



CFG NINJA AUDITS

Security Assessment

Pumpy Penguin Token

November 29, 2023

Audit Status: Pass

Audit Edition: Advance














POWERED BY
BLADE POOL

Risk Analysis

















Classifications of Manual Risk Results

Classification	Description
 Critical	Danger or Potential Problems.
 Major	Be Careful or Fail test.
 Minor	Pass, Not-Detected or Safe Item.
 Informational	Function Detected

Manual Code Review Risk Results

Contract Priviledge	Description
 Buy Tax	10
 Sale Tax	10
 Cannot Buy	Pass
 Cannot Sale	Pass
 Max Tax	10
 Modify Tax	Not-Detected
 Fee Check	Pass
 Is Honeypot?	Not detected
 Trading Cooldown	Not Detected
 Can Pause Trade?	Pass
 Pause Transfer?	Not-Detected



Contract Priviledge	Description
 Max Tx?	Pass
 Is Anti Whale?	Not Detected
 Is Anti Bot?	Not Detected
 Is Blacklist?	Not Detected
 Blacklist Check	Pass
 is Whitelist?	Detected
 Can Mint?	Pass
 Is Proxy?	Not Detected
 Can Take Ownership?	Not detected
 Hidden Owner?	Not detected
 Owner	0x0559c6f1a72357A8c2546659d8Edd40aefAAAB1a
 Self Destruct?	Not Detected
 Other?	Not detected
 Other?	Not detected
 Holders	3
 Auditor Confidence	High

The following quick summary it's added to the project overview; however, there are more details about the audit and its results. Please read every detail.



Project Overview

Token Summary

Parameter	Result
Address	0xEF91428F4C578e149A5572B25917f7472af6bFaB
Name	Pumpy Penguin
Token Tracker	Pumpy Penguin (PP)
Decimals	18
Supply	420,000,000,000,000,000
Platform	Binance Smart Chain
compiler	v0.8.4+commit.c7e474f2
Contract Name	PumpyPenguin
Optimization	Yes with 200 runs
LicenseType	MIT
Language	Solidity
Codebase	https://bscscan.com/address/0xEF91428F4C578e149A5572B25917f7472af6bFaB#code
Payment Tx	0x5727c8eff4ae5ce4b88f8c89d7a09d190e77fd5420acc97be19a45eae02e9290



Main Contract Assessed Contract Name

Name	Contract	Live
Pumpy Penguin	0xEF91428F4C578e149A5572B25917f7472af6bFaB	Yes

TestNet Contract Assessed Contract Name

Name	Contract	Live
Pumpy Penguin	0x83F47276c3aE0b2f52fefb27E68B9Db04cf4a8D7	Yes

Solidity Code Provided

SolID	File Sha-1	FileName
PumpyPenguin	2a165091d86027bbfff565ad385c8ccdbcc27159	PumpyPenguin.sol
PumpyPenguin		
PumpyPenguin		
PumpyPenguin		



Call Graph

The contract for Pumpy Penguin has the following call graph structure.



Smart Contract Vulnerability Checks

The Smart Contract Weakness Classification Registry (SWC Registry) is an implementation of the weakness classification scheme proposed in EIP-1470. It is loosely aligned to the terminologies and structure used in the Common Weakness Enumeration (CWE) while overlaying a wide range of weakness variants that are specific to smart contracts.

ID	Severity	Name	File	location
SWC-100	Pass	Function Default Visibility	PumpyPenguin.sol	L: 0 C: 0
SWC-101	Pass	Integer Overflow and Underflow.	PumpyPenguin.sol	L: 0 C: 0
SWC-102	Pass	Outdated Compiler Version file.	PumpyPenguin.sol	L: 0 C: 0
SWC-103	Low	A floating pragma is set.	PumpyPenguin.sol	L: 7 C: 0
SWC-104	Pass	Unchecked Call Return Value.	PumpyPenguin.sol	L: 0 C: 0
SWC-105	Pass	Unprotected Ether Withdrawal.	PumpyPenguin.sol	L: 0 C: 0
SWC-106	Pass	Unprotected SELFDESTRUCT Instruction	PumpyPenguin.sol	L: 0 C: 0
SWC-107	Pass	Read of persistent state following external call.	PumpyPenguin.sol	L: 0 C: 0
SWC-108	Low	State variable visibility is not set..	PumpyPenguin.sol	L: 20 C: 12
SWC-109	Pass	Uninitialized Storage Pointer.	PumpyPenguin.sol	L: 0 C: 0
SWC-110	Pass	Assert Violation.	PumpyPenguin.sol	L: 0 C: 0



ID	Severity	Name	File	location
SWC-111	Pass	Use of Deprecated Solidity Functions.	PumpyPenguin.sol	L: 0 C: 0
SWC-112	Pass	Delegate Call to Untrusted Callee.	PumpyPenguin.sol	L: 0 C: 0
SWC-113	Pass	Multiple calls are executed in the same transaction.	PumpyPenguin.sol	L: 0 C: 0
SWC-114	Pass	Transaction Order Dependence.	PumpyPenguin.sol	L: 0 C: 0
SWC-115	Low	Authorization through tx.origin.	PumpyPenguin.sol	L: 1130 C: 97, L: 1130 C: 97
SWC-116	Pass	A control flow decision is made based on The block.timestamp environment variable.	PumpyPenguin.sol	L: 0 C: 0
SWC-117	Pass	Signature Malleability.	PumpyPenguin.sol	L: 0 C: 0
SWC-118	Pass	Incorrect Constructor Name.	PumpyPenguin.sol	L: 0 C: 0
SWC-119	Pass	Shadowing State Variables.	PumpyPenguin.sol	L: 0 C: 0
SWC-120	Pass	Potential use of block.number as source of randomness.	PumpyPenguin.sol	L: 0 C: 0
SWC-121	Pass	Missing Protection against Signature Replay Attacks.	PumpyPenguin.sol	L: 0 C: 0
SWC-122	Pass	Lack of Proper Signature Verification.	PumpyPenguin.sol	L: 0 C: 0
SWC-123	Pass	Requirement Violation.	PumpyPenguin.sol	L: 0 C: 0
SWC-124	Pass	Write to Arbitrary Storage Location.	PumpyPenguin.sol	L: 0 C: 0



ID	Severity	Name	File	location
SWC-125	Pass	Incorrect Inheritance Order.	PumpyPenguin.sol	L: 0 C: 0
SWC-126	Pass	Insufficient Gas Griefing.	PumpyPenguin.sol	L: 0 C: 0
SWC-127	Pass	Arbitrary Jump with Function Type Variable.	PumpyPenguin.sol	L: 0 C: 0
SWC-128	Pass	DoS With Block Gas Limit.	PumpyPenguin.sol	L: 0 C: 0
SWC-129	Pass	Typographical Error.	PumpyPenguin.sol	L: 0 C: 0
SWC-130	Pass	Right-To-Left-Override control character (U+202E).	PumpyPenguin.sol	L: 0 C: 0
SWC-131	Pass	Presence of unused variables.	PumpyPenguin.sol	L: 0 C: 0
SWC-132	Pass	Unexpected Ether balance.	PumpyPenguin.sol	L: 0 C: 0
SWC-133	Pass	Hash Collisions with Multiple Variable Length Arguments.	PumpyPenguin.sol	L: 0 C: 0
SWC-134	Pass	Message call with hardcoded gas amount.	PumpyPenguin.sol	L: 0 C: 0
SWC-135	Pass	Code With No Effects (Irrelevant/Dead Code).	PumpyPenguin.sol	L: 0 C: 0
SWC-136	Pass	Unencrypted Private Data On-Chain.	PumpyPenguin.sol	L: 0 C: 0

We scan the contract for additional security issues using MYTHX and industry-standard security scanning tools.



Smart Contract Vulnerability Details

SWC-103 - Floating Pragma.

CWE-664: Improper Control of a Resource Through its Lifetime.

References:

Description:

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Remediation:

Lock the pragma version and also consider known bugs (<https://github.com/ethereum/solidity/releases>) for the compiler version that is chosen.

Pragma statements can be allowed to float when a contract is intended for consumption by other developers, as in the case with contracts in a library or EthPM package. Otherwise, the developer would need to manually update the pragma in order to compile locally.

References:

Ethereum Smart Contract Best Practices - Lock pragmas to specific compiler version.



Smart Contract Vulnerability Details

SWC-108 - State Variable Default Visibility

CWE-710: Improper Adherence to Coding Standards

Description:

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

Remediation:

Variables can be specified as being public, internal or private. Explicitly define visibility for all state variables.

References:

Ethereum Smart Contract Best Practices - Explicitly mark visibility in functions and state variables



Smart Contract Vulnerability Details

SWC-115 - Authorization through tx.origin

CWE-477: Use of Obsolete Function

Description:

tx.origin is a global variable in Solidity which returns the address of the account that sent the transaction. Using the variable for authorization could make a contract vulnerable if an authorized account calls into a malicious contract. A call could be made to the vulnerable contract that passes the authorization check since tx.origin returns the original sender of the transaction which in this case is the authorized account.

Remediation:

tx.origin should not be used for authorization. Use msg.sender instead.

References:

Solidity Documentation - tx.origin

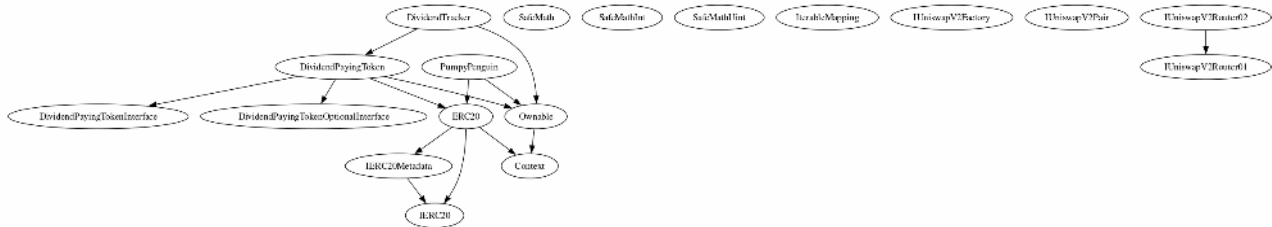
Ethereum Smart Contract Best Practices - Avoid using tx.origin

SigmaPrime - Visibility.



Inheritance

The contract for Pumpy Penguin has the following inheritance structure.



Smart Contract Advance Checks



ID	Severity	Name	Result	Status
PP-01	Minor	Potential Sandwich Attacks.	Pass	Not-Found
PP-02	Minor	Function Visibility Optimization	Pass	Not-Detected
PP-03	Major	Lack of Input Validation.	Pass	Not-Detected
PP-04	Major	Centralized Risk In addLiquidity.	Pass	Not-Detected
PP-05	Major	Missing Event Emission.	Fail	Detected
PP-06	Minor	Conformance with Solidity Naming Conventions.	Pass	Not-Detected
PP-07	Minor	State Variables could be Declared Constant.	Pass	Not-Found
PP-08	Minor	Dead Code Elimination.	Pass	Not-Found
PP-09	Major	Third Party Dependencies.	Pass	Not-Found
PP-10	Major	Initial Token Distribution.	Pass	Not-Found
PP-11	Minor	Airdrop is present in code.	Pass	Not-Detected
PP-12	Major	Centralization Risks In The X Role	Pass	Not-Found
PP-13	Informational	Extra Gas Cost For User..	Pass	Not-Found
PP-6	Critical	Unnecessary Use Of SafeMath	Fail	Detected
PP-15	Medium	Symbol Length Limitation due to Solidity Naming Standards.	Pass	Not-Detected



ID	Severity	Name	Result	Status
PP-16	Meduium	Invalid collection of Taxes during Transfer.	Pass	Not-Detected
PP-17	Informational	Conformance to numeric notation best practice.	Pass	Not-Found
PP-18	Minor	Stop Transactions by using Enable Trade.	Pass	Not-Detected



PP-05 | Missing Event Emission.

Category	Severity	Location	Status
Volatile Code	 Major	PumpyPenguin.sol: 125, 14	 Detected

Description



Detected missing events for critical arithmetic parameters. There are functions that have no event emitted, so it is difficult to track off-chain changes. The linked code does not create an event for the transfer.

Remediation

Emit an event for critical parameter changes. It is recommended emitting events for the sensitive functions that are controlled by centralization roles.



PP-14 | Unnecessary Use Of SafeMath

Category	Severity	Location	Status
Logical Issue	 Critical	PumpyPenguin.sol: 7,9	 Detected

Description

The SafeMath library is used unnecessarily. With Solidity compiler versions 0.8.0 or newer, arithmetic operations

will automatically revert in case of integer overflow or underflow.

library SafeMath {

An implementation of SafeMath library is found.

using SafeMath for uint256;

SafeMath library is used for uint256 type in contract.

Remediation

We advise removing the usage of SafeMath library and using the built-in arithmetic operations provided by the






Solidity programming language

Project Action








Technical Findings Summary

Classification of Risk

Severity	Description
 Critical	Risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
 Major	Risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
 Medium	Risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform
 Minor	Risks can be any of the above but on a smaller scale. They generally do not compromise the overall integrity of the Project, but they may be less efficient than other solutions.
 Informational	Errors are often recommended to improve the code's style or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

Findings

Severity	Found	Pending	Resolved
 Critical	0	0	0
 Major	1	0	0
 Medium	0	0	0
 Minor	1	0	0
 Informational	0	0	0
Total	2	0	0



Social Media Checks

Social Media	URL	Result
Twitter	https://twitter.com/pumpypenguin	Pass
Other		Fail
Website	https://pumpypenguin.com	Pass
Telegram	https://t.me/pumpypenguin	Pass

We recommend to have 3 or more social media sources including a completed working websites.

Social Media Information Notes:

Auditor Notes: undefined

Project Owner Notes:



Assessment Results

Score Results

Review	Score
Overall Score	83/100
Auditor Score	83/100
Review by Section	Score
Manual Scan Score	36/53
SWC Scan Score	34 /37
Advance Check Score	13 /19

The Following Score System Has been Added to this page to help understand the value of the audit, the maximum score is 100, however to attain that value the project must pass and provide all the data needed for the assessment. Our Passing Score has been changed to 80 Points, if a project does not attain 80% is an automatic failure. Read our notes and final assessment below.

Audit Passed



Assessment Results

Important Notes:

- Owner can't set max tx amount.■
- No high-risk Exploits/Vulnerabilities Were Found in the Source Code.

Auditor Score =83
Audit Passed



Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different requirements on the input variables than a setter function.

Coding Best Practices

ERC 20 Coding Standards are a set of rules that each developer should follow to ensure the code meets a set of criteria and is readable by all the developers.



Disclaimer

CFGNINJA has conducted an independent security assessment to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the reviewed code for the scope of this assessment. This report does not constitute agreement, acceptance, or advocacy for the Project, and users relying on this report should not consider this as having any merit for financial advice in any shape, form, or nature. The contracts audited do not account for any economic developments that the Project in question may pursue, and the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude, and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are entirely free of exploits, bugs, vulnerabilities or deprecation of technologies.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence, regardless of the findings presented. Information is provided 'as is, and CFGNINJA is under no covenant to audited completeness, accuracy, or solidity of the contracts. In no event will CFGNINJA or its partners, employees, agents, or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions or actions with regards to the information provided in this audit report.

The assessment services provided by CFGNINJA are subject to dependencies and are under continuing development. You agree that your access or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies with high levels of technical risk and uncertainty. The assessment reports could include false positives, negatives, and unpredictable results. The services may access, and depend upon, multiple layers of third parties.

