STAYSAFU
AUDIT

APRIL 15TH, 2022

GRIZZLY ROCKET

TABLE OF CONTENTS

AUDIT SUMMARY

This report was written for Grizzly Rocket (GRR) in order to find flaws and vulnerabilities in the Grizzly Rocket project's source code, as well as any contract dependencies that weren't part of an officially recognized library.

A comprehensive examination has been performed, utilizing Static Analysis, Manual Review, and Grizzly Rocket Deployment techniques. The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors
- Assessing the codebase to ensure compliance with current best practices and industry standards
- Ensuring contract logic meets the specifications and intentions of the client
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders
- Through line-by-line manual review of the entire codebase by industry expert

AUDIT OVERVIEW

PROJECT SUMMARY

Project name	GRIZZLY ROCKET
Description	Grizzly rocket is an ERC20 token. It aims to become a whole ecosystem, including staking and scanning features.
Platform	Binance Smart Chain
Language	Solidity
Codebase	https://pastebin.com/9Fc2Yvbf

FINDINGS SUMMARY

Vulnerability	Total
Critical	0
Major	0
Medium	2
Minor	5
Informational	3

EXECUTIVE SUMMARY

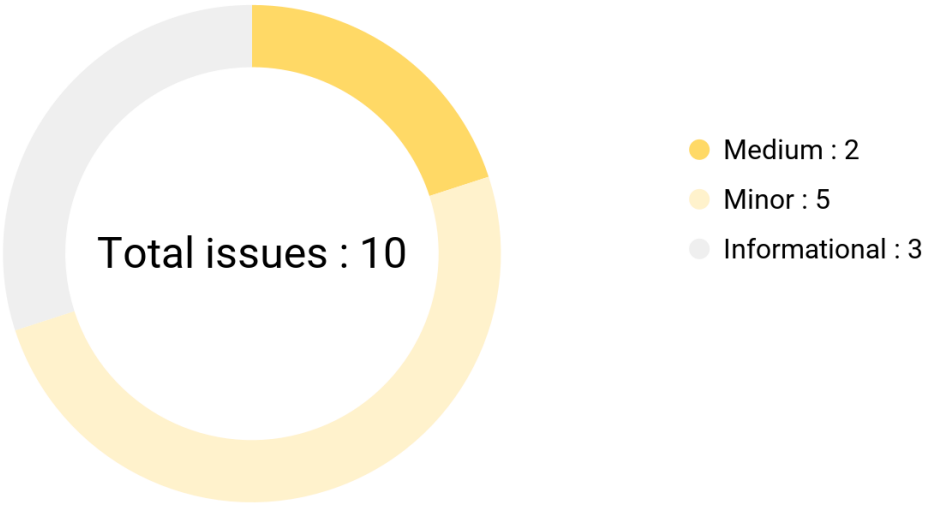
Grizzly rocket is an ERC20 token. It includes reflection, presale and vault mechanisms, and will be present on Pancakeswap. When the project is completed, owners will be able to stake their tokens to earn money, to vote together to choose the next ecosystem project, to have vip access to the beta of the grizzly p2e earn and to win giveaways every week.

Transactions are taxed as follows:

- Buying is taxed by 4% for rewards
- Buying is taxed by 4% for marketing
- Buying is taxed by 9% for liquidity mechanism
- Selling is taxed by 5% for rewards
- Selling is taxed by 5% for marketing
- Selling is taxed by 12% for liquidity mechanism

There have been no major or critical issues related to the codebase and all findings listed here are minor or informational. The major security problems are the dependence on a decentralized exchange platform and the centralization of privileges. The code should be standardized to meet industry standards.

AUDIT FINDINGS



Code	Title	Severity
CENT-1	Centralization of major privileges	Medium
EXT-1	External protocol dependencies	Medium
BLOC-1	Usage of block.timestamp	Minor
THRE-1b	Missing threshold for minor func	Minor
THRE-1a	Missing threshold for minor func	Minor
THRE-1c	Missing threshold for minor func	Minor
COMP-1	Unfixed version of compiler	Minor
UINT-1	Wrong uint type	Informational

MSG-2	Missing error message	Informational
MSG-3	Too long error message	Informational

MSG-2 | Missing error message

Description

Some require statements of the smart contract guarantee validity of conditions but do not throw any error message if these conditions are not met. We recommend documenting errors as best as possible in order to improve their handling and the user interface.

1 issue of this type has been found in the smart contract.

Recommendation

We recommend adding an error message to the following require statement :

```
//Edited code containing missing error message
//l1016
require(IERC20(busdAddress).transfer(marketingAddress,mar
ketingFunds), "Transfer to marketing failed");
```

MSG-3 | Too long error messages

Description

The smart contract has some error messages that are too long. The industry standards specify error messages must have a maximal length of 32 bytes. We recommend having the shortest possible error messages to optimize gas costs (see github.com/ethereum/solidity/issues/4588) and improve error handling. 15 issues of this type have been found in the smart contract.

Recommendation

We recommend shortening these error messages :

```
//l 106
require(success, "Unable to send value");
//l 123
require(address(this).balance >= value, "Insuffcient
balance for call");
//l 175
require(newOwner != address(0), "Owner cannot be 0
address");
//l 196
require(_previousOwner == msg.sender, "Unauthorized to
unlock");
//l 557
_approve(sender, _msgSender(),
_allowances[sender][_msgSender()].sub(amount, "ERC20:
transfer over allowance"));
```

```
//l 567
_approve(_msgSender(), spender,
_allowances[_msgSender()][spender].sub(subtractedValue,
"ERC20: decreased allowance < 0"));
//l 605
require(rAmount <= _rTotal, "Amount over total
reflections");
//l 634
require(owner != address(0), "ERC20: approve from 0
address");
//l 635
require(spender != address(0), "ERC20: approve to 0
address");
//l 646
require(from != address(0), "ERC20: transfer from 0
address");
//l 647
require(to != address(0), "ERC20: transfer to 0
address");
//l 648
(amount > 0, "Transfer amount must be > 0");
//l 653
require(amount <= _bMaxTxAmount, "Transfer over max buy
amount");
//l 657
require(amount <= _sMaxTxAmount, "Transfer over max sell
amount");
//l 992
require(_token != address(this), "You cannot take all
native token");
```

UINT-1 | Wrong uint type

Description

Some variables in the contract are of type uint, but not of the right size. In order to optimize gas costs when deploying and using the contract, we invite you to assign the right size uint to each variable.

16 errors of this type have been found in the smart contract.

Recommendation

We recommend changing these uint sizes :

```
//Edited code containing resized uint :
//A fee cannot be that high that it needs being uint256.
We advise changing them into uint8 (and change the way
you handle them, by multiplying them by 100 if needed) or
changing them into uint 16.
//l 434-451
uint8_taxFee = 0;
uint8 private _vaultFee = 0;
uint8 private _tempTaxFee = 0;
uint8 private _tempVaultFee = 0;
uint8 private _tempLiquidityFee = 0;
uint8 private _buyTaxFee = 0;
uint8 private _buyVaultFee = 0;
uint8 private _sellTaxFee = 0;
uint8 private _sellVaultFee = 0;
uint8 public _buyRewardFee = 4;
uint8 public _buyMarketingFee = 4;
uint8 public _buyLiquidityFee = 9;
```

```
uint8 private _liquidityFee = 9;
uint8 public _sellRewardFee = 5;
uint8 public _sellMarketingFee = 5;
uint8 public _sellLiquidityFee = 12;
```

COMP-1 | Unfixed version of compiler

Description

Grizzly Rocket's contract does not have locked compiler versions, meaning a range of compiler versions can be used. This can lead to differing bytecodes being produced depending on the compiler version, which can create confusion when debugging as bugs may be specific to a specific compiler version(s).

To rectify this, we recommend setting the compiler to a single version, the lowest version tested to be compatible with the code, an example of this change can be seen below.

1 error of this type has been found in the smart contract.

Recommendation

We recommend fixing the compiler version to the most recent one :

```
//Edited code containing fixed compiler version
//l6
pragma solidity 0.8.13;
```

BLOC-1 | Using block.timestamp

Description

The use of block.timestamp can be problematic. The timestamp can be partially manipulated by the miner (see https://cryptomarketpool.com/block-timestamp-manipulation-attack/). In this smart contract this is not critical as in the worst case an attacker could force the automatic liquify to run faster.

Recommendation

We fully understand the smart contract's logic of the Grizzly Rocket token. The use of block.timestamp is required to power the swapping and locking mechanism and we cannot replace it. Nevertheless, it is still useful to point out this kind of potential security problem.

THRE-1a | Missing threshold for max buy amount

Description

The maximum transaction change function for buying does not have a safety threshold. Even though this function is protected by the onlyOwner modifier, it is important to add a threshold to prevent an attacker from stopping trading as easily.

1 error of this type has been found in the smart contract.

Recommendation

We recommend adding these thresholds to concerned functions :

```
//Edited code containing threshold. We leave it to you to decide which threshold best suits the logic of the project
//l 925
function setBuyMaxTxAmount(uint256 bMaxTxAmount) external onlyOwner {
    require(bMaxTxAmount > threshold, "Cannot set max tx over XXX"); //where XXX is the threshold you set
        _bMaxTxAmount = bMaxTxAmount;
    }
```

THRE-1b | Missing threshold for max sell amount

Description

The maximum transaction change function for selling does not have a safety threshold. Even though this function is protected by the onlyOwner modifier, it is important to add a threshold to prevent an attacker from stopping trading as easily.

1 error of this type has been found in the smart contract.

Recommendation

We recommend adding these thresholds to concerned functions :

```
//Edited code containing threshold. We leave it to you to
decide which threshold best suits the logic of the
project
//l 929
function setSellMaxTxAmount(uint256 sMaxTxAmount)
external onlyOwner {
    require(sM   axTxAmount > threhsold, "Cannot set max
tx over XXX"); //where XXX is the threshold you set
       _sMaxTxAmount = sMaxTxAmount;
    }
```

THRE-1c | Missing threshold for fees setting

Description

The fees updating function doesn't include a security threshold. Even though this function is protected by the onlyOwner modifier, it is important to add a threshold to prevent an attacker from setting high taxes as easily.

1 error of this type has been found in the smart contract.

Recommendation

We recommend adding these thresholds to concerned functions :

```
//Edited code containing threshold. We leave it to you to
decide which threshold best suits the logic of the
project
//l 957
 function setReawardAndMarketingFee(uint256
_sellRewardsPercent,uint256 _sellMarketingPercent,
uint256 _sellLiquidityPercent, uint256
_buyRewardsPercent,uint256 _buyMarketingPercent, uint256
_buyLiquidityPercent ) external onlyOwner {
    _sellRewardFee = _sellRewardsPercent;
    _sellMarketingFee = _sellMarketingPercent;
    _buyRewardFee = _buyRewardsPercent;
    _buyMarketingFee = _buyMarketingPercent;
    _buyLiquidityFee
=_buyLiquidityPercent+_buyRewardsPercent+_buyMarketingPer
cent;
    _sellLiquidityFee = _sellLiquidityPercent+
```

```
_sellMarketingPercent+_sellRewardsPercent;
    _liquidityFee =
_buyLiquidityPercent+_buyRewardsPercent+_buyMarketingPerc
ent;
    require(_sellRewardFee < rewardfeethreshold, "sell
reward fee cannot be over XXX");
    require(_sellMarketingFee < rewardfeethreshold, "sell
marketing fee cannot be over XXX");
    require(_buyRewardFee < buyrewardfeethreshold, "buy
reward fee cannot be over XXX");
    require(_buyLiquidityFee < _buyLiquidityFeethreshold,
"buy liquidity fee cannot be over XXX");
    require(_sellLiquidityFee <
sellliquidityfeethreshold, "sell liquidity fee cannot be
over XXX");
    require(_liquidityFee < liquidityFeethreshokd,
"Liquidity fee cannot be over XXX");
}
```

CENT-1 | Centralization of major privileges

Description

The onlyOwn modifier of the smart contract gives major privileges over it (in particular, it can change the addresses that collect the fees and change the fees)*. This can be a problem, in the case of a hack, an attacker who has taken possession of these privileged accounts could damage the project and the investors.

*This list is not exhaustive but presents the most sensitive points

Recommendation

We recommend at least establishing a community governance protocol to avoid such centralization. For more information, see https://solidity-by-example.org /app/multi-sig-wallet/

EXT-1 | Dependence to an external protocol

Description

The contract is serving as the underlying entity to interact with third party Pancakeswap protocols. The scope of the audit would treat this third party entity as black box and assume it is fully functional. However in the real world, third parties may be compromised and may have led to assets lost or stolen.

Recommendation

We encourage the team to constantly monitor the security level of the entire Pancakeswap project, as the security of the token is highly dependent on the security of the decentralized exchange platform.

Global security warnings

These are safety issues for the whole project. They are not necessarily critical problems but they are inherent in the structure of the project itself. Potential attack vectors for these security problems should be monitored.

CENT-1 | Global SPOF (Single Point Of Failure)

The project's smart contracts often have a problem of centralized privileges. The owner system in particular can be subject to attack. To prevent this, we advise establishing secure project administration protocols and strengthening the security of project administrators.

Compliance with industry standards

The way the contract is developed and its compliance with industry standards are part of the project. In order to increase the optimization of the latter, we recommend refining the code to best fit industry best practices, in particular the use of error messages and uint types.

DISCLAIMER

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement.

This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement.

This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without StaySAFU's prior written consent.This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts StaySAFU to perform a security assessment.

This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance. This report should not be used in any way

to make decisions around investment or involvement with any particular project.

This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk.

StaySAFU's position is that each company and individual are responsible for their own due diligence and continuous security. StaySAFU's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or fun.