#HashEx
Blockchain Security

Bitcrush Arcade Bankroll

smart contracts
final audit report

November 2021

hashex.org

contact@hashex.org

Contents

1. 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.

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, then 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. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

2. Overview

HashEx was commissioned by the Bitcrush Arcade Bankroll team to perform an audit of their smart contracts. The audit was conducted between October 21 and October 27, 2021.The code located in @Bitcrush-Arcade/crush_contracts GitHub repository was audited after fe8a04b commit. A recheck was made on commit a827980.

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 for understanding 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.

2.1 Summary

Project name	Bitcrush Arcade Bankroll
URL	http://bitcrusharcade.com
Platform	Binance Smart Chain
Language	Solidity

2.2 Contracts

Name	Address
BitcrushStaking	`0x9D1Bc6843130fCAc8A609Bd9cb02Fb8A1E95630e`

BitcrushBankroll	0xF43A7d04DcD76601dE0B0d03D761B09fBF095502
CRUSHToken	0x0Ef0626736c2d484A792508e99949736D0AF807e
BitcrushLiveWallet	0x5326C45a31DEEBa15EDC68055bF69b2682c9B215

3. Found issues

Critical	1 (3%)
High	13 (37%)
Medium	10 (29%)
Low	9 (26%)
Informational	2 (5%)

BitcrushStaking

ID	Title	Severity	Status
01	Setter methods are not restricted	Critical	Resolved
02	Unrestricted freezing	High	Resolved
03	Admin's extra powers	High	Acknowledged
04	No emergencyWithdraw for users	High	Resolved
05	Bankroll address change	High	Resolved
06	emergencyWithdraw() function problem	High	Resolved
07	Unclear totalPool increase	Medium	Acknowledged

08	Ignored return value of token transfer	Medium	Resolved
09	Wrong rewards calculation	Medium	Resolved
10	autoCompoundLimit is not restricted	Medium	Resolved
11	totalClaimed and .claimedAmount are not updated	Medium	Resolved
12	Wrong reward distribution math	Medium	Resolved
13	Useless approve	Low	Resolved
14	Unused variable	Low	Resolved
15	Implicit visibility modifiers	Low	Resolved
16	Unchecked constructor argument	Low	Resolved
17	Default initialization	Low	Resolved
18	The code does not match the comments	Informational	Resolved

BitcrushBankroll

ID	Title	Severity	Status
01	Setter functions do not restrict input values	High	Resolved

02	Admin's extra powers	High	Resolved
03	StakingPool address change	High	Resolved
04	Function access	High	Acknowledged
05	Ignored return value of token transfer	Medium	Resolved
06	addToBankroll doesn't check negativeBankroll	Medium	Resolved
07	Default initialization	Low	Resolved
08	Immutable variables	Low	Resolved

CRUSHToken

ID	Title	Severity	Status
01	Owner's unlimited access to minting	Medium	Resolved

BitcrushLiveWallet

ID	Title	Severity	Status
01	Ability to change the staking address	High	Acknowledged
02	Locking users' balance by owner	High	Acknowledged
03	Rewards distribution by owner	High	Acknowledged

04	Unchecked input parameters of setters	High	Resolved
05	Unlocked deposit	Medium	Resolved
06	Lack of event Deposit	Low	Resolved
07	Unused variable	Low	Resolved
08	Immutable variable	Informational	Resolved

4. Contracts

4.1 BitcrushStaking

4.1.1 Overview

In this contract, users can stake CRUSH tokens to get rewards. Rewards will be in CRUSH tokens too. There are two types of rewards:

- Daily distribution. Tokens coming from the admin
- Tokens from Bankroll contract

Also, funds in staking can be borrowed by the Bankroll contract. The borrowed amount is taken from users' deposits proportionally. With time this amount will be returned to the stakers by the Bankroll contract. When total borrowed amount is not zero, then users are getting higher daily distribution.

4.1.2 Issues

01. Setter methods are not restricted

Critical ✓ Resolved

The setter method that allows to set values bankroll (L79) and liveWallet (L86) is not restricted. That makes it possible for anyone to set Bankroll and LiveWallet contracts, which can be a vulnerability. Since the LiveWallet receives tokens from the Staking contract during freezing, an attacker can steal all tokens by changing the contract to a vulnerable one.

Recommendation

We strongly recommend restricting access to functions calling for setter functions with the onlyOwner modifier at least.

Update

The issue is fixed. The methods were restricted with the onlyOwner modifier.

02. Unrestricted freezing

- High
- ✓ Resolved

Functions freezeStaking (L355) and unfreezeStaking (L366) can be called by anyone. As a result, rewards can be calculated incorrectly. The function freezeStaking also sends tokens to the LiveWallet contract.

Recommendation

It's recommended to allow calling freeze and unfreeze functions only to the contracts that require this option.

Update

The issue is fixed. The methods are required to be called only by the Bankroll contract.

03. Admin's extra powers

- High
- ⓘ Acknowledged

The function emergencyTotalPoolWithdraw (L404) makes the owner being able to withdraw the whole totalPool.

Recommendation

We do not recommend leaving ways to withdraw users' funds. If it's not possible to avoid contract management by admin, consider using the Compound's Timelock mechanism and a Multisig.

04. No emergencyWithdraw for users

- High
- ✓ Resolved

There is no emergencyWithdraw function for users. Even calling the function leaveStaking cannot ensure users' ability to withdraw their tokens in case of emergency because it tries to call bankroll.recoverBankroll(withdrawalFee); while the bankroll address can be changed by the owner to an arbitrary address.

Recommendation

We strongly recommend adding (restoring) a function that allows users to withdraw their tokens without calls to external contracts. It is needed for the safety of users.

05. Bankroll address change

- High
- Resolved

If the admin changes bankroll address then the users might lose their money. This may happen because the old bankroll contract did not pay off his debt to the staking contract, and the new bankroll contract won't do that instead of the old contract.

Recommendation

We recommend checking totalFrozen before changing bankroll address.

06. emergencyWithdraw() function problem

- High
- Resolved

In some cases emergencyWithdraw() function can fail. The contract takes for the truth that it has enough tokens for reward distribution. But it is not always true. In case the owner sent a low amount in addRewardToPool() function, some users will not be able to withdraw their funds.

Recommendation

In emergencyWithdraw() function withdraw only user's deposit.

07. Unclear totalPool increase

- Medium
- ⓘ Acknowledged

The only function where the totalPool is increased is addRewardToPool function (L95). It's not clear who and why should call it and how they are motivated.

Recommendation

The logic of the value increase should be explained at least.

Bankroll team response

totalPool is used to keep track of available funds to distribute as APY. it is updated by addToReward the pool to add funds to distribute APY by the admin. Once totalPool runs out APY distribution is stopped until funds are topped up again using addToRewardPool.

08. Ignored return value of token transfer

- Medium
- ✓ Resolved

Return values of calls for crush.transfer and crush.transferFrom are ignored.

Recommendation

Handle the returned value or consider using the SafeERC20 library.

Update

Fixed. The library SafeBEP20 is used.

09. Wrong rewards calculation

- Medium
- ✓ Resolved

In the function enterStaking() in case the user already had stakings before calling this function, they get more rewards for their previous stakings. This happens because the variables stakings[msg.sender].lastBlockStaked and stakings[msg.sender].stakedAmount are updated before calling the getReward() function.

Also, another problem with rewards calculation exists. For example, we have 2 users: user1, user2. Following actions occur:

1. user1 stakes 100 tokens
2. For example, 50 blocks are mined
3. user2 stakes 1000 tokens
4. user1 claims the reward

Logically, user1 should claim all the rewards from the first 50 blocks when he was the only user that staked. But in this contract, user1 gets a reward as if user2 has been staking his tokens for the same amount of time as user1.

10. autoCompoundLimit is not restricted

- Medium
- Resolved

The owner can change autoCompoundLimit. If the new autoCompoundLimit variable is too big then compoundAll() won't work because the gas limit becomes larger than the block gas limit.

Recommendation

It's recommended to restrict such values, which can impact the contract's behavior, by some appropriate range of values.

Update

Fixed. Max limit for autoCompoundLimit is set to 30.

11. totalClaimed and .claimedAmount are not updated

- Medium
- Resolved

In the functions leaveStaking(), enterStaking() and singleCompound() the global variables stakings[msg.sender].claimedAmount and totalClaimed are not updated.

Bankroll team response

totalClaimed is used to track historic funds given out as APY, even if a user unstakes or withdraws funds the record of previous APY awarded is maintained. claimedAmount is all-time users' claimed funds. Updated in compoundAll. Leaving staking doesn't change

historic values.

12. Wrong reward distribution math

- Medium
- ✓ Resolved

In the function compoundAll() only a part of users' rewards is updated. During the update, the function considers the current amount of totalStaked tokens. If someone stakes a certain amount of tokens between two compoundAll() calls, the first and the second one will calculate the reward according to different totalStaked.

Bankroll team response

Added variable lastStaking per account to keep track of staked value before claimed amount. Keeping the percentages fixed until the next batch iteration. Applies for both staking and profit reward calculations.

Update

Attempt to fix this issue in lines 138-143 was unsuccessful. It can be passed easily by the double stake. For example, a user stakes 1000 tokens, and right after that stakes 1 more token. After that 1000 of the user's tokens pass through this protection.

13. Useless approve

- Low
- ✓ Resolved

In the line 181 there is a call for approve of crush tokens, which does nothing:

```
crush.approve( address(this), 0);
```

14. Unused variable

- Low ✓ Resolved

Variable profitShare is only set, but never used.

15. Implicit visibility modifiers

- Low ✓ Resolved

Global variables MAX_CRUSH_PER_BLOCK, MAX_FEE and divisor do not have explicit visibility modifiers

16. Unchecked constructor argument

- Low ✓ Resolved

There is no checking the _crushPerBlock argument in the constructor. It shouldn't be bigger than MAX_CRUSH_PER_BLOCK

17. Default initialization

- Low ✓ Resolved

Variables pendingStakedValue, lastAutoCompoundBlock, batchStartingIndex, totalFrozen, accRewardPerShare, accProfitPerShare, totalProfitsClaimed, totalProfitDistributed and autoCompoundLimit are initialized with their default values. So the initialization can be removed in order to optimize gas.

18. The code does not match the comments

- Informational ✓ Resolved

Values of frozenEarlyWithdrawFee and frozenEarlyWithdrawFeeTime variables don't match the comments that refer to them.

4.2 BitcrushBankroll

4.2.1 Overview

This is a contract that is used for collecting funds, which will be used for payout to lottery winners. Missing funds will be borrowed from the staking contract. Upon receipt of funds, the debt to the staking contract will be repaid.

4.2.2 Issues

01. Setter functions do not restrict input values

- High ✓ Resolved

Functions setProfitThreshold, setHouseBankrollShare, setProfitShare, setLotteryShare and setReserveShare don't check input value.

Recommendation

It's strongly recommended to check input values of setters especially the ones that represent shares or percentages.

Update

Function setShares checks if houseBankrollShare + profitShare + lotteryShare + reserveShare equals 100%, but this is wrong because there is burn too, that takes its shares. After setShares function call in transferProfit the function will have wrong values.

02. Admin's extra powers

- High ✓ Resolved

The owner is able to withdraw all tokens (the amount that is represented by the totalBankroll variable), by calling the EmergencyWithdrawBankroll function.

Recommendation

If it's not possible to avoid contract management by admin, consider using the Compound's Timelock mechanism and a Multisig.

Update

Fixed. The function was removed

03. StakingPool address change

- High ✓ Resolved

If the owner changes stakingPool address then users that staked tokens in the old staking contract might lose their tokens in case negativeBankroll is positive. Because the BitcrushBankroll contract will not be able to return the tokens.

Recommendation

We recommend checking negativeBankroll variable before changing stakingPool address.

Update

The issue was fixed. The variable stakingPool is defined as immutable and only is set in the constructor.

04. Function access

- High ⓘ Acknowledged

The function payOutUserWinning() should be restricted to liveWallet address only. The owner can give access to this function to EOA and this account gets the opportunity to withdraw all tokens from BitcrushBankroll and BitcrushStaking contracts.

Recommendation

Remove a check that msg.sender is authorizedAddresses and add a check that msg.sender is liveWallet.

05. Ignored return value of token transfer

- Medium ✓ Resolved

Return values of calls for crush.transfer and crush.transferFrom are ignored.

06. addToBankroll doesn't check negativeBankroll

- Medium
- Resolved

Functions payOutUserWinning and addUserLoss handle a case when the Bankroll account does not have enough tokens to pay a winner. The function payOutUserWinning saves the remaining in the negativeBankroll variable and marks the contract as depleted(poolDepleted becomes true). The addUserLoss handles the values when receiving tokens. At the same time, addToBankroll receiving tokens doesn't handle them.

Recommendation

We recommend checking whether the value of negativeBankroll is greater than 0 each time when the contract calls the function transferFrom of the token.

Update

Fixed. The function addToBankroll handles values negativeBankroll and poolDepleted.

07. Default initialization

- Low
- Resolved

The variables poolDepleted, profitThreshold, totalWinnings and totalProfit are initialized with their default values. So the initialization can be removed in order to optimize gas.

08. Immutable variables

- Low ✓ Resolved

Variables crush, reserve and lottery can be immutable.

Update

Only variable crush has been set as immutable.

4.3 CRUSHToken

4.3.1 Overview

The main token that will be used in this system.

4.3.2 Issues

01. Owner's unlimited access to minting

- Medium ✓ Resolved

The owner is able to mint CRUSH-tokens to any address. It makes the owner overpowered and the tokens distribution centralized.

Recommendation

We recommend considering using Compound's Timelock mechanism and the Multisig to avoid the centralized tokens distribution.

Update

Deployed to 0x0Ef0626736c2d484A792508e99949736D0AF807e CRUSH token's ownership was already renounced.

4.4 BitcrushLiveWallet

4.4.1 Overview

In this contract, users can make bets and participate in the lottery.

4.4.2 Issues

01. Ability to change the staking address

- High ⓘ Acknowledged

The Owner can set the stakingPool variable to EOA through the setStakingPool() function and after that increase balance by calling the addToUserWinnings() function. With these actions, they can get access to the users' tokens. Also, this can be achieved by changing the bankroll variable to EOA.

Recommendation

It's recommended to not leave the ability to change such values to the only owner. These values can severely impact the contract's behavior. If it's not possible to avoid contract management by the admin, consider using Compound's Timelock mechanism and a Multisig.

02. Locking users' balance by owner

- High ⓘ Acknowledged

The owner is able to lock balances for any user of the LiveWallet contract by calling updateBetLock function.

Recommendation

We do not recommend leaving such ability to anyone since it makes the contract centralized.

03. Rewards distribution by owner

- High ⓘ Acknowledged

Only the owner is able to call functions registerLoss and registerWin, that distribute users' rewards and losses. The functions also receive parameter values, which the users won or lost. So it's not calculated on-chain, only the owner decides who wins or loses and how many tokens.

Recommendation

We recommend implementing such calculations within smart contracts.

04. Unchecked input parameters of setters

- High ✓ Resolved

There are no checks of input arguments in setter functions setLockPeriod() and setEarlyWithdrawFee().

Recommendations

We recommend checking such values since withdrawing is depended on that.

05. Unlocked deposit

- Medium ✓ Resolved

Deposit via function addbetWithAddress is not locked and a user, for whom the deposit was created, is able to withdraw tokens right after depositing.

Recommendation

It's recommended to set the value of .lockTimeStamp to betAmounts as it's done at L50 in addbet function.

06. Lack of event Deposit

- Low ✓ Resolved

In function addbetWithAddress the event Deposit is not emitted.

Recommendation

Emit the event if the function is considered as a way to deposit.

07. Unused variable

- Low ✓ Resolved

The state variable lossBurn and the function setLossBurn() are redundant. So, they can be removed in order to save gas.

08. Immutable variable

- Informational ✓ Resolved

The state variable crush can be immutable in order to save gas.

5. Conclusion

The audited contracts strongly depend on the owner's behavior. The contracts contain lots of High severity issues. It's not recommended for deployment until all these issues are fixed.

The Bitcrush Arcade team has responded to the initial version of this report. They confirm that the high level of system centralization was the intended design. Most of the issues were resolved in the updated contracts.

The audited repository almost doesn't contain tests. We strongly recommend adding tests for other contracts to ensure that the contracts work as intended. The recommended coverage level is 90% at least.

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

The contracts are deployed to the mainnet of Binance Smart Chain:

0x0Ef0626736c2d484A792508e99949736D0AF807e CRUSHToken,

0x9D1Bc6843130fCAc8A609Bd9cb02Fb8A1E95630e BitcrushStaking,

0xF43A7d04DcD76601dE0B0d03D761B09fBF095502 BitcrushBankroll,

0x5326C45a31DEEBa15EDC68055bF69b2682c9B215 BitcrushLiveWallet.

CRUSHToken ownership was renounced, the other 3 contracts had the same owner by Nov'24 2021.

Appendix A. Issues' severity classification

Critical. Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.

High. Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.

Medium. Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.

Low. Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.

Informational. Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.

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

 contact@hashex.org

 @hashexbot

 blog.hashex.org

 linkedin

 github

 twitter

HashEx
Blockchain Security