ABDK CONSULTING

SMART CONTRACT AUDIT

Balancer

Balancer-v2

Solidity

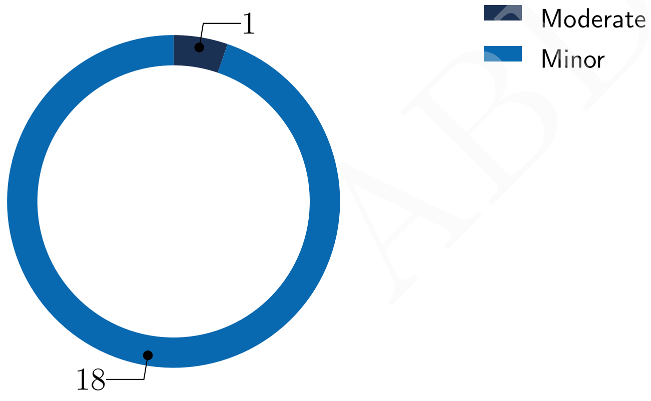
abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich

27th May 2022

We've been asked to review 15 files in a Github repository. We found 1 moderate issue, and a few less important issues.



Findings

ID	Severity	Category	Status
CVF-1	Minor	Procedural	Info
CVF-2	Minor	Suboptimal	Info
CVF-3	Minor	Bad naming	Info
CVF-4	Minor	Suboptimal	Fixed
CVF-5	Minor	Suboptimal	Info
CVF-6	Minor	Suboptimal	Info
CVF-7	Minor	Bad datatype	Fixed
CVF-8	Minor	Suboptimal	Info
CVF-9	Minor	Suboptimal	Info
CVF-10	Minor	Unclear behavior	Fixed
CVF-11	Moderate	Suboptimal	Info
CVF-12	Minor	Unclear behavior	Fixed
CVF-13	Minor	Procedural	Info
CVF-14	Minor	Suboptimal	Info
CVF-15	Minor	Suboptimal	Info
CVF-16	Minor	Unclear behavior	Fixed
CVF-17	Minor	Suboptimal	Fixed
CVF-18	Minor	Procedural	Info
CVF-19	Minor	Suboptimal	Fixed

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	May 19, 2022	D. Khovratovich	Initial Draft
0.2	May 20, 2022	D. Khovratovich	Minor revision
1.0	May 27, 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 repository:

- authorizer/IBasicAuthorizer.sol
- authorizer/TimelockAuthorizer.sol
- authorizer/TimelockAuthorizerMigrator.sol
- authorizer/TimelockExecutor.sol
- helpers/BalancerErrors.sol
- helpers/IAuthentication.sol
- helpers/InputHelpers.sol
- helpers/ISignaturesValidator.sol
- helpers/ITemporarilyPausable.sol
- interfaces/IAuthorizer.sol
- interfaces/IFlashLoanRecipient.sol
- interfaces/IProtocolFeesCollector.sol
- interfaces/IVault.sol
- misc/IWETH.sol
- openzeppelin/Address.sol

The fixes were provided in the bb0d705 commit.

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
- **Category** Procedural
- **Status** Info
- **Source** TimelockAuthorizer.sol

Description We didn't review these files.
Client Comment These files have been included in previous audits and no meaningful changes have been made since.

Listing 1:

```
import "@balancer-labs/v2-solidity-utils/contracts/openzeppelin/
    ↪ Address.sol";
import "@balancer-labs/v2-solidity-utils/contracts/helpers/
    ↪ InputHelpers.sol";
import "@balancer-labs/v2-solidity-utils/contracts/helpers/
    ↪ BalancerErrors.sol";
import "@balancer-labs/v2-solidity-utils/contracts/helpers/
    ↪ IAuthentication.sol";

import "../interfaces/IVault.sol";
import "../interfaces/IAuthorizer.sol";
```

3.2 CVF-2

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

Recommendation It would be more efficient to replace these two fields with a single enum field representing the current execution state.
Client Comment Merging these two values into a single enum would reduce readability for offchain watchers as they would need to decode the enum. Switching to an enum wouldn't have any gas benefits as these bools already sit in the same slot so the additional readability is preferable.

Listing 2:

```
bool executed;
bool cancelled;
```

3.3 CVF-3

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

Recommendation Events are usually named via nouns, such as "Schedule", "Execution", "Cancellation", "ActionDelay", etc.
Client Comment These names are consistent with the Balancer style of "ObjectAction" which provide context on what is changing and how. The recommended names don't provide this context as they are lacking information on either the "Object" or "Action".

Listing 3:

```
event ExecutionScheduled(bytes32 indexed actionId, uint256 indexed scheduledExecutionId);

event ExecutionExecuted(uint256 indexed scheduledExecutionId);

event ExecutionCancelled(uint256 indexed scheduledExecutionId);

event ActionDelaySet(bytes32 indexed actionId, uint256 delay);

event PermissionGranted(bytes32 indexed actionId, address indexed account, address indexed where);

event PermissionRevoked(bytes32 indexed actionId, address indexed account, address indexed where);

event RootSet(address indexed root);
```

3.4 CVF-4

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

Description This function is redundant, as the "root" variable is already public.
Client Comment Addressed in 4f69fd7. The ‘root‘ variable is made private for consistency with CVF 5

Listing 4:

```
function isRoot(address account) public view returns (bool) {
```

3.5 CVF-5

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

Recommendation This function would be redundant if the "_rootTransferDelay" variable would be public.
Client Comment The current implementation is consistent with the code-style of Balancer contracts which is that storage variables are kept internal or private and we expose getters in the style "getX".

Listing 5:

```
function getRootTransferDelay() public view returns (uint256) {
```

3.6 CVF-6

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

Recommendation This function would be redundant if the "_vault" variable would be public.
Client Comment See CVF 5

Listing 6:

```
function getVault() external view returns (address) {
```

3.7 CVF-7

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

Recommendation The return type could be more specific.
Client Comment Addressed in e6bb4c1a. No address may interact with TimelockExecutor except the Authorizer so it's not meaningful to add an interface for this.

Listing 7:

```
function getVault() external view returns (address) {

function getExecutor() external view returns (address) {
```

3.8 CVF-8

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

Recommendation This function would be redundant if the "_executor" variable would be public.
Client Comment See CVF 5

Listing 8:

```
171 function getExecutor() external view returns (address) {
```

3.9 CVF-9

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

Description This check is redundant as it is anyway performed in the next line.
Client Comment These require statements provide useful feedback to callers who provide a non-existent ‘executionId‘ while removing these lines would result in an unhelpful "invalid opcode" error. Clarity is preferred for these actions.

Listing 9:

```
274 require(scheduledExecutionId < scheduledExecutions.length, "ACTION_DOES_NOT_EXIST");

357 require(scheduledExecutionId < scheduledExecutions.length, "ACTION_DOES_NOT_EXIST");

379 require(scheduledExecutionId < scheduledExecutions.length, "ACTION_DOES_NOT_EXIST");
```

3.10 CVF-10

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

Description This is emitted even if nothing has really changed.
Client Comment Addressed in 1dda63e9

Listing 10:

```
288 emit RootSet(newRoot);

314 emit ActionDelaySet(actionId, delay);
```

3.11 CVF-11

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

Description This check prevents setting delay for any action, other than "setAuthorizer", above the "setAuthorizer" action delay, but doesn't prevent setting the "setAuthorizer" action delay below the delay of some other action. This means that the "setAuthorizer" action delay is not guaranteed to be the highest one.

Recommendation Consider using the minimum of a delay set for an action and the "setAuthorizer" action delay as the effective delay for the action.

Client Comment This check is intended to flag when the delay which is being passed by the user won't be enforced (as otherwise it may be bypassed by changing the Authorizer) so they don't believe that the delay for an action is longer than it is in reality - automatically shortening the delay passed by the user would work against this goal.

We don't expect the 'setAuthorizer' delay to be reduced, or if done so it would be it would be as part of a widespread review of the delays on various actions.

Listing 11:

```
bool isAllowed = actionId == setAuthorizerActionId || delay <=
    ↪ delaysPerActionId[setAuthorizerActionId];
require(isAllowed, "DELAY_EXCEEDS_SET_AUTHORIZER");
```

3.12 CVF-12

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

Recommendation This event should include the data returned by the execution, otherwise it would be very hard to obtain this data.

Client Comment Addressed in a1c79af2

Listing 12:

```
emit ExecutionExecuted(scheduledExecutionId);
```

3.13 CVF-13

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

Description This seems odd, as one, who is able to schedule actions, may not only cancel the actions he scheduled, but also the action scheduled by other users.
Client Comment This is expected. Requiring that only the address which scheduled an action may cancel it prevents recovery from a malicious action being scheduled. Should an address exploit this to perform a DOS, they would have their permissions revoked (an action which they cannot cancel).

Listing 13:

```
// The permission to cancel a scheduled action is the same one used to schedule it
```

3.14 CVF-14

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

Recommendation It would be more efficient to pass a single array of structs with two fields instead of two parallel arrays. This would also make the length check unnecessary.
Client Comment This is true however would come at the expense of requiring passing tuples manually. Considering the poor support for these in many interfaces it's preferred to pass arrays of value types as we believe this would prevent help mistakes.

Listing 14:

```
    bytes32[] memory actionIds,

    address[] memory where

    bytes32[] memory actionIds,

    address[] memory where

function renouncePermissions(bytes32[] memory actionIds, address[] memory where) external {
```

3.15 CVF-15

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

Description This function requires the caller to be a revoker, i.e. to be able to revoke the permission from anybody. Such requirement seems redundant for a user who wants to renounce his own permission.
Recommendation Consider removing this requirement.
Client Comment This function doesn't require the caller to be a revoker. This can be seen by comparing this function with the 'revokePermissions' function which does perform this check.

Listing 15:

```
function renouncePermissions(bytes32[] memory actionIds, address[] memory where) external {
```

3.16 CVF-16

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

Description This should be calculated only if "hasPermission[granularActionId, account, where)" is false.
Client Comment Addressed in 8f7385b8

Listing 16:

```
bytes32 globalActionId = getActionId(actionId, WHATEVER);
```

3.17 CVF-17

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

Recommendation This could be optimized via assembly like this: function _decodeSelector (bytes memory data) internal pure returns (bytes4 result) { if (data.length < 4) return bytes4 (0); assembly { result := mload (add (data, 0x20)); } }
Client Comment Addressed in d6f3cb4a

Listing 17:

```
return bytes4(data[0]) | (bytes4(data[1]) >> 8) | (bytes4(data[2]) >> 16) | (bytes4(data[3]) >> 24);
```

3.18 CVF-18

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

Description We didn't review this file.
Client Comment See CVF 1

Listing 18:

```
import "@balancer-labs/v2-solidity-utils/contracts/openzeppelin/Address.sol";
```

3.19 CVF-19

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

Recommendation By specifying "using Address for address" this line could be simplified to: return target.functionCall (data);
Client Comment Addressed in 8112a115

Listing 19:

```
return Address.functionCall(target, data);
```