 Moderate Issues

CVF-1. INFO

- **Category** Suboptimal
- **Source** Airdrop.sol

Recommendation This mapping is redundant, as streams[beneficiary] != 0 could be used as a criteria that a beneficiary already claimed his airdrop.

Client Comment *We agree that the mapping is redundant, but since the refactor only relies on an optimization and the airdrop contract was already launched and used by users, we don't find the change being worth it at this point.*

```
19 mapping(address => bool) public claimed;
```

CVF-2. INFO

- **Category** Suboptimal
- **Source** DebtManager.sol

Recommendation The ”mulDivUp” calls here could be replaced with simple divisions rounding up.

Client Comment *Those divisions are done on purpose so Exactly contracts keep using solmate's FixedPointMath library. As for now, we will not implement a custom-optimized division rounding up.*

136
```
  loopCount = uint256(amount).mulDivUp(1, tokens[0].balanceOf(
    ↪ address(balancerVault)));
  amounts[0] = uint256(amount).mulDivUp(1, loopCount);
```

180
```
r.loopCount = amount.mulDivUp(1, r.tokens[0].balanceOf(address(
  ↪ balancerVault)));
r.amounts[0] = amount.mulDivUp(1, r.loopCount);
```

224
```
r.loopCount = r.repayAssets.mulDivUp(1, r.tokens[0].balanceOf(
  ↪ address(balancerVault)));
```

227
```
r.amounts[0] = r.repayAssets.mulDivUp(1, r.loopCount);
```

308
```
r.loopCount = repayAssets.mulDivUp(1, r.tokens[0].balanceOf(address(
  ↪ balancerVault)));
```

311
```
r.amounts[0] = repayAssets.mulDivUp(1, r.loopCount);
```

369
```
r.loopCount = r.repayAssets.mulDivUp(1, r.tokens[0].balanceOf(
  ↪ address(balancerVault)));
```

371
```
r.amounts[0] = r.repayAssets.mulDivUp(1, r.loopCount);
```

CVF-3. INFO

- **Category** Suboptimal
- **Source** DebtManager.sol

Description This variable is redundant.

Recommendation Just pass the original msg.sender as a part of ”userData” to the ”flashLoan” function.

Client Comment *The variable is used to avoid having to send an extra field or argument throughout the functions. We agree with the refactor but won't change it for the moment.*

```
475 address private _msgSender;
```

CVF-4. INFO

- **Category** Procedural
- **Source** EXA.sol

Description ”Unchained” initializers should be used here.

Client Comment *The Exa contract was already initialized so we won't change the code in the initialize now. Besides, for those specific initializers, there are no side effects.*

```
11 __ERC20Permit_init("exactly");
__ERC20Votes_init();
```

CVF-5. INFO

- **Category** Suboptimal
- **Source** Airdrop.sol

Description The leaf structure assumes that there could be several leafs for a single beneficiary, but with different amounts, while the ”claimed” mapping allows a beneficiary to claim at most only leaf.

Recommendation Consider using leaf hashes or indexes as keys in the ”claimed” mapping, instead of beneficiary addresses.

Client Comment *We acknowledge this finding, but for our airdrop mechanic there was only one leaf per address. We believe this is a design choice and since the airdrop was already launched we won't change this for now considering there's no side effect.*

```
36 assert(!claimed[msg.sender]);
   assert(proof.verify(root, keccak256(abi.encode(msg.sender, amount)))
      ↪ );
```

CVF-6. INFO

- **Category** Overflow/Underflow
- **Source** DebtManager.sol

Description Overflow is possible when converting type.

Recommendation Consider using safe conversion.

Client Comment *For this to be exploitable someone needs to be able to manipulate at least one of the variables of those operations and none of the components are manipulatable. In this case, there are 3 variables: targetDeposit, collateral, and deposit. The deposit variable is the only one received as an argument but can't be manipulated either because there's a prior transfer that needs to occur. This transfers an amount of real assets since the token is being validated too.*

```
131 int256 amount = int256(targetDeposit) - int256(collateral + deposit)
      ↪ ;
```

CVF-7. INFO

- **Category** Flaw
- **Source** DebtManager.sol

Description Due to rounding errors, r.positionAssets * r.loopCount may be less than the original value of ”r.positionAssets”.

Recommendation Consider taking this into account and using an incremented value of ”r.positionAssets” in some loop iterations to repay the correct total position assets.

Client Comment *We already tried different code implementations to avoid rounding mismatches around the positionAssets variable. However, every possibility that we tried ended up having another implication with partial fixed operations when loopCount > 1. For that reason, we decided to leave the code as it is and the team transferred some units of each underlying asset to the proxy contract after deployment. In this way, positionAssets is rounded in favor of the user and transactions won't revert if those specific scenarios are met. However, in many other times the rounding favors the contract so these units are also compensated throughout time.*

```
228 r.positionAssets = r.positionAssets / r.loopCount;
```

CVF-8. INFO

- **Category** Flaw
- **Source** DebtManager.sol

Description Due to rounding errors, positionAssets * r.loopCount may be less than the original value of ”positionAssets”.

Recommendation Consider taking this into account and using an incremented value of ”positionAssets” in some loop iterations to repay the correct total position assets.

Client Comment *We already tried different code implementations to avoid rounding mismatches around the positionAssets variable. However, every possibility that we tried ended up having another implication with partial fixed operations when loopCount > 1. For that reason, we decided to leave the code as it is and the team transferred some units of each underlying asset to the proxy contract after deployment. In this way, positionAssets is rounded in favor of the user and transactions won't revert if those specific scenarios are met. However, in many other times the rounding favors the contract so these units are also compensated throughout time.*

```
309 positionAssets = positionAssets / r.loopCount;
```

CVF-9. FIXED

- **Category** Suboptimal
- **Source** DebtManager.sol

Description This logic is very dangerous, as it allows a recursive call from a malicious contract to reuse the currently set "_msgSender".

Recommendation Consider requiring "_msgSender" to be zero before the call.

Client Comment *We decided to add a new safety check with a boolean variable _msgSenderSet. This will avoid recursive calls although we believe reentrancy should not happen since we are validating all contracts for which the 'DebtManager' makes the external calls.*

```
478 if (_msgSender == address(0)) _msgSender = msg.sender;
```

CVF-10. INFO

- **Category** Suboptimal
- **Source** DebtManager.sol

Description The "_msgSender" logic seems to be a dangerous mess, as it introduces a hidden parameter passed between functions and modifiers, that is very hart to track.

Recommendation Consider removing the "_msgSender" variable, and passing the corresopnding value explicitly where necessary.

Client Comment *Due to the sensible implementation that is already written and working, we believe it wouldn't be prudent to do a big refactor at this stage. Although it might look messy, this code has already been audited, tested, and validated. Efforts were made to explore simpler refactoring options, but none of the alternative approaches gained the team's consensus.*

```
495 if (sender == address(0)) _msgSender = p.account;
```

CVF-11. INFO

- **Category** Suboptimal
- **Source** DebtManager.sol

Description This check seems to be useless and dangerous. The ”MAX_ALLOWANCE_SURPLUS” is the same for all tokens, while its true value could be very different for different tokens. For example, for WBTC, 0.01e18 tokens is actually 100000000 bitcoins.

Recommendation Consider removing this check.

Client Comment *This is not accurate since the MAX_ALLOWANCE_SURPLUS is being used as a percentage that is applied to any asset independently of the number of decimals that they have. Besides that, we believe it isn't useless since we are applying this check to add an extra layer of security to be sure that there are no accounts with a considerable positive allowance after transactions are confirmed. Given that the recent hack was profitable because there were accounts with allowances, those functions will override any allowance that the user already set so we want to try to achieve the same behavior in a responsible way. We can check the transaction result off-chain during gas estimation so we can validate this on the web-app and will not prevent users from interacting with the contract.*

```
500 if (token.allowance(p.account, address(this)) > p.value.mulWadDown(
        ↪ MAX_ALLOWANCE_SURPLUS)) {
```

CVF-12. INFO

- **Category** Unclear behavior
- **Source** EXA.sol

Recommendation It is unclear how this could compile as there is no ”permit” implementation inherited.

Client Comment *EXA.sol is inheriting ERC20VotesUpgradeable and that abstract contract inherits ERC20PermitUpgradeable, which is the one that has the __ERC20Permit_init(string memory name) implementation. We are using Open Zeppelin's v4.*

```
11 __ERC20Permit_init("exactly");
```

7 Minor Issues

CVF-13. INFO

- **Category** Procedural
- **Source** Airdrop.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.

Recommendation Consider specifying as "^0.8.0". Also relevant for: DebtManager.sol, EXA.sol.

Client Comment *The team will be working on this soon.*

```
2 pragma solidity 0.8.17;
```

CVF-14. INFO

- **Category** Procedural
- **Source** Airdrop.sol

Description We didn't review these files.

```
5 import { MerkleProofLib } from "solmate/src/utils/MerkleProofLib.sol
    ↪ ";
  import { SafeTransferLib, ERC20 } from "solmate/src/utils/
    ↪ SafeTransferLib.sol";
```

CVF-15. INFO

- **Category** Bad datatype
- **Source** Airdrop.sol

Recommendation The duration should be a named constant.

Client Comment *We believe 4 * 4 weeks is clear enough for it not to be a constant, also considering that it is only being used once.*

```
47 durations: Durations({ cliff: 0, total: 4 * 4 weeks }),
```

CVF-16. INFO

- **Category** Procedural
- **Source** Airdrop.sol

Recommendation This interface should be declared in a separate file named ”ISablierV2LockupLinear.sol”.

Client Comment *This is a style choice and we are being consistent with other Exactly contracts in which we follow the same code structure.*

```
57 interface ISablierV2LockupLinear {
```

CVF-17. INFO

- **Category** Procedural
- **Source** Airdrop.sol

Description Defining top-level structures in a file named after a contract makes it harder to navigate through code.

Recommendation Consider moving the structures definitions into the contract or moving them into a separate file.

Client Comment *This is a style choice and we are being consistent with other Exactly contracts in which we follow the same code structure.*

```
61 struct Durations {
```

```
66 struct Broker {
```

```
71 struct CreateWithDurations {
```

CVF-18. INFO

- **Category** Procedural
- **Source** DebtManager.sol

Description We didn't review these files.

```
4 import { SafeTransferLib } from "solmate/src/utils/SafeTransferLib.
    ↪ sol";
  import { FixedPointMathLib } from "solmate/src/utils/
    ↪ FixedPointMathLib.sol";
```

```
13 import { Auditor, IPriceFeed, MarketNotListed } from "../Auditor.sol
    ↪ ";
```

CVF-19. INFO

- **Category** Readability
- **Source** DebtManager.sol

Description The code below looks like it is always executed, while it is only executed when amount > 0.

Recommendation Consider putting the rest of the function into an explicit "else" branch.

Client Comment *This is a style choice. We believe it is better for readability as it is.*

```
135 }
```

CVF-20. FIXED

- **Category** Bad naming
- **Source** DebtManager.sol

Description The function name is very confusing, as it collides with the built-in function "call" and also the function actually doesn't call anything.

Recommendation Consider renaming.

Client Comment *We renamed the function from 'call' to 'hash'.*

```
452 function call(bytes memory data) internal returns (bytes memory) {
```

CVF-21. FIXED

- **Category** Suboptimal
- **Source** DebtManager.sol

Recommendation The ”memCallHash != 0” check is redundant, as it is superseded by the subsequent check.

Client Comment *We removed the check.*

```
462 assert(msg.sender == address(balancerVault) && memCallHash !=
        ↪ bytes32(0) && memCallHash == keccak256(userData));
```

CVF-22. INFO

- **Category** Procedural
- **Source** DebtManager.sol

Description Declaring top-level errors in a file named after a contract makes it harder to navigate through code.

Recommendation Consider either moving the error definitions into the contract or moving them into a separate file.

Client Comment *This is a style choice and we are being consistent with other Exactly contracts in which we follow the same code structure.*

```
548 error AllowanceSurplus();
    error InvalidOperation();
```

CVF-23. INFO

- **Category** Procedural
- **Source** DebtManager.sol

Description Declaring top-level structures in a file named after a contract makes it harder to navigate through code.

Recommendation Consider either moving the structures definitions into the contract or moving them into a separate file.

Client Comment *This is a style choice and we are being consistent with other Exactly contracts in which we follow the same code structure.*

551 `struct Permit {`

560 `struct Permit2 {`

565 `struct SwapCallbackData {`

576 `struct RollVars {`

CVF-24. INFO

- **Category** Procedural
- **Source** DebtManager.sol

Recommendation These interfaces should be declared in separate files named ”IBalancerVault.sol” and ”IPermit2” respectively.

Client Comment *This is a style choice and we are being consistent with other Exactly contracts in which we follow the same code structure.*

589 `interface IBalancerVault {`

598 `interface IPermit2 {`

CVF-25. INFO

- **Category** Bad datatype
- **Source** EXA.sol

Recommendation The initial supply should be a named constant.

Client Comment *It is only being used once, so we prefer to leave it as it is.*

```
13 _mint(msg.sender, 10_000_000e18);
```

ABDK
Consulting

About us

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.

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting