HashEx

Bitcrush Arcade

smart contracts audit report

Prepared for:

bitcrusharcade.io

Authors: HashEx audit team

July 2021

Contents

Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report.

The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

Introduction

HashEx was commissioned by the Bitcrush Arcade team to perform an audit of their smart contracts. The audit was conducted between June 28 and July 03, 2021.

The code located in the @Bitcrush-Arcade/crush_contracts Github repository was audited after the `5c2c771` commit. The whitepaper is available on the team's website.
The CRUSH token is deployed to Binance Smart Chain (BSC):
`0x0Ef0626736c2d484A792508e99949736D0AF807e`.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts.
- Formally check the logic behind given smart contracts.

Information in this report should be used to understand the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

Update: Bitcrush Arcade has responded to this report. Individual responses were added after each item in the section. The updated code is located in the same repository after the `4833d0a` commit.
The BitcrushStaking contract is deployed to Binance Smart Chain (BSC):
`0x0Ef0626736c2d484A792508e99949736D0AF807e`.

Contracts overview

`CrushCoin.sol`

Mintable capped ERC20 token. Also referred to as CRUSHToken or Token.

`staking.sol`

Staking contract for CRUSH token. Also referred to as BitcrushStaking or Staking.

Found issues

ID	Title	Severity	Response
01	Token: mint() is open for the owner	High	Fixed
02	Staking: no cap for crushPerBlock	High	Fixed
03	Staking: no cap for fees	High	Fixed
04	Staking: compoundAll() reduces rewards	High	Responded
05	Staking: emergencyTotalPoolWithdraw() function	High	Responded
06	Staking: claim() not updating totalPool	High	Fixed
07	Staking: unchecked math	High	Fixed
08	Staking: compoundAll() gas limit	Medium	Responded
09	Staking: data not cleared on leaving	Medium	Fixed
10	Token: empty constructor	Low	Responded
11	Token: tokensBurned variable not needed	Low	Responded
12	Staking: no explicit visibility	Low	P/Fixed
13	Staking: variables naming	Low	P/Fixed
14	Staking: unindexed events	Low	Fixed
15	Staking: checks on uint <= 0	Low	Fixed
16	Staking: excessive computations	Low	Fixed
17	Staking: typos in comments	Low	Fixed
18	Staking: converting block number to time	Low	P/Fixed
19	General recommendations	Low	Responded

#01 Token: mint() is open for the owner — High

CRUSH is a mintable ERC20 standard [1] token with `mint()` accessible for the owner even above the cap which makes the token vulnerable in case of the owner being hacked or acts maliciously.

Update: the issue was fixed with the renouncing of the ownership in BSC tx:

0x7d46b85ebac08768b7ae1bf8d9f32722ccf638d9b3fb0e88e5b11f8f2b809613

#02 Staking: no cap for crushPerBlock — High

`setCrushPerBlock()` function is used for updating the `crushPerBlock` parameter. Although it's the onlyOwner function, we recommend adding safety guards — capping the new value. If the owner's account gets compromised or the owner acts maliciously, the attacker can set an arbitrary big value for the `crushPerBlock` variable and take all the reward's pool. This regards only the case when the owner's account is not trusted or is not properly secured.

Update: the issue was fixed.

#03 Staking: no cap for fees — High

`setPerformanceFeeCompounder()`, `setPerformanceFeeBurn()`, `setEarlyWithdrawFee()`, `setPerformanceFeeReserve()`, `setEarlyWithdrawFeeTime()`, `setFeeDivisor()` functions lack the safety guards (limit from above). This regards only the case when the owner's account is not trusted or is not properly secured.

Update: the issue was fixed.

#04 Staking: compoundAll() reduces rewards — High

`compoundAll()` is a public method to compound all of the rewards for all stakers. It works the same as `singleCompound()` does for msg.sender but takes multiple additional fees. Taking into account #02 issue, it's possible to drain the rewards pool in 1 transaction.

Bitcrush team response: this is by design. Logic behind this being that executing compounding for all stakers falls under a user and they would be rewarded accordingly. singleCompound won't charge any fees as it would be for a single user. Solving issue #02 will solve the depletion of rewardPool on a single block.

#05 Staking: emergencyTotalPoolWithdraw() function — High

`emergencyTotalPoolWithdraw()` function transfers all the reward's pool to the owner. This issue regards only the case when the owner's account is not trusted or is not properly secured.

Bitcrush team response: as this version of the pool is temporary and will only last a few weeks, Pool withdrawal will be left intact so rewards may be withdrawn and added to the permanent pool when it is time. The permanent pool will have a timelock. Only use is in this case of an exploit found that would require the owner to be able to pull the reward pool. It is important to note that while the rewards can be claimed in the case of transfer or emergency due to exploit, the staked funds are unable to be accessed by the owner.

#06 Staking: claim() not updating totalPool — High

`claim()` function doesn't update `totalPool` variable making it higher value than actual balance of the contract. It's possible that conflicting values would force the user to withdraw with `emergencyWithdraw()` and lose their rewards.

Update: the issue was fixed.

#07 Staking: unchecked math — High

Unchecked math is used in L58, 112, 214, 266. At least L56 unchecked addition theoretically may overflow `totalPool` as CRUSH token could be minted without cap, see #01.

Update: the issue was fixed.

#08 Staking: compoundAll() gas limit — Medium

`compoundAll()` function uses `for()` loop with unlimited length of processed addresses. With the current value of block gas limit 6e7 this function can handle ~1600 users.

Bitcrush team response: as this pool is temporary and will be live only for a few weeks, we do not see the need to change in the short term. However this has been taken into consideration for the next pool iteration and will be remedied before the next version of the pool is launched.

#09 Staking: data not cleared on leaving — Medium

`leaveStaking()` and `leaveStakingCompletely()` functions clear only the `addressIndexes[]` array but not the `stakings[]` mapping.

Update: the issue was fixed.

#10 Token: empty constructor — Low

No need for an empty constructor in the CRUSHToken contract.

Bitcrush team response: will not redeploy token, but will handle these sort of best practices in future contracts.

#11 Token: tokensBurned variable not needed Low

`tokensBurned` variable could be removed as well as the `MAX_SUPPLY` constant. Initial mint may be performed with the internal `_mint()` function in the constructor.

Bitcrush team response: will not redeploy token, but will handle these sort of best practices in future contracts.

#12 Staking: no explicit visibility Low

`crush` address has no explicit visibility.

Update: the issue was partially fixed. Constants are implicitly public.

#13 Staking: variables naming Low

Multiple variables L11-19 should be written in mixedCase format to conform to Solidity naming conventions [2].

Update: the issue was partially fixed. `divisor` constant should be named UPPERCASE.

#14 Staking: unindexed events Low

Emitted events contain no indexed parameters, which makes it harder to filter them.

Update: the issue was fixed.

#15 Staking: checks on uint <= 0 Low

No need to check uint variables <= 0 as unsigned int cannot be below zero, see L168, 189.

Update: the issue was fixed.

#16 Staking: excessive computations Low

`leaveStakingCompletely()` function performs consecutive updating of the state variables instead of using the local ones for gas savings.

`claim()` function contains useless checking in L208 after the same one from the `getReward()`.

Update: the issue was fixed.

#17 Staking: typos in comments Low

Typo in comment L231.

Update: the issue was fixed.

#18 Staking: converting block number to time Low

Initial value of `EarlyWithdrawFeeTime` variable is set to 1 day but it's used as a number of blocks in 3 days. We recommend to avoid using the 3 seconds to block as a constant because it varies with time.

Update: the issue was partially fixed. 3 seconds to block constant is still used in updated code, but it's used in the explicit form of `blockPerSecond` variable (which should be declared constant and named in UPPERCASE).

Bitcrush team response: frontend will show "approximately 72 hours".

#19 General recommendations Low

We recommend using the fixed pragma version and naming the variables according to Solidity code style. We also recommend to use only one time scale, either in timestamps or block numbers, but not to mix them.

The audited code provided with tests.

Functions lack documentation. We recommend adding NatSpec documentation at least for public and external functions and classes.

Contracts are extremely dependent on the owner's account.

Update: the issue was partially fixed.

Update 2: the deployed version of the BitcrushStaking contract is using old versions of SafeMath and Ownable contracts with Solidity version 0.4.0+ which may not be thoroughly tested with the Solidity version 0.6.2 of the BitcrushStaking staking contract.

Conclusion

7 high severity issues were found. The contracts are highly dependent on the owner's account. Users using the project have to trust the owner and that the owner's account is properly secured.

Audit includes recommendations on the code improving and preventing potential attacks.

Update: Bitcrush Arcade has responded to this report. Most of the issues were fixed. 2 high severity issues remain in the updated code. Individual responses were added after each item in the section. The updated code is located in the same repository after the `4833d0a` commit.
The BitcrushStaking contract is deployed to Binance Smart Chain (BSC): `0x0Ef0626736c2d484A792508e99949736D0AF807e`.

References

1. ERC-20 standard
2. Solidity Docs: Naming Styles

Appendix A. Issues' severity classification

We consider an issue critical, if it may cause unlimited losses or breaks the workflow of the contract and could be easily triggered.

High severity issues may lead to limited losses or break interaction with users or other contracts under very specific conditions.

Medium severity issues do not cause the full loss of functionality but break the contract logic.

Low severity issues are typically nonoptimal code, unused variables, errors in messages. Usually, these issues do not need immediate reactions.

Appendix B. List of examined issue types

Business logic overview

Functionality checks

Following best practices

Access control and authorization

Reentrancy attacks

Front-run attacks

DoS with (unexpected) revert

DoS with block gas limit

Transaction-ordering dependence

ERC/BEP and other standards violation

Unchecked math

Implicit visibility levels

Excessive gas usage

Timestamp dependence

Forcibly sending ether to a contract

Weak sources of randomness

Shadowing state variables

Usage of deprecated code