SAFETIN AUDIT

GEMPAD

April 17th, 2022

TABLE OF CONTENTS

AUDIT SUMMARY

This report was written for Gempad (GEMS) in order to find flaws and vulnerabilities in the Gempad 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 Gempad 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	Gempad
Description	Gempad is a frontline protocol for users and project-owners designed to help to launch their projects and tokens in the easiest way possible. The Gempad token is the ERC20 which powers the project.
Platform	BNB Chain
Language	Solidity
Codebase	https://pastebin.com/42KXmjXN

FINDINGS SUMMARY

Vulnerability	Total	Resolved
Critical	0	0
Major	0	0
Medium	2	1
Minor	8	5
Informational	2	2

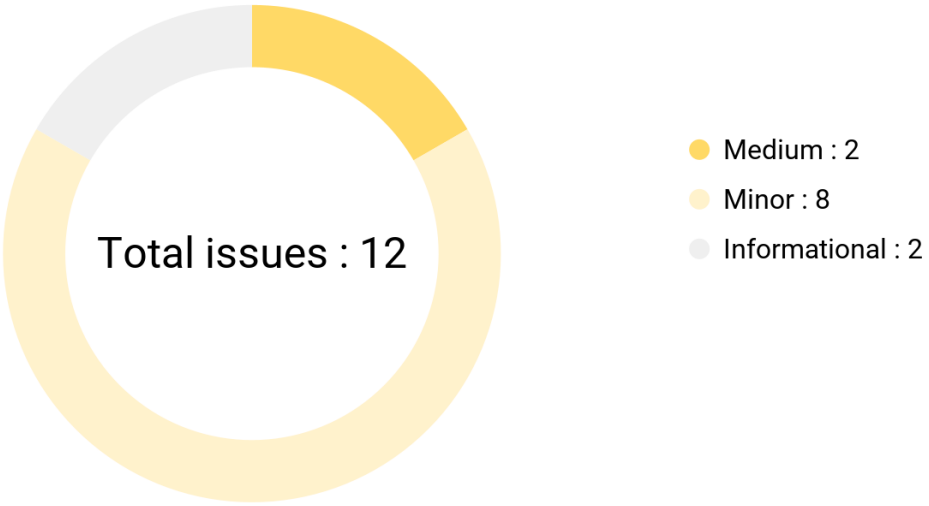
EXECUTIVE SUMMARY

Gempad is the first launchpad to offer raising funds in any crypto token. The project is powered by an ERC20 token, the Gempad (GEMS). Gempad holders have access to events like seed rounds, private sales, partial sales and more. Transactions are taxed as follows (updatable taxes below a certain threshold):

- Sales are taxed at less than 30%, distributed between reward fee, liquidity fee and marketing fee.
- Purchases are taxed at less than 30%, distributed between reward fee, liquidity fee and marketing fee.
- Transfers are taxed at less than 30%, distributed between reward fee, liquidity fee and marketing fee.

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

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
BLOC-2	Usage of block.number	Minor
COMP-1	Unfixed version of compiler	Minor
TX-1	Use of tx.origin	Minor
THRE-1a	Missing threshold for minor func	Minor

THRE-1b	Missing threshold for minor func	Minor
THRE-1c	Missing threshold for minor func	Minor
THRE-3	Insufficient threshold	Minor
UINT-1	Unoptimized uint size	Informational
MSG-3	Too long error message	Informational

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. 4 issues of this type have been found in the smart contract.

Recommendation

We recommend shortening these error messages :

```
//Line references with edited error messages
//line 570 :
"The pair cannot be removed from
automatedMarketMakerPairs" -> "The pair cannot be
removed"
//line 731 :
"_transfer:: Transfer Delay enabled.  Only one purchase
per block allowed." -> "Only one purchase per block"
//line 744 :
"Buy transfer amount exceeds the maxTransactionAmount."
-> "buy transfer over max amount"
//line 758 :
"Sell transfer amount exceeds the maxTransactionAmount."
-> "Sell transfer over max amount"
```

UINT-1 | Unoptimized uint size

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 recommend to assign the right size uint to each variable.
44 errors of this type have been found in the smart contract.

Recommendation

We recommend changing these uint sizes. We listed all the changes needed on this gist.

COMP-1 | Unfixed version of compiler

Description

Gempad token'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.

Recommendation

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

```
//Edited code containing fixed compiler version
//l2
pragma solidity 0.8.13;
```

BLOC-1 | Use of 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/).

Recommendation

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

BLOC-2 | Use of block.number

Description

The use of block.number can be problematic. The timestamp can be partially manipulated by the miner (see https://cryptomarketpool.com/block-timestamp-manipulation-attack/). Since the timestamp of a block cannot be fully trusted, the exact block counting at an exact timestamp cannot be fully trusted.

Recommendation

We fully understand the smart contract's logic of the Gempad token. The use of block.number is required to power transfer and trading mechanisms and we cannot replace it. Nevertheless, it is still useful to point out this kind of potential security problem.

TX-1 | Use of tx.origin

Description

The use of tx.origin is strongly deprecated by the industry. It can lead to phishing attacks by falsifying the identity of the original caller of the function. Read more about it here.

Recommendation

In the case of this smart contract, we recommend using msg.sender instead of tx.origin.

THRE-1a | Missing threshold for max sell amount

Description

The maximum transaction amount change function 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 setting max transaction amount as 0 as easily.

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

Recommendation

We recommend adding a threshold to the concerned function. We leave it to you to decide which threshold best fits the logic of the project :

```
//Edited code containing threshold
//l 535
function updateMaxTransactionAmount(uint256
_maxTransactionAmount)
    external
    onlyOwner
    {
    require(_maxTransactionAmount*(10**decimals()) > XXX,
“Cannot set max amount under XXX”);
    maxTransactionAmount =
_maxTransactionAmount*(10**decimals());
    emit
UpdateMaxTransactionAmount(_maxTransactionAmount);
    }
```

THRE-1b | Missing threshold for max wallet size

Description

The maximum wallet size change function 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 setting max wallet size as 0 as easily.

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

Recommendation

We recommend adding a threshold to the concerned function. We leave it to you to decide which threshold best fits the logic of the project :

```
//Edited code containing a threshold
//l 543
function updateMaxWallet(uint256 _maxWallet) external
onlyOwner {
    require(_maxWallet*(10**decimals()) > XXX, "Cannot
set max wallet under XXX" );
    maxWallet = _maxWallet*(10**decimals());
    emit UpdateMaxWallet(_maxWallet);
}
```

THRE-1c | Missing threshold for gas price limit

Description

The gas price limit setting function 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 setting max transaction price as 0 as easily.

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

Recommendation

We recommend adding a threshold to the concerned function. We leave it to you to decide which threshold best fits the logic of the project :

```
//Edited code containing threshold
//l 585
function updateGasPriceLimit(uint256 gas) external
onlyOwner {
    require(gas > XXX, "Cannot set gas price limit <
XXX"); //here the threshold could be 2*the average
transaction gas cost
        _gasPriceLimit = gas * 1 gwei;
}
```

THRE-3 | Insufficient threshold

Description

The protect-block fee setting function does have a safety threshold, but it is set to 100%. Of course, we don't want the fees to be set above 100%. However, it is problematic that they can be set to a value close to that (like 99.99% for example). Even though this function is protected by the onlyOwner modifier, it is important to add a threshold to prevent an attacker from setting the protect-block fee as a very high value as easily.
1 error of this type has been found in the smart contract.

Recommendation

We recommend changing this threshold. As an example, a 50% threshold could be a flexible and safe threshold. However, we leave it to you to decide which threshold best fits the logic of the project :

```
//Edited code containing 50% threshold
//l 658
function setProtectBlockFee(
    uint256 protectBlockRewardFee,
    uint256 protectBlockLiquidityFee,
    uint256 protectBlockMarketingFee
) external onlyOwner {
    _protectBlockRewardFee = protectBlockRewardFee;
    _protectBlockLiquidityFee = protectBlockLiquidityFee;
    _protectBlockMarketingFee = protectBlockMarketingFee;
    require(
```

```
        _protectBlockRewardFee +
             _protectBlockLiquidityFee +
             _protectBlockMarketingFee <
             500,
    "Must keep fees below 50%"
     );
}
```

CENT-1 | Centralization of major privileges

Description

The onlyOwner modifier of the smart contract gives major privileges over it (changing the staking address, lock the trade)*. This can be a problem, in the case of a hack, an attacker who has taken possession of this privileged account could damage the project and the investors.

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

Recommendation

We recommend at least to use a multi-sig wallet as the owner address, and at best to establish 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 interacts with 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. We fully understand that the business logic of the Gempad token is designed to work with PancakeSwap.

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 and authorization system in particular can be subject to attack. To address this security issue we recommend using a multi-sig wallet, 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 Safetin'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 Safetin 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

Safetin security assessment 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. Safetin's position is that each company and individual are responsible for their own due diligence and continuous security. Safetin'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.