HashEx

RugZombie

smart contracts audit report

Prepared for:

rugzombie.io

Authors: HashEx audit team

September 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 RugZombie team to perform an audit of their smart contracts. The audit was conducted between August 16 and August 19, 2021.

The audited contracts are deployed to the Binance Smart Chain (BSC):
0x50ba8BF9E34f0F83F96a340387d1d3888BA4B3b5 ZombieToken,
0x590Ea7699A4E9EaD728F975efC573f6E34a5dC7B DrFrankenstein.
The documentation can be found on the team's gitbook.
At the time of writing the report, ZombieToken is owned by DrFrankenstein which is owned by the wrapper contract under the 6 hours Timelock contract with an externally owned account (EOA) as an admin.

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: the RugZombie team has responded to this report. Individual responses were added after each item in the section. The updated code is located in the github repository @rug-zombie/rug-zombie-contracts after the 1e1b0fa commit. SafeOwner wrapper contract's ownership was transferred to the GoodZombie contract:
0xAe689a88bEe2E25E098Dd38970582c096fAbbB08 GoodZombie.

Contracts overview

ZombieToken

A copy of SushiToken [1]. Implementation of ERC20 token standard with the minting open for the owner and governance from Yam Finance [2] (fork of Compound Governance Alpha, audit available [3]).

DrFrankenstein

The Staking contract is similar to MasterChef by SushiSwap [4] (the audit of which is available [5,6]) with multiple modifications.

Found issues

ID	Title	Severity	Response
01	ZombieToken: delegates not transferred	High	Responded
02	ZombieToken: mint without creating delegators	High	Responded
03	DrFrankenstein: no checks on input data in addPool()	High	Responded
04	DrFrankenstein: emergencyWithdraw() with pid=0 does not burn undeadTokens	High	Responded
05	DrFrankenstein: wrong router breaks unlock() and withdrawEarly()	High	Fixed
06	DrFrankenstein: emission rate not limited	High	Fixed
07	DrFrankenstein: priceConsumer contract uses unverified library	Medium	Fixed
08	DrFrankenstein: massUpdate is optional	Medium	Responded
09	DrFrankenstein: allowed duplicated pools	Medium	Fixed
10	DrFrankenstein: tokens with transfer fees are not supported	Medium	Responded
11	DrFrankenstein: migrator can be updated in frontrun	Medium	Fixed
12	DrFrankenstein: depositRug() parameters	Medium	Fixed
13	DrFrankenstein: migrator is set to 0	Medium	Fixed
14	DrFrankenstein: input data not filtered	Medium	Fixed
15	DrFrankenstein: sending BNB via .transfer()	Medium	Responded
16	Multiple: outdated OpenZeppelin contracts	Medium	Responded
17	ZombieToken: whitelist can’t be undone	Low	Responded
18	ZombieToken: treasury & lpStorage are GnosisSafeProxy	Low	Acknowledged

19	DrFrankenstein: declaring vars with 0 assignment	Low	Acknowledged
20	DrFrankenstein: naming variables	Low	Acknowledged
21	DrFrankenstein: addGrave() should deploy new token	Low	Acknowledged
22	DrFrankenstein: leaveStaking() requires holding undead tokens	Low	Acknowledged
23	General recommendations & gas optimizations	Low	Acknowledged

#01 ZombieToken: delegates not transferred High

`_moveDelegates()` function in L200 of ZombieToken contract is designed to be used with every token transfer. However, in ZombieToken it's called with minting new tokens, but not with the usual transfers. The origin (SushiToken) has a warning (L8) about this issue. The issue allows minting any number of delegation votes. A test for minting delegation votes can be seen in Appendix C.

Recommendation: stick with the original governance logic and move delegates with transfers or disable governance functionality. At the time of writing this report, the DrFrankenstein contract owns ZombieToken and mints rewards and delegators to the dev address and to the UndeadBar contract.

Team response: this was a result of us forking Pancakeswap, we will not use the ZombieToken contract for governance as the logic is flawed.

#02 ZombieToken: mint without creating delegators High

The `mint()` function of BEP20 contract L212 does not create the voting power for the address.

Recommendation: stick with the original governance logic and move delegates with transfers or disable governance functionality. At the time of writing this report, the DrFrankenstein contract owns ZombieToken and mints rewards and delegators to the dev address and to the UndeadBar contract.

Team response: this was a result of us forking Pancakeswap, we will not use the ZombieToken contract for governance as the logic is flawed.

#03 DrFrankenstein: no checks on input data in addPool() — High

The `addPool()` function L195 should check the input `lpToken` address to be the real pair address of the current swap router. Otherwise, `unpairBurnAndTreasureLP()` may fail permanently for that pool.

Recommendation: we recommend filtering input data wherever is possible.

Team response: we do not want to restrict our contract to supporting LP's from only one router as we have pools for Apeswap and Pancakeswap LP's. In hindsight we would have included better multi-dex support but through UI changes, pools containing non-native LP's are still functional to our users.

#04 DrFrankenstein: emergencyWithdraw() with pid=0 does not burn undeadTokens — High

The `emergencyWithdraw()` function at L571 doesn't burn the corresponding amount of `undead` tokens when called with zero pid. It opens the possibility to mint an unlimited amount of undead tokens, just like the Cake SYRUP incident [7].

Recommendation: we recommend informing the users and trading platforms that undead tokens are worth nothing and should not be traded.

Team response: this was a result of us forking Pancakeswap, we will not use the ZombieToken contract for governance as the logic is flawed.

#05 DrFrankenstein: wrong router breaks unlock() and withdrawEarly() — High

The `setPancakeRouter()` function L679 may be used by the owner to update the swap router address without checking the new parameter. Setting the wrong `pancakeswapRouter` variable may cause locked users' ruggedTokens because `unlock()` function will be reverted. At the time of writing this report DrFrankestein contract is controlled via 6 hours Timelock.

Recommendation: we recommend not to change the router address as the migrator variable is set to zero address and can't be changed for the deployed contract. For the contract's update we recommend checking the new router address with any specific view function before setting.

Team response: this is addressed in our GoodZombie contract. The `setPancakeRouter()` function can now only be called on routers addresses that are whitelisted by our team gnosis multisig wallet. Under the event where the contract owner is compromised, only whitelisted routers could be set.

Update: the issue was fixed.

#06 DrFrankenstein: emission rate not limited — High

The `updateMultiplier()` function L185 is used for updating the `BONUS_MULTIPLIER` 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, an attacker can set an arbitrary big value for the `BONUS_MULTIPLIER` variable. In such a case the number of tokens till cap will be minted soon and the token price will drop. Moreover, it may mess up the rewards if called without the `massUpdatePools()` function.

Recommendation: limit the `BONUS_MULTIPLIER` parameter and call the `updateMultiplier()` function only with `massUpdatePools()`.

Team response: the GoodZombie contract restricts `BONUS_MULTIPLIER` from being set greater than 5. `massUpdatePools()` is also now called within `updateMultiplier()`.

Update: the issue was fixed.

#07 DrFrankenstein: priceConsumer contract uses unverified library — Medium

The `priceConsumer` variable leads to the PriceConsumerV3 contract with the unverified Percentages library.

Update: the issue was fixed.

#08 DrFrankenstein: massUpdate is optional — Medium

`addPool()`, `addGrave()` and `set()` functions have optional `_withUpdate` flag which calls `massUpdatePools()` if set true and may cause unfair rewards in case of rarely updated pools [8].

Team response: `_withUpdate()` is intentionally left optional for the `addPool()`, `addGrave()` and `set()` functions as we often call these functions in bulk, then call `massUpdatePools()` after the last transaction. Since we have a lot of pools, calling `massUpdatePools()` with every transaction costs a lot of gas.

#09 DrFrankenstein: allowed duplicated pools — Medium

`addPool()` and `addGrave()` functions allow adding a new pool with the lpToken of the existing pool. It would mess up the rewards and it won't be possible to edit or remove the wrong pools.

Team response: fixed in the GoodZombie contract. The `addPool()` and `addGrave()` functions can now only create pools containing non duplicate lpTokens.

Update: the issue was fixed.

#10 DrFrankenstein: tokens with transfer fees are not supported — Medium

DrFrankenstein contract doesn't support pools of tokens with fees on transfers as transfers aren't checked for the resulting balance.

Team response: we don't plan to create pools for tokens with transfer fees.

#11 DrFrankenstein: migrator can be updated in frontrun — Medium

The `setMigrator()` function can be used by the owner to frontrun changing the migrator contract address when a user calls `migrate()` function. The deployed version of the code doesn't allow to change the zero migrator address as the ownership is transferred to the SafeOwner contract that can't transfer it further.

Team response: this is not an issue as DrFrankenstein is owned by a SafeOwner contract and `setMigrator()` can no longer be called.

Update: the issue was fixed.

#12 DrFrankenstein: depositRug() parameters — Medium

The `depositRug()` function L622 could be called with `pid=0` and doesn't check the input amount of tokens to be transferred. The `isUnlocked()` modifier performs an excessive check in L161 and `user.rugDeposited` values are not in use.

Team response: pool `pid=0` is being retired. In addition to this, calling `depositRug()` on `pid=0` will fail as the `pool.rugToken` is set to the 0 address, making it not a threat to those who do this.

Update: the issue was fixed.

#13 DrFrankenstein: migrator is set to 0 — Medium

The `migrator` variable is set to zero address and can't be changed for the deployed contract because the ownership is transferred to the SafeOwner contract that has no implementation for updating the migrator address in DrFrankenstein.

Team response: this was intentional, we patched the migrator with our SafeOwner contract as we don't plan to use it in our project. Users will withdraw their funds manually during a migration. `emergencyWithdraw()` will always be a safe means for users to withdraw their funds so the migrator is not a necessity.

Update: the issue was fixed.

#14 DrFrankenstein: input data not filtered Medium

System variables can be updated by the owner by calling `setGraveNft()`, `setUnlockFee()`, `setPriceConsumer()`, `setPancakeRouter()` functions L657-681. Setting the wrong values might break withdrawals and deposits. However, `emergencyWithdraw()` doesn't depend on these parameters.

Team response: the `setPriceConsumer()` and `setPancakeRouter()` input is now filtered by the GoodZombie contract, only accepting addresses whitelisted by our team multisig.

A filter on `setUnlockFee()` was also added, adding a minimum unlock fee requirement. Our team is aware that small unlock fees can round to 0 when converting to BNB price, we will switch to a more precise priceConsumer contract before using a small unlock fee on a pool.

The `setgraveNft()` function now requires the nft contract to return true to the `hasReviveRug()` function before changing a grave's NFT.

Update: the issue was fixed.

#15 DrFrankenstein: sending BNB via .transfer() Medium

The recommended way for sending ETH/BNB is to use `.call{value}` with the ReentrancyGuard. Only this method is gas customizable.

Team response: since our contract only stores BNB intermittently within the `unlock()` function, using the `transfer()` function creates no risk to our users.

#16 Multiple: outdated OpenZeppelin contracts Medium

Both ZombieToken and DrFrankenstein use third-party contracts with changed pragma versions, i.e. code of OpenZeppelin v3 is compiled with 0.8.4 compilator. Such discrepancies may cause unpredictable errors and should be avoided by importing the corresponding versions without modifications.

Team response: our contract is a pancakeswap fork refactored to solidity 0.8.4, we also refactored the OpenZepplin dependencies to 0.8.4 and ensured the contract works as expected through tests.

#17 ZombieToken: whitelist can't be undone Low

The `whitelistAddress()` function L275 allows the owner to whitelist address to bypass `whaleDetection()` check and can't be reversed if the wrong address was whitelisted.

Team response: the whitelist is a feature that was only used for the first 30 minutes after launch. For context our token had a whale detection feature preventing wallets from holding 2% of our total supply during launch. This was to prevent bots from buying and dumping the token at launch

and was permanently lifted 30 minutes afterwards. The whitelist was necessary at the time because some addresses needed to be exempt from this rule (Eg. LP Contract, Pancakeswap router, treasury and founder multisig wallets), but is no longer used as the feature was only active during launch.

#18	ZombieToken: treasury & lpStorage are GnosisSafeProxy	Low

`treasury` and `lpStorage` variables lead to the unverified GnosisSafeProxy contracts so fees may be locked if these contracts were deployed with mistakes.

#19	DrFrankenstein: declaring vars with 0 assignment	Low

Declaring variables with simultaneous 0 assignment spends more gas than default declaring with the same result.

#20	DrFrankenstein: naming variables	Low

`BONUS_MULTIPLIER` can be changed and therefore should be named in mixedCase.

#21	DrFrankenstein: addGrave() should deploy new token	Low

To avoid pool duplication, the `addGrave()` function could deploy its own token each time it is called.

#22	DrFrankenstein: leaveStaking() requires holding undead tokens	Low

Minting undead tokens in enterStaking() implies that users must keep them until they leave the staking pool. But the possible transfers of the undead tokens may cause users' frustration if they lose control over them. Documentation should emphasize the significance of these tokens.

#23	General recommendation & gas optimizations	Low

ZombieToken: `whaleDetection` should check the launch status prior to reading from storage.
DrFrankenstein: `zombie`, `undead`, `treasury`, `lpStorage`, `zombiePerBlock`, `startBlock` and `burnAddr` variables should be declared constants/immutable.
DrFrankenstein: repetitive reads from storage in withdrawals and deposits should be avoided by using local variables.
DrFrankenstein: `leaveStaking()` reads `isGrave` variable when it's always true for the first pool.

DrFrankenstein: excessive require statements in L377, L428 because the safe subtraction `user.amount-_amount` performed takes place in the next line.
DrFrankenstein: `isUnlocked()` modifier reads all the structure parameters of `userInfo[gid][msg.sender]` and `poolInfo[_gid]` instead of reading 4 variables.
General lack of events: the `whitelistAddress()` function in ZombieToken and zero custom events for changing parameters in DrFrankenstein.

Conclusion

6 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.

We strongly recommend using unmodified OpenZeppelin contracts via import and adding tests with coverage of at least 90%, before the deployment to the mainnet.

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

The audited contracts are deployed to the Binance Smart Chain (BSC):
`0x50ba8BF9E34f0F83F96a340387d1d3888BA4B3b5` ZombieToken,
`0x590Ea7699A4E9EaD728F975efC573f6E34a5dC7B` DrFrankenstein.

Update: the RugZombie team has responded to this report. Individual responses were added after each item in the section. The updated code is located in the github repository @rug-zombie/rug-zombie-contracts after the 1e1b0fa commit. 2 high severity issues were fixed by transferring SafeOwner's ownership to the GoodZombie contract which provides the safety guards for the DrFrankenstein contract's functions:
`0xAe689a88bEe2E25E098Dd38970582c096fAbbB08` GoodZombie.

References

1. SushiToken on GitHub
2. YAMGovernance on github
3. Compound Alpha audit by OpenZeppelin
4. SushiSwap's MasterChef contract
5. SushiSwap audit by PeckShield
6. SushiSwap audit by Quantstamp
7. Update on the SYRUP Incident
8. Dracula Protocol Medium

Appendix A. Issues' severity classification

We consider an issue to be critical, if it may cause unlimited losses, or break 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

Appendix C. Delegation votes minting test

The code of the test in hardhat framework

```
describe('Delegation power attack', async function () {
  it('Alice generates voting power for Carol', async function()  {
    await zombieToken.liftLaunchWhaleDetection();
    await zombieToken.mint(deployer.address, 100);
    await zombieToken.delegate(carol.address);
    await zombieToken.transfer(alice.address, 100);

    await zombieToken.connect(alice).delegate(carol.address);
    await zombieToken.connect(alice).transfer(bob.address, 100);
    await zombieToken.connect(bob).delegate(carol.address);
    const votes = await zombieToken.getCurrentVotes(carol.address);
    console.log('Carol’s votes', votes.toString())
  })
});
```

The test output (Carol gets voting power of 300):

```
HairToken
  Alice generates voting power for Carol
    Carol’s votes  300
✓ should multiply delegation power on transfers
```