HashEx

Blockchain Security

GummyBull

smart contracts
audit report

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

2 Overview

HashEx was commissioned by theGummyBull team to perform an audit of their smart contracts.

The audited code is located in the Github repository https://github.com/gummybull/gummies.contract after the commit 8842d7c. Contracts in the audited scope are: GummyPotV2 and FlavorTeams. Audit was conducted between September 12 and September 21, 2021.

Recheck was done after the commit f7e1a47.

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.

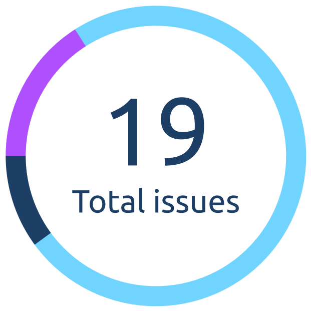
2.1 Summary

Project name	GummyBull
URL	https://gummybull.io/
Platform	Binance Smart Chain
Language	Solidity

2.2 Contracts

Name	Address
GummyPotV2	0xBCa070b271baDB2b3D92a4Fc5F2da742b1E2D4B1
FlavorTeams	0xDD6f2e452574332F1465A6d87d479BeBE3eB1E26

3 Found issues



■	Medium	3 (16%)
■	Low	14 (74%)
■	Info	2 (10%)

GummyPotV2

ID	Title	Severity	Status
01	Partner buyback functionality is susceptible to frontrun attacks	■ Medium	Acknowledged
02	DoS attack by buying tickets	■ Low	Acknowledged
03	Lacks of checks on input parameters	■ Low	Acknowledged
04	Require not-zero amount to participate	■ Low	Fixed
05	Skip the execution of requestToCloseCurrentRound if no participants	■ Low	Fixed

06	Lack of events for changing important values	Low	Fixed
07	Define startedOnDay as uint40 and repack the structure	Low	Fixed
08	Methods should be declared external	Low	Fixed
09	Make utcOffset immutable	Low	Acknowledged
10	Return values of ERC20 token transfers is not checked	Low	Fixed
11	Variables naming style	Info	Fixed

FlavorTeams

ID	Title	Severity	Status
01	Closing request can be sent to a round with no participants	Medium	Fixed
02	Partner buyback functionality is susceptible to frontrun attacks	Medium	Acknowledged
03	Methods should be declared external	Low	Fixed
04	Make utcOffset immutable	Low	Acknowledged
05	Return values of ERC20 token transfers is not checked	Low	Fixed

06	Lack of events for changing important values	■ Low	Fixed
07	Define startedOnDay as uint40 and repack the structure	■ Low	Fixed
08	Variables naming style	■ Info	Fixed

4 Contracts

4.1 GummyPotV2

4.1.1 Overview

A contract where a user can buy tickets for Gummy token for participating in lottery.

4.1.2 Issues

01. Partner buyback functionality is sucseptible to frontrun attacks

▪ Medium ⓘ Acknowledged

Function _partnerBuyBackAndBurn has no checks on slippage on conversion Gummy tokens to partner tokens.If a significant amount is going swapped it creates incentives for frontrunning the transaction.Imagine the function swaps 1000 Gummy -> 10000 WBNB -> 10 partner -> BURN.

If somebody front-runs this transaction, manipulates the pool and lowers the price:

1000 gummy -> 100 WBNB -> 0.1 partner -> BURN.

Then stablizes the pool back, the manipulator ends up with 99% amount of gummy instead of them being burned.

Also the value of the deadline set to block.timetamp + 600 does not in fact check the deadline as as soon as the transaction is mined the router checks that block.timestamp + 600 >= block.timestamp.

```
function _partnerBuyBackAndBurn(uint256 amountPartnerToBurn) private {
    if(amountPartnerToBurn > 0){
        address[] memory _path = new address[](3);

        _path[0] = address(gummyToken);
        _path[1] = WBNB;
        _path[2] = partnerToken;

        gummyToken.approve(address(router), amountPartnerToBurn);
        router.swapExactTokensForTokensSupportingFeeOnTransferTokens(
            amountPartnerToBurn,
            0,
            _path,
            burnAddressPartner,
            block.timestamp + 600
        );
    }
}
```

Team response

It is absolutely correct that this creates a front running attack vector, either on GUMMY or BANANA. However, because the buyback and burn is fragmented (it happens on each lottery ticket purchase) and only on 10% of the amount, we do not expect this to be an issue (amounts should mostly be small). Also, the 5% transaction fee applied on every GUMMY transaction makes these attacks on the GUMMY side pretty unlikely to be profitable. Also, many reflect tokens such as BabyCake use similar mechanisms where a small percentage (7% in the case of BabyCake) is used to swap against ETH without any slippage check. Because this attack vector is pretty unlikely in my opinion, I do not think it would need to be addressed further.

02. DoS attack by buying tickets

- Low ⓘ Acknowledged

The contract is susceptible to a DoS atack. A malicious actor may buy tickets many times with function iWantToGetSpammed(). Then on closing the round function _drawWinner() may run out of block gas limit. This will lead to a situation when the round is not closed, and the contract is effectively broken. The rewards are not distributed to the winner and the new round doesn't start.

Recommendation

Cap the maximum number of participants in the lottery that can buy tickets in one round so gas usage won't exceed the block gas limit.

Team response

1. findUpperBound has a log(n) complexity, where n is the number of participants. This means that the gas cost will increase logarithmically in the number of participants.

2. findUpperBound uses an array that is in storage which reduces significantly the gas usage.

I conducted a gas analysis on the findUpperBound function, here are a few numbers on the gas used per number of participants.

- 65,500 participants: 64,441 gas used,

- 115,500 participants: 66,874 gas used,

- 365,500 participants: 71,740 gas used.

The gas used increases indeed logarithmically in the number of participants. To fix ideas, with ~10,000,000,000 participants (10 billion), the gas used would be ~130k, which is still well within gas limit. Given that a call to iWantToGetSpammed costs ~0.4 USD, 10 billion calls would cost ~10B * 0.4 = USD 4B.

03. Lacks of checks on input parameters

▪ Low ⓘ Acknowledged

The constructor of the GummyPotV2 contract lacks zero checks on input parameters.

Recommendation

Add checks require(_address != address(0), "ZERO_ADDRESS"); for address parameters

Team response

The contract is deployed by a script that is ensuring all parameters are set correctly. Adding additional checks would cost more gas (though only once) and does not appear to be necessary

04. Require not-zero amount to participate

▪ Low ✓ Fixed

The function iWantToGetSpammed() can be called with zero amount.

Recommendation

Require non-zero amount to participate by checking amount > 0 in function iWantToGetSpammed()

05. Skip the execution of requestToCloseCurrentRound if no participants

▪ Low ✓ Fixed

Gas can be saved if execution is skipped if there are no participants in the round.

06. Lack of events for changing important values

▪ Low ✓ Fixed

Functions setBurnNumerator(), setUtcOffset(), setBurnPartnerNumerator() lack events for important settings change.

Recommendation

Emit events in the specified functions

07. Define startedOnDay as uint40 and repack the structure

■ Low ✓ Fixed

Less bytes for days can be used. The fields in the struct startedOnDay, closingRequested, isClosed can be packed to one storage slot to save gas.

08. Methods should be declared external

■ Low ✓ Fixed

Everywhere where method is used only externaly (never internaly) it's better to set the modifer to external not public to save gas.

Recommendation

Make functions pause() and unpause() external

09. Make utcOffset immutable

▪ Low ⓘ Acknowledged

Changing the offset back and forward is contrintuitive and may lead to the contract misusing, because the today() may go back and forward, so conditions like at L265

```
rounds[currentRound].startedOnDay != today()
```

could be violated, in this situation, the owner can finish the round earlier than it is expected by users.

Recommendation

It's better and safer to define the timezene offset immutable and deploy a new contract for a new timezone.

Team response

The case where the offset would be changed and we would go back in time poses no issue. Two cases exist: 1. if the offset is increased, there is 0 issue, 2. if the offset is decreased, there is no problem either, it might cause two rounds to have the same startedOnDay (which is not a problem) but all tickets will always be processed correctly. Though I agree it might be somehow counterintuitive, I prefer to leave this function here because this contract redeployment is costly in terms of frictions. That being said, note that changing UTC offset would be a very rare occurrence.

10. Return values of ERC20 token transfers is not checked

■ Low ✓ Fixed

The result of token transfers is not checked (L206, L245, L292, L306, L340).

Recommendation

Use OpenZepelin's SafeERC20 library or check return success status explicitly.

11. Variables naming style

■ Info ✓ Fixed

burnAddressPartner field should be written in upper case to conform to Solidity naming conventions.

4.2 FlavorTeams

4.2.1 Overview

A lottery contract where a user can choose one of four teams and send Gummy tokens to participate. If the chosen team wins, the user will get his part of the reward.

4.2.2 Issues

01. Closing request can be sent to a round with no participants

▪ Medium ✓ Fixed

The function yumYum() does not check if a round has any participants when sending request to close it in L245. This breaks the function _drawWinningFlavor() which is called on closing the round.

02. Partner buyback functionality is susceptible to frontrun attacks

▪ Medium ⓘ Acknowledged

Function _partnerBuyBackAndBurn has no checks on slippage on conversion Gummy tokens to partner tokens. If a significant amount is going swapped it creates incentives for frontrunning the transaction.

Team response

it is absolutely correct that this creates a front running attack vector, either on GUMMY or BANANA. However, because the buyback and burn is fragmented (it happens on each lottery ticket purchase) and only on 10% of the amount, we do not expect this to be an issue (amounts should mostly be small). Also, the 5% transaction fee applied on every GUMMY transaction makes these attacks on the GUMMY side pretty unlikely to be profitable. Also, many reflect tokens such as BabyCake use similar mechanisms where a small percentage (7% in the case of BabyCake) is used to swap against ETH without any slippage check. Because this attack vector is pretty unlikely in my opinion, I do not think it would need to be addressed further.

03. Methods should be declared external

- Low ✓ Fixed

Everywhere where method is used only externaly (never internaly) it’s better to set the modifer to external not public to save gas.

Recommendation:

Make functions pause() and unpause() external

04. Make utcOffset immutable

- Low ⓘ Acknowledged

Changing the offset back and forward is contrintuitive and may lead to the contract misusing, because the today() may go back and forward.

Recommendation

It’s better and safer to define the timezene offset immutable and deploy a new contract for a new timezone.

Team response

The case where the offset would be changed and we would go back in time poses no issue. Two cases exist: 1. if the offset is increased, there is 0 issue, 2. if the offset is decreased, there is no problem either, it might cause two rounds to have the same startedOnDay (which is not a problem) but all tickets will always be processed correctly. Though I agree it might be somehow counterintuitive, I prefer to leave this function here because this contract redeployment is costly in terms of frictions. That being said, note that changing UTC offset would be a very rare occurrence.

05. Return values of ERC20 token transfers is not checked

- Low ✓ Fixed

The result of token transfers is not checked (L260, L307, L339).

Recommendation

Use OpenZepelin's SafeERC20 library or check return success status explicitly.

06. Lack of events for changing important values

- Low ✓ Fixed

The function setUtcOffset() should emit an event on changing an important value in the contract.

Recommendation

Emit OnUtcOffsetChanged(newOffset) event

07. Define startedOnDay as uint40 and repack the structure

▪ Low ✓ Fixed

Less bytes for days can be used. The fields in the struct startedOnDay, closingRequested, isClosed can be packed to one storage slot to save gas.

08. Variables naming style

▪ Info ✓ Fixed

Variables numTeams, partnerTeam, partnerTeamBis, burnNumerator, partnerTeamBurnNumLoss, partnerTeamBurnNumWin, partnerTeamBurnNumWinBis, burnAddressPartner should be written in upper case to conform to Solidity naming conventions.

5. Conclusion

3 medium severity issues were found. The contracts are well documented and tested.The audit includes recommendations on the code improvements and preventing potential attacks.

Update: all issues were addressed by the team after audit (either fixed or responded). Individual responses are added below the issues.

The updated contracts are deployed to Binance Smart Chain (BSC) mainnet:
0xBCa070b271baDB2b3D92a4Fc5F2da742b1E2D4B1 (GummyPotV2),

0xDD6f2e452574332F1465A6d87d479BeBE3eB1E26 (FlavorTeams)

6. 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.

7. Appendix B

- 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

HashEx
Blockchain Security