1INCH CUMULATIVE MERKLE DROP SMART CONTRACT AUDIT

August 26, 2021

MixBytes()

CONTENTS

1.INTRODUCTION 2
DISCLAIMER 2
PROJECT OVERVIEW 2
SECURITY ASSESSMENT METHODOLOGY 3
EXECUTIVE SUMMARY 5
PROJECT DASHBOARD 5
2.FINDINGS REPORT 7
2.1.CRITICAL 7
CRT-1 Shorted root hash can be brute forced 7
2.2.MAJOR 8
2.3.WARNING 8
WRN-1 Possible incorrect initialization 8
WRN-2 Owner can set unlimited amount 9
WRN-3 Incorrect change of the root 10
WRN-4 Possible work with uninitialized `merkleRoot` 11
WRN-5 `merkleProof` can has length == 0 12
WRN-6 Possible insufficient balance 13
2.4.COMMENT 14
CMT-1 Unnecessary use of safeMath 14
CMT-2 Not enough comments 15
CMT-3 Unusable library 16
CMT-4 Incorrect comment 17
CMT-5 Public access 18
CMT-6 Non-consistent number formatting 19
CMT-7 Possible gas savings 20
3.ABOUT MIXBYTES 21

1. INTRODUCTION

1.1 DISCLAIMER

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of 1Inch. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 PROJECT OVERVIEW

1inch is a well known protocol which allows users to make the most profitable swaps between assets via using complex swaps in several protocols.

1.3 SECURITY ASSESSMENT METHODOLOGY

A group of auditors are involved in the work on the audit who check the provided source code independently of each other in accordance with the methodology described below:

01 Project architecture review:
> Reviewing project documentation
> General code review
> Reverse research and study of the architecture of the code based on the source code only
> Mockup prototyping

Stage goal:
Building an independent view of the project's architecture and identifying logical flaws in the code.

02 Checking the code against the checklist of known vulnerabilities:
> Manual code check for vulnerabilities from the company's internal checklist
> The company's checklist is constantly updated based on the analysis of hacks, research and audit of the clients' code
> Checking with static analyzers (i.e Slither, Mythril, etc.)

Stage goal:
Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flashloan attacks, etc.)

03 Checking the code for compliance with the desired security model:
> Detailed study of the project documentation
> Examining contracts tests
> Examining comments in code
> Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit
> Exploits PoC development using Brownie

Stage goal:
Detection of inconsistencies with the desired model

04 Consolidation of interim auditor reports into a general one:
> Cross-check: each auditor reviews the reports of the others
> Discussion of the found issues by the auditors
> Formation of a general (merged) report

Stage goal:
Re-check all the problems for relevance and correctness of the threat level and provide the client with an interim report.

05 Bug fixing & re-check:
> Client fixes or comments on every issue
> Upon completion of the bug fixing, the auditors double-check each fix and set the statuses with a link to the fix

Stage goal:
Preparation of the final code version with all the fixes

06 Preparation of the final audit report and delivery to the customer.

Findings discovered during the audit are classified as follows:

FINDINGS SEVERITY BREAKDOWN

Level	Description	Required action
Critical	Bugs leading to assets theft, fund access locking, or any other loss funds to be transferred to any party	Immediate action to fix issue
Major	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.	Implement fix as soon as possible
Warning	Bugs that can break the intended contract logic or expose it to DoS attacks	Take into consideration and implement fix in certain period
Comment	Other issues and recommendations reported to/acknowledged by the team	Take into consideration

Based on the feedback received from the Customer's team regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The project team is aware of this finding. Recommendations for this finding are planned to be resolved in the future. This finding does not affect the overall safety of the project.
No issue	Finding does not affect the overall safety of the project and does not violate the logic of its work.

1.4 EXECUTIVE SUMMARY

Cumulative merkle drop is a project from 1inch protocol, allowing users to withdraw some tokens from contract in case they have allowance to do it. Allowance to withdraw funds from contract is stored via merkle tree. As always, to save ETH for users 1inch implemented their own merkle tree proof verification function, which is more efficient than openzeppelin well known solution.

1.5 PROJECT DASHBOARD

Client	1Inch
Audit name	Cumulative merkle drop
Initial version	96fb63d0cbfea73603e7500961c71e8ab1fb8c10
Final version	1f8b2a6ed27d1b2d18cf8475e42eece60f41c896
Date	August 16, 2021 - August 26, 2021
Auditors engaged	2 auditors

FILES LISTING

CumulativeMerkleDrop.sol	https://github.com/1inch/cumulative-merkle-drop/tree/96fb63d0cbfea73603e7500961c71e8ab1fb8c10/contracts/CumulativeMerkleDrop.sol
CumulativeMerkleDrop128.sol	https://github.com/1inch/cumulative-merkle-drop/tree/96fb63d0cbfea73603e7500961c71e8ab1fb8c10/contracts/CumulativeMerkleDrop128.sol
CumulativeMerkleDrop160.sol	https://github.com/1inch/cumulative-merkle-drop/tree/96fb63d0cbfea73603e7500961c71e8ab1fb8c10/contracts/CumulativeMerkleDrop160.sol
ICumulativeMerkleDrop.sol	https://github.com/1inch/cumulative-merkle-drop/tree/96fb63d0cbfea73603e7500961c71e8ab1fb8c10/contracts/interfaces/ICumulativeMerkleDrop.sol
ICumulativeMerkleDrop128.sol	https://github.com/1inch/cumulative-merkle-drop/tree/96fb63d0cbfea73603e7500961c71e8ab1fb8c10/contracts/interfaces/ICumulativeMerkleDrop128.sol

ICumulativeMerkleDrop160.sol	https://github.com/1inch/cumulative-merkle-drop/tree/96fb63d0cbfea73603e7500961c71e8ab1fb8c10/contracts/interfaces/ICumulativeMerkleDrop160.sol

FINDINGS SUMMARY

Level	Amount
Critical	1
Major	0
Warning	6
Comment	7

CONCLUSION

Smart contracts have been audited and several suspicious places have been spotted. During the audit one issue was marked critical, as it might lead to undesired behaviour. No major issues were spotted, several warnings and comments were found and discussed with the client. After working on the reported findings all of them were resolved or acknowledged. So, the contracts are assumed as secure to use according to our security criteria.Final commit identifier with all fixes: `1f8b2a6ed27d1b2d18cf8475e42eece60f41c896`

2. FINDINGS REPORT

2.1 CRITICAL

CRT-1	Shorted root hash can be brute forced
File	CumulativeMerkleDrop128.sol CumulativeMerkleDrop160.sol
Severity	Critical
Status	Acknowledged

DESCRIPTION

Malicious user can brute force proof for case `leaf = _keccak128(abi.encodePacked(address(malicious_user), token.balanceOf(address(this))))` because root hash is shorted:
CumulativeMerkleDrop128.sol#L95
CumulativeMerkleDrop160.sol#L95

RECOMMENDATION

We recommend to use only 256bit implementation of cumulative merkle drop, because it is a garauntee that hash collision wouldn't appear.

CLIENT'S COMMENTARY

We'll deprecate 128 and 160 bit versions

2.2 MAJOR

Not Found

2.3 WARNING

WRN-1	Possible incorrect initialization
File	CumulativeMerkleDrop128.sol CumulativeMerkleDrop.sol CumulativeMerkleDrop160.sol
Severity	Warning
Status	Acknowledged

DESCRIPTION

In `CumulativeMerkleDropXXX` constructor `token_` can be zero address:
CumulativeMerkleDrop128.sol#L21
CumulativeMerkleDrop.sol#L22
CumulativeMerkleDrop160.sol#L21

RECOMMENDATION

We recommend to add the following check:

```
require(token_ != address(0), "CMD: Incorrect address");
```

CLIENT'S COMMENTARY

Won't fix

WRN-2	Owner can set unlimited amount
File	CumulativeMerkleDrop128.sol CumulativeMerkleDrop.sol CumulativeMerkleDrop160.sol
Severity	Warning
Status	Acknowledged

DESCRIPTION

Owner can set merkle root in which, for his account, allowed amount can be very big:
CumulativeMerkleDrop128.sol#L24-L27
CumulativeMerkleDrop.sol#L25-L28
CumulativeMerkleDrop160.sol#L24-L27

RECOMMENDATION

We recommend to use multisig wallet for `owner`.

CLIENT'S COMMENTARY

Noted

WRN-3	Incorrect change of the root
File	CumulativeMerkleDrop128.sol CumulativeMerkleDrop.sol CumulativeMerkleDrop160.sol
Severity	Warning
Status	Acknowledged

DESCRIPTION

In the following function it is possible to update root with the same value:
CumulativeMerkleDrop128.sol#L26
CumulativeMerkleDrop.sol#L27
CumulativeMerkleDrop160.sol#L26

RECOMMENDATION

We recommend to add the following check:

```
require(merkleRoot_ != merkleRoot, "CMD: Same root");
```

CLIENT'S COMMENTARY

Won't fix

WRN-4	Possible work with uninitialized `merkleRoot`
File	CumulativeMerkleDrop128.sol CumulativeMerkleDrop.sol CumulativeMerkleDrop160.sol
Severity	Warning
Status	Acknowledged

DESCRIPTION

In the following function it is possible to `merkleRoot == 0`:
CumulativeMerkleDrop128.sol#L35
CumulativeMerkleDrop.sol#L36
CumulativeMerkleDrop160.sol#L36

RECOMMENDATION

We recommend to add check, that `merkleRoot` was initialized.

CLIENT'S COMMENTARY

Won't fix

WRN-5	`merkleProof` can has length == 0
File	CumulativeMerkleDrop128.sol CumulativeMerkleDrop.sol CumulativeMerkleDrop160.sol
Severity	Warning
Status	Acknowledged

DESCRIPTION

In the following function it is possible to `merkleProof.length == 0`:
CumulativeMerkleDrop128.sol#L39
CumulativeMerkleDrop.sol#L40
CumulativeMerkleDrop160.sol#L39

RECOMMENDATION

We recommend to add the check on length of proof (`merkleProof.length == precalculated_value`) and also add following check:

```
require(merkleProof.length > 0, "CMD: Incorrect proof");
```

CLIENT'S COMMENTARY

Won't fix

WRN-6	Possible insufficient balance
File	CumulativeMerkleDrop128.sol CumulativeMerkleDrop.sol CumulativeMerkleDrop160.sol
Severity	Warning
Status	No Issue

DESCRIPTION

`token` balance of the contract can be less than `amount`:
CumulativeMerkleDrop128.sol#L48
CumulativeMerkleDrop.sol#L49
CumulativeMerkleDrop160.sol#L48

RECOMMENDATION

We recommend to add the following check:

```
require(IERC20(token).balanceOf(address(this)) >= amount, "CMD: Insufficient balance");
```

CLIENT'S COMMENTARY

This is indirectly enforced by transfer function

2.4 COMMENT

CMT-1	Unnecessary use of safeMath
File	CumulativeMerkleDrop128.sol CumulativeMerkleDrop.sol CumulativeMerkleDrop160.sol
Severity	Comment
Status	Fixed at 5d19a1af

DESCRIPTION

`cumulativeAmount` always greater than `preclaimed` because it is checked above:
CumulativeMerkleDrop128.sol#L47
CumulativeMerkleDrop.sol#L48
CumulativeMerkleDrop160.sol#L47

RECOMMENDATION

We recommend to use uncheked {} math to safe some gas.

CMT-2	Not enough comments
File	CumulativeMerkleDrop128.sol CumulativeMerkleDrop.sol CumulativeMerkleDrop160.sol
Severity	Comment
Status	No Issue

DESCRIPTION

In the following function many assembly operations are used without comments about their functionality:
CumulativeMerkleDrop128.sol#L74-L96
CumulativeMerkleDrop.sol#L60-L83
CumulativeMerkleDrop160.sol#L74-L96

RECOMMENDATION

We recommend to add comments for each operation in assembly.

CLIENT'S COMMENTARY

There is a simple solidity version of verify that is commented out, that is used as an explanation of what assembly code is doing.

CMT-3	Unusable library
File	CumulativeMerkleDrop.sol
Severity	Comment
Status	Fixed at 1f8b2a6e

DESCRIPTION

MerkleProof library is not used in this contract:
CumulativeMerkleDrop.sol#L14

RECOMMENDATION

We recommend to remove this library from contract.

CMT-4	Incorrect comment
File	ICumulativeMerkleDrop160.sol ICumulativeMerkleDrop128.sol ICumulativeMerkleDrop.sol
Severity	Comment
Status	Fixed at da991f54

DESCRIPTION

Comment for `MerkelRootUpdated` event is copied from `Claimed` event:
ICumulativeMerkleDrop160.sol#L7
ICumulativeMerkleDrop128.sol#L7
ICumulativeMerkleDrop.sol#L7

RECOMMENDATION

We recommend to change the comment.

CMT-5	Public access
File	CumulativeMerkleDrop.sol CumulativeMerkleDrop128.sol CumulativeMerkleDrop160.sol
Severity	Comment
Status	Fixed at 0ca41cde

DESCRIPTION

Next function can be internal:
CumulativeMerkleDrop.sol#L58
CumulativeMerkleDrop128.sol#L72
CumulativeMerkleDrop160.sol#L72

RECOMMENDATION

We recommend to mark this function as `internal` in every `CumulativeMerkleDropXXX.sol` contract.

CMT-6	Non-consistent number formatting
File	CumulativeMerkleDrop.sol CumulativeMerkleDrop128.sol CumulativeMerkleDrop160.sol
Severity	Comment
Status	No Issue

DESCRIPTION

In `verifyAsm` for 256 and 128 bit versions keccak argument is written in decimal format:
CumulativeMerkleDrop.sol#L78
CumulativeMerkleDrop128.sol#L92
but in 160 bit - in hex format
CumulativeMerkleDrop160.sol#L92
This fact may confuse readers of this code.

RECOMMENDATION

We recommend to use single format.

CLIENT'S COMMENTARY

We'll remove 160 and 128 bit versions. So that will not be relevant.

CMT-7	Possible gas savings
File	CumulativeMerkleDrop.sol CumulativeMerkleDrop128.sol CumulativeMerkleDrop160.sol
Severity	Comment
Status	Acknowledged

DESCRIPTION

Function `verifyAsm` costs a lot of gas, so it's meaningful to place checks before that function. We can place check for `preclaimed < cumulativeAmount`
CumulativeMerkleDrop.sol#L44
CumulativeMerkleDrop128.sol#L43
CumulativeMerkleDrop160.sol#L43
also `amount <= IERC20(token).balanceOf(address(this))`.

RECOMMENDATION

We recommend to place the following checks above `verifyAsm`:

```
require(preclaimed < cumulativeAmount, "CMD: Nothing to claim");
require(amount <= IERC20(token).balanceOf(address(this)), "CMD: Insufficient balance");

// Verify the merkle proof
bytes16 leaf = _keccak128(abi.encodePacked(account, cumulativeAmount));
require(verifyAsm(merkleProof, expectedMerkleRoot, leaf), "CMD: Invalid proof");
```

CLIENT'S COMMENTARY

Won't fix

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build open-source solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

BLOCKCHAINS

- Ethereum
- Cosmos
- EOS
- Substrate

TECH STACK

- Python
- Solidity
- Rust
- C++

CONTACTS

https://github.com/mixbytes/audits_public

https://mixbytes.io/

hello@mixbytes.io

https://t.me/MixBytes

https://twitter.com/mixbytes