SAULIDITY

BLOCKCHAIN SECURITY

2023

SMART CONTRACT SECURITY ANALYSIS

PREPARED BY

SAULIDITY

PRESENTED TO

Star System Labs

MADE IN CANADA

1055 Rue Lucien-L'Allier
Montreal, QC H3G 3C4

audit@saulidity.com
www.saulidity.com

2023

SAULIDITY

SECURITY ASSESSMENT

Smart Contract Audit

saulidity.com
Saulidity
@Saulidity

DISCLAIMER

This report does not constitute financial advice, and Saulidity is not accountable or liable for any negative consequences resulting from this report, nor may Saulidity be held liable in any way. You agree to the terms of this disclaimer by reading any part of the report. If you do not agree to the terms, please stop reading this report immediately and delete and destroy any and all copies of this report that you have downloaded and/or printed. This report was entirely based on information given by the audited party and facts that existed prior to the audit. Saulidity and/or its auditors cannot be held liable for any outcome, including modifications (if any) made to the contract(s) for the audit that was completed. No modifications have been made to the contract(s) by the Saulidity team unless it is indicated explicitly. The audit does not include the project team, website, logic, or tokenomics, but if it does, it will be indicated explicitly. The security is evaluated only on the basis of smart contracts only. There were no security checks performed on any apps or activities. There has not been a review of any product codes. It is assumed by Saulidity that the information and materials given were not tampered with, censored, or misrepresented. Even if this report exists and Saulidity makes every effort to uncover any security flaws, you should not rely completely on it and should conduct your own independent research. Saulidity hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Saulidity, for any amount or kind of loss or damage that may result to you or any other person or any kind of company, community, association and institution. Saulidity is the exclusive owner of this report, and it is published by Saulidity. Without Saulidity's express written authorization, use of this report for any reason other than a security interest in the individual contacts, or use of sections of this report, is forbidden.

TABLE OF CONTENTS

INTRODUCTION

Saulidity is a renowned blockchain security firm based in Montreal QC that provides a suite of vital services, including smart contract audits, penetration testing, node audits, and blockchain project development.

In a market where confidence and trust are key, a genuine project may simply increase its user base enormously with an official audit performed by Saulidity. The security of blockchain projects has never been more crucial than it is in today's rapidly expanding digital landscape. In the face of burgeoning technology, the integrity and security of blockchain networks is paramount. The decentralized nature of these networks, while presenting unparalleled opportunities for transparency and disintermediation, also exposes them to unique security threats.

Potential vulnerabilities in smart contracts, nodes, or overall network design could be exploited by malicious actors, leading to significant financial loss, data breaches, and damage to reputation. As such, **comprehensive security audits and assessments are not just beneficial, but essential in preventing such instances, ensuring the long-term success of blockchain projects.**

Saulidity applies extensive expertise and profound understanding of blockchain technology to safeguard your digital assets and maintain the robustness of your blockchain projects to fortify your projects, secure your investments, and empower you with the confidence that your blockchain initiatives are secure and reliable.

The information in this report should be used to understand the smart contract's risk exposure and as a guide to improving the code by addressing the concerns that were discovered. For a thorough understanding of the analysis, please read the entire document.

For a thorough understanding of the audit, please read the entire document.

The information in this report should be used to understand the smart contract's risk exposure and as a guide to improving the smart contract's security posture by addressing the concerns that were discovered.

The security specialists do complete studies independently of one another in order to uncover any security issues in the contracts as comprehensively as feasible. For optimum security and professionalism, all of our audits are undertaken by at least two independent auditors.

SCOPE & INFO

Available Saulidity audit packages:

- **Essential Audit**
- **Standard Audit**
- **Premium Audit**

- **Platform Pentest**
- **Custom Audit**

We conducted a review on the following smart contract(s):

- PrimordialPePe.sol
- MiningRig.sol

Star System Labs engaged Saulidity to conduct an **Essential Audit** of their smart contracts. This foundational review can be followed by a more in-depth audit package should the client determine it necessary based on our initial report.

The project's website, logic, or tokenomics have not been vetted by Saulidity.

The security specialists did a complete study independently of one another in order to uncover any security issues in the contracts as comprehensively as feasible within the scope chosen by the client.

During our audit, we conducted an inquiry using automated analysis and manual review approaches. The purpose of this audit is to:

- Identify potential security issues with the smart contracts

Project Name	Star System Labs
Commit ID	0b1fd5566021ae474b50678c3bfe610a6bc4c362
Updated Commit ID	N/A
Contract Address	N/A
Report ID	mkSAUL001 V2.0
Website	https://www.starsystemlabs.com
Code language	Solidity

We analyze smart contracts for both well-known and more specific vulnerabilities.

Here are some of the most well-known vulnerabilities:

ITEM	DESCRIPTION
Default Visibility	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.
Integer Overflow and Underflow	If unchecked math is used, all math operations should be safe from overflows and underflows.
Outdated Compiler Version	It is recommended to use a recent version of the Solidity compiler.
Floating Pragma	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.
Unchecked Call Return Value	The return value of a message call should be checked.
Access Control & Authorization	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.
Selfdestruct	The contract should not be destroyed until it has funds belonging to users.
Check-Effect-Interaction	CEI pattern should be followed if the code performs any external call.

ITEM	DESCRIPTION
Default Visibility	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.
Integer Overflow and Underflow	If unchecked math is used, all math operations should be safe from overflows and underflows.
Outdated Compiler Version	It is recommended to use a recent version of the Solidity compiler.
Floating Pragma	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.
Unchecked Call Return Value	The return value of a message call should be checked.
Access Control & Authorization	Ownership takeover should not bepossible. All crucial functions should be protected. Users could not affect data that belongs to other users.
Selfdestruct	The contract should not be destroyed until it has funds belonging to users.
Check-Effect-Interaction	CEI pattern should be followed if the code performs any external call.

ITEM	DESCRIPTION
Signature Unique Id	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.
Shadowing State Variable	State variables should not be shadowed.
Weak Sources of Randomness	Random values should never be generated from Chain Attributes.
Incorrect Inheritance Order	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.
Calls Only to Trusted Addresses	All external calls should be performed only to trusted addresses.
Presence of unused variables	The code should not contain unused variables if this is not justified by design.

METHODOLOGY

Saulidity conducted a mixture of manual and automated security evaluations. An **Essential Audit** package is carried out using the following steps:

- Smart contract walkthrough
- Graphing out functionality and contract logic/connectivity/functions
- Scanning of contracts for vulnerabilities
- Static Analysis

APPENDIX

Vulnerabilities can be divided into four threat levels: Critical, High, Medium and Low. The classification is mainly based on the impact, likelihood of utilization and other factors.

Critical flaws can result in the loss of assets or the alteration of data and are often simple to exploit.

High-level vulnerabilities are challenging to exploit, but they can have a big influence on how smart contracts are executed, such as giving the public access to key features.

Although **medium-level** vulnerabilities should be fixed, they generally cannot result in the loss of assets or the manipulation of data.

Low-level and Lowest/Code Style/Optimization flaws are typically caused by code fragments that are out-of-date, useless, etc. and cannot significantly affect execution.

EXECUTIVE SUMMARY

0

CRITICAL SEVERITY

-

0

HIGH SEVERITY

-

1

MEDIUM

- Locked Ether

3

LOW

- Missing Zero Address Validation x 2
- Missing Event for Access Control

0

LOWEST/ CODE STYLE/ OPTIMIZED PRACTICE

-

SEVERITY	FOUND
Critical	0
High	0
Medium	1
Low	3
Lowest / Code Style / Optimized Practice	0

ACCORDING TO THE ANALYSIS, THERE MAY BE A HIGH SEVERITY VULNERABILITY IN ONE OF THE PROVIDED CONTRACTS. THE FINDINGS ARE PRESENTED IN THE ANALYSIS SECTION OF THE REPORT.

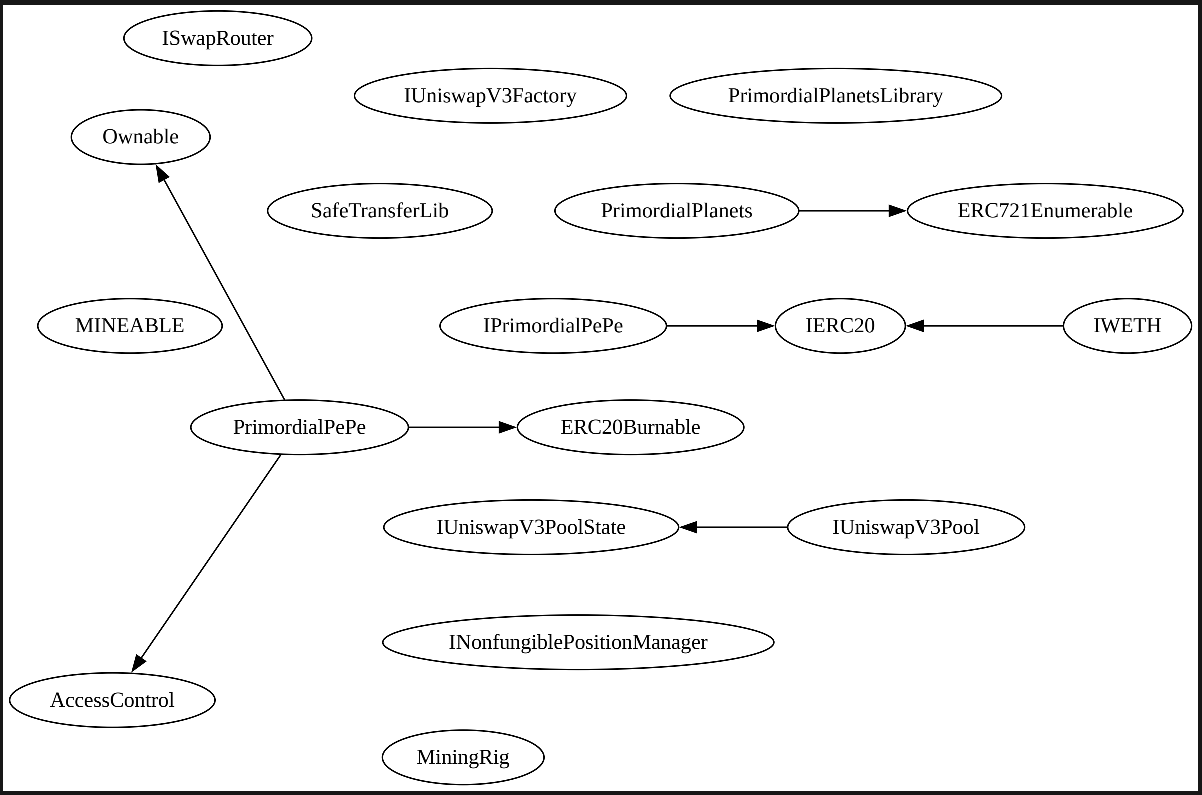
GRAPHING

Inheritance is a fundamental concept in object-oriented programming (OOP) that allows a class (referred to as a child or derived class) to inherit characteristics and functionalities from another class (known as a parent or base class). In the context of smart contracts in Solidity, inheritance is used to establish relationships between contracts, enabling code reuse, responsibility separation, and promoting modularity.

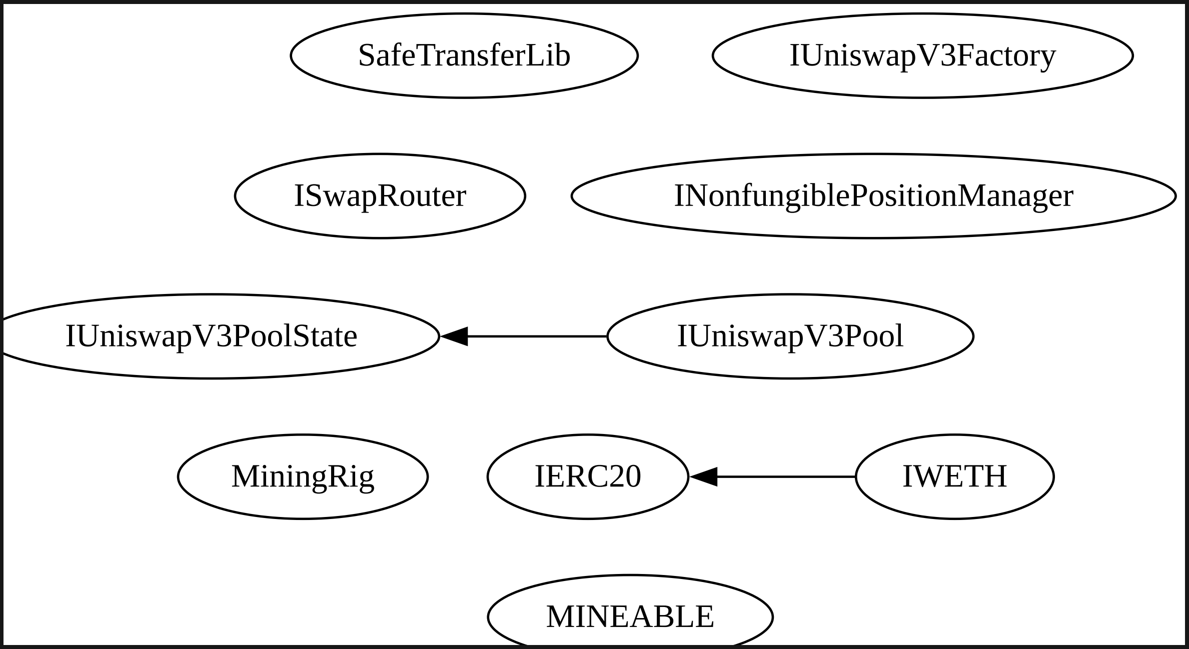
A **call graph** of a smart contract provides a visual representation of the function calls and dependencies within the contract. It illustrates the flow of execution and the relationships between functions. The call graph displays nodes representing individual functions and edges representing the calls made between them.The call graph allows for a comprehensive view of the contract's function hierarchy, enabling the identification of critical functions, entry points, and external dependencies.It highlights the paths of execution, including any loops or recursive calls, which can be crucial for understanding the contract's behavior and potential risks.

A **contract interaction** graph provides a visual representation of the relationships and interactions between different smart contracts within an ecosystem. It shows how contracts interact with each other through function calls, events, and state variables. Readers can visualize the relationships and dependencies between contracts, ensuring a comprehensive analysis of the smart contract ecosystem.The graph can be used to highlight potential security risks, communication challenges, or optimization opportunities arising from the contract interactions.

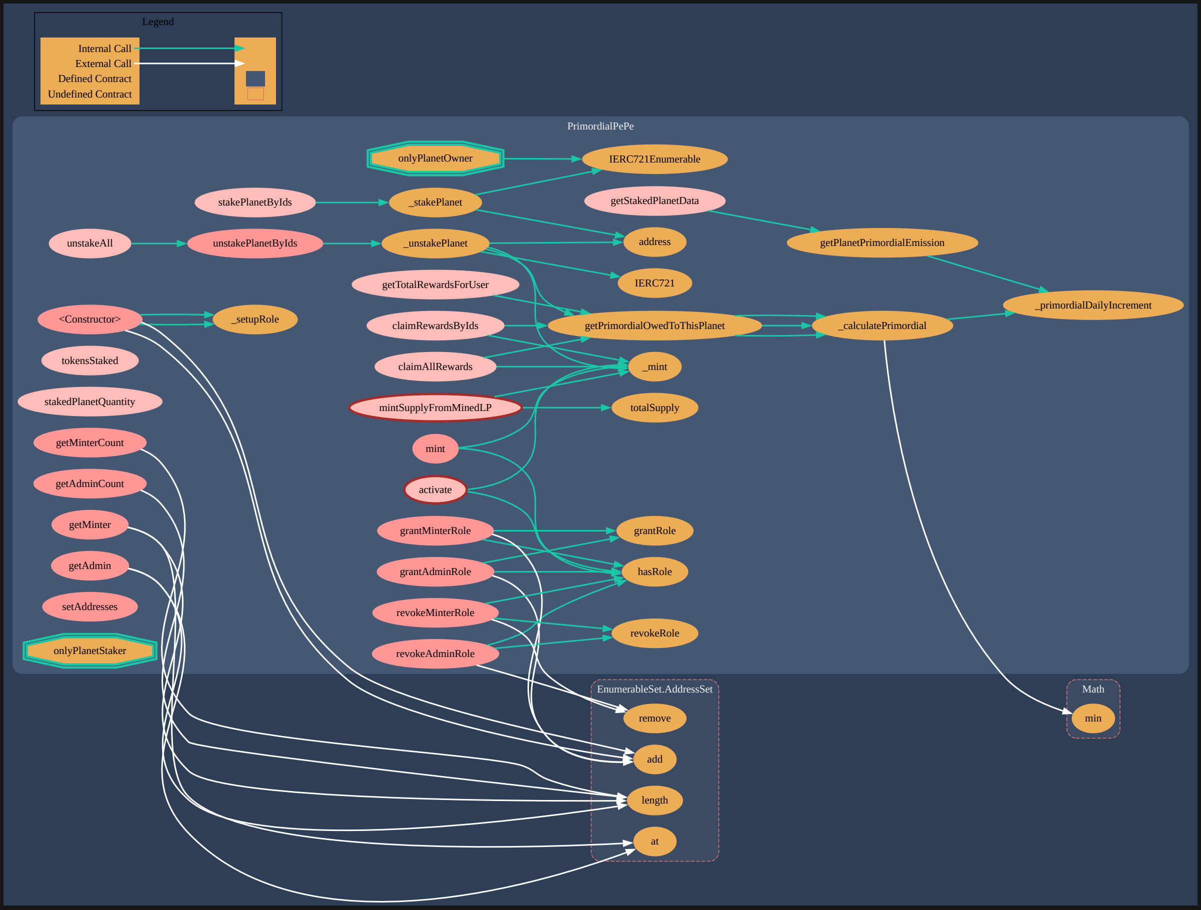
PRIMORDIALPEPE.SOL



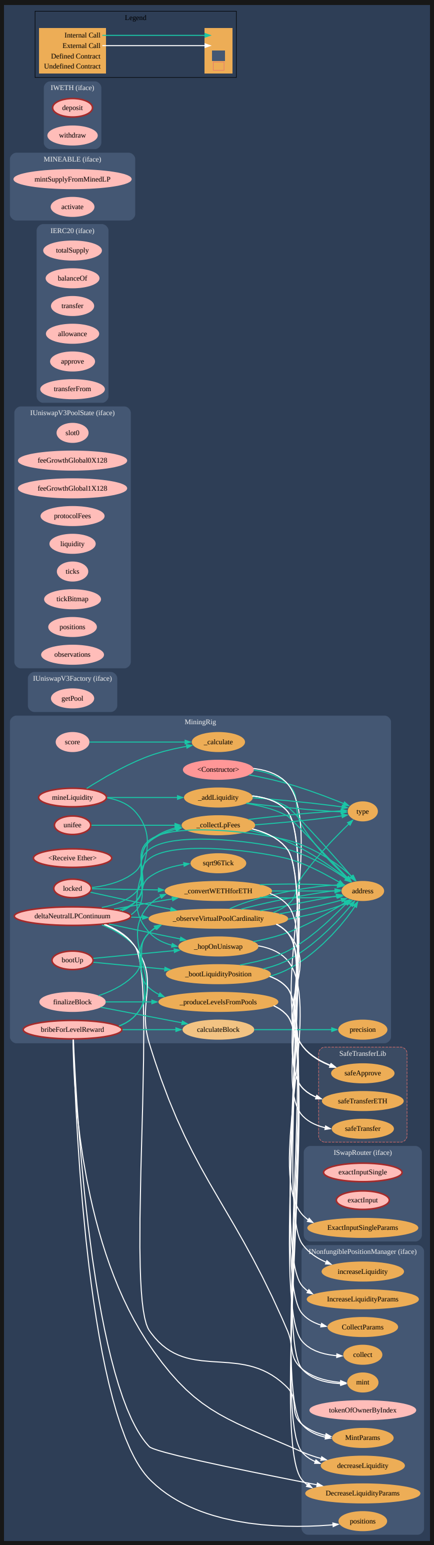
MININGRIG.SOL



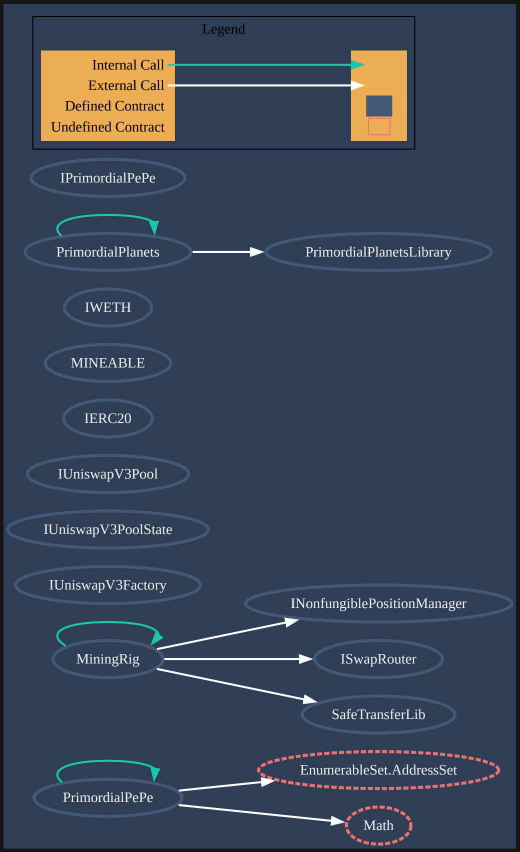
PRIMORDIALPEPE.SOL



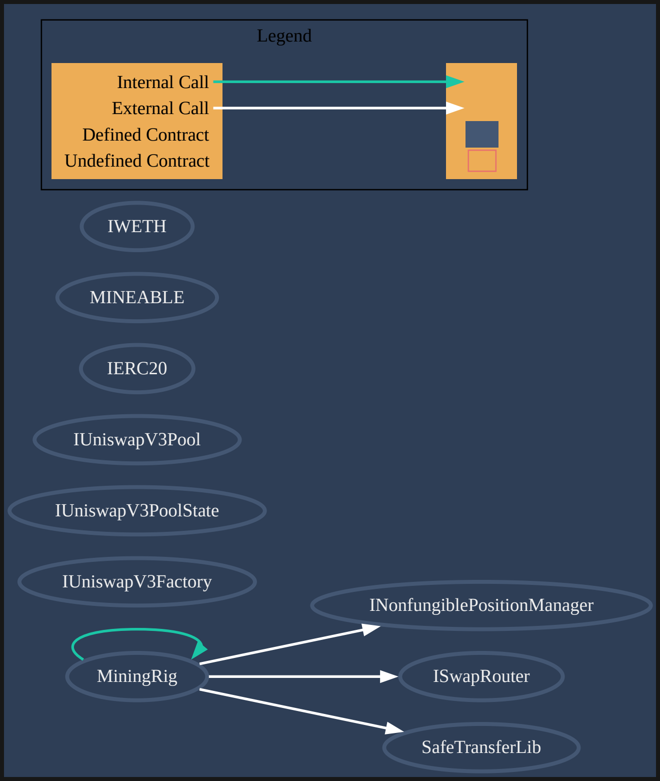
MININGRIG.SOL



PRIMORDIALPEPE.SOL



MININGRIG.SOL



ANALYSIS

In the scope of this audit, after analyzing the cyclomatic complexity of the functions present in the contracts, we can see that the majority of the functions have a complexity of 1 to 3.

This indicates that the functions in the contract are relatively simple and easy to understand. A cyclomatic complexity of 1 to 3 suggests a limited number of decision points and loops, which helps reduce the overall complexity of the contract.This facilitates contract maintenance and decreases the risk of errors related to excessive complexity.

It is important to note that cyclomatic complexity alone does not guarantee absolute security of the contract.

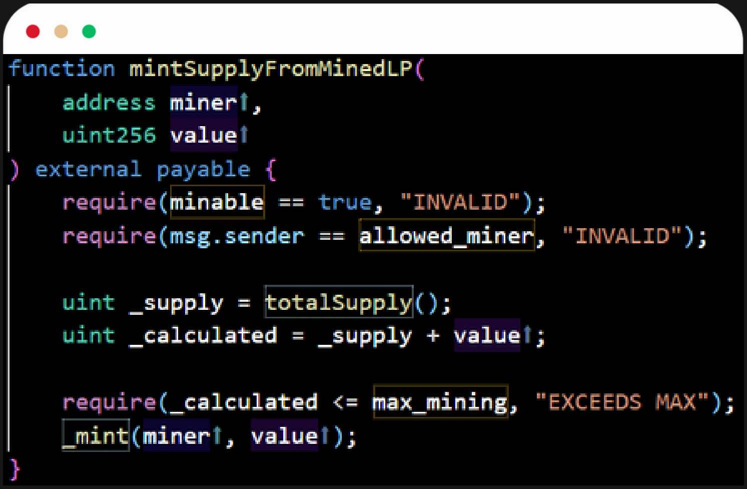
Contract: PrimordialPepe.sol

Issue: Locked Ether

Severity: **Medium**

Location: L309-315, L317-329

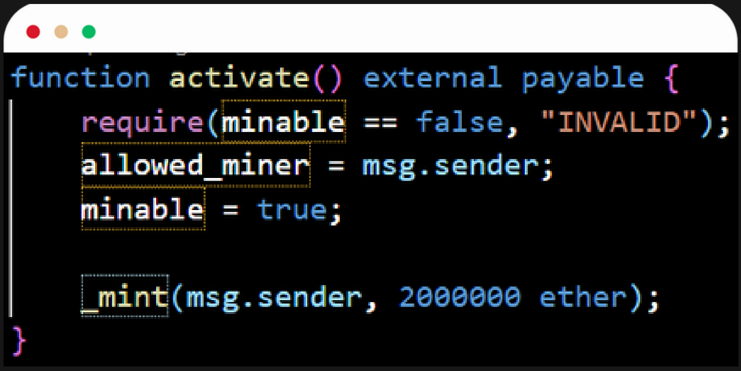
Description: Any Ether sent to this contract may be permanently stuck or "locked" inside the contract, rendering it inaccessible.



```
function mintSupplyFromMinedLP(
    address miner,
    uint256 value
) external payable {
    require(minable == true, "INVALID");
    require(msg.sender == allowed_miner, "INVALID");

    uint _supply = totalSupply();
    uint _calculated = _supply + value;

    require(_calculated <= max_mining, "EXCEEDS MAX");
    _mint(miner, value);
}
```



```
function activate() external payable {
    require(minable == false, "INVALID");
    allowed_miner = msg.sender;
    minable = true;

    _mint(msg.sender, 2000000 ether);
}
```

Comment: It's essential to address this in the contract design and provide clarity on the intended behavior. If this behavior is intentional (i.e dev wants to create a contract that can accept funds but never release them), then it is best to communicate this clearly. However, in most cases, such a design would likely be an oversight.

Allevation ✓: [Star System Labs] MiningRig is designed to have temporary access to ETH but it doesn't retain it indefinitely. The primary functionality of the MiningRig is to extend liquidity to its paired token, as influenced by the PEPE token and the Uniswap router. As a result, the liquidity provider (LP) tokens are held within the MiningRig's purview. In instances of locking or events of a similar nature, these LP tokens are subject to liquidation and are then transferred to a multi-signature contract. This mechanism is in place to facilitate the redistribution of funds among token holders and operates as the liquidity pool launch for $SDIV. When users introduce ETH into the system with the objective of mining the respective token, they are also supporting PEPE or Pond. The Rig's role here is to channel these funds into purchasing and offering liquidity, a process informally termed 'mining'.Therefore, the Rig's control over the ETH remains transient. It continues until the activation of a designated function which redirects the ETH/token to an address that is already set up with a multi-signature contract.This particular address is labeled 'COSMIC_DISTILLERY'.

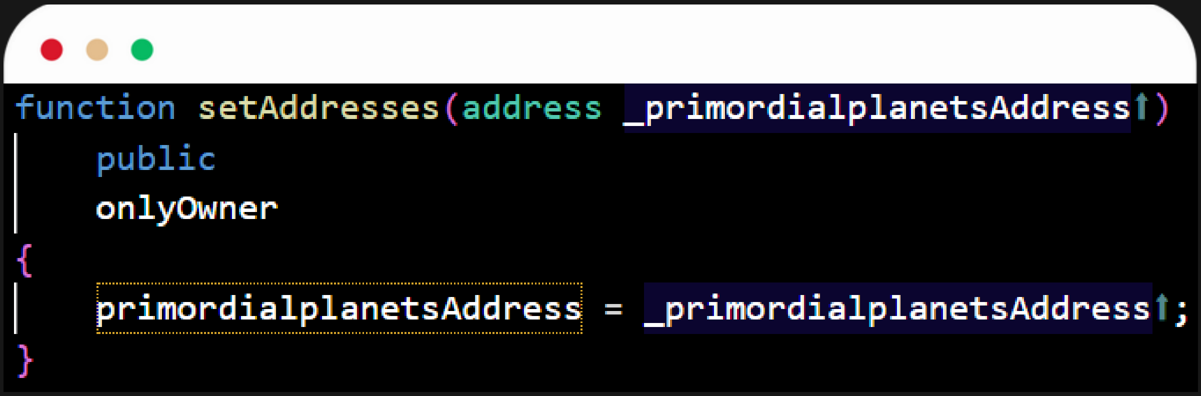
Contract: PrimordialPepe.sol

Issue: Missing Zero Address Validation

Severity: Low

Location: L333-338

Description: Lack of zero address validation. Checking for the zero-address can help to prevent errors and vulnerabilities that may arise from passing an invalid address to a function. For example, if a function transfers funds to an invalid address, the funds will be irretrievably lost.



```
function setAddresses(address _primordialplanetsAddress↑)
        public
        onlyOwner
{
        primordialplanetsAddress = _primordialplanetsAddress↑;
}
```

Comment: It is generally recommended to include a zero-address check in functions that expect an Ethereum address as a parameter. Therefore, we recommend making sure that the address is not zero by adding checks.

Allevation ✓: [Star System Labs] The address is not established during deployment as it wouldn't exist at that juncture. Consequently, implementing a non-zero check at this phase would incapacitate the function. The client has deliberately abstained from including this non-zero verification, given that the address would assume a null value for approximately 30 days before a valid address could be designated.

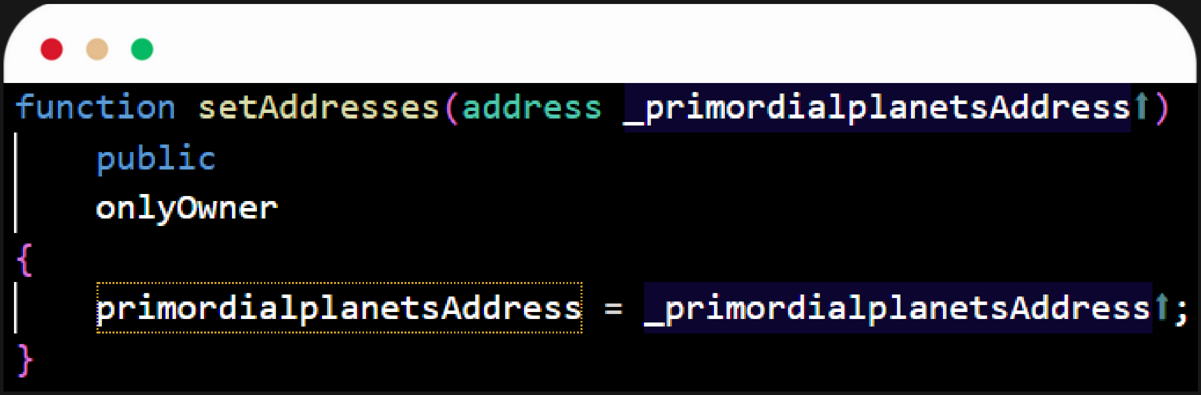
Contract: PrimordialPepe.sol

Issue: Missing Event for Access Control

Severity: Low

Location: L333-338

Description: Events that are missing for access control parameters.



```
function setAddresses(address _primordialplanetsAddress↑)
        public
        onlyOwner
{
        primordialplanetsAddress = _primordialplanetsAddress↑;
}
```

Comment: It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

Allevation ✓: The **onlyOwner** modifier is exclusively implemented to secure the capability of setting this specific address, ensuring that any designated ADMIN cannot modify it. Once the contract has been provided with the NFT address, the ownership will be renounced, thus eliminating the possibility of invoking this function. Notably, if ADMIN privileges were permitted, such a restriction would be unfeasible.

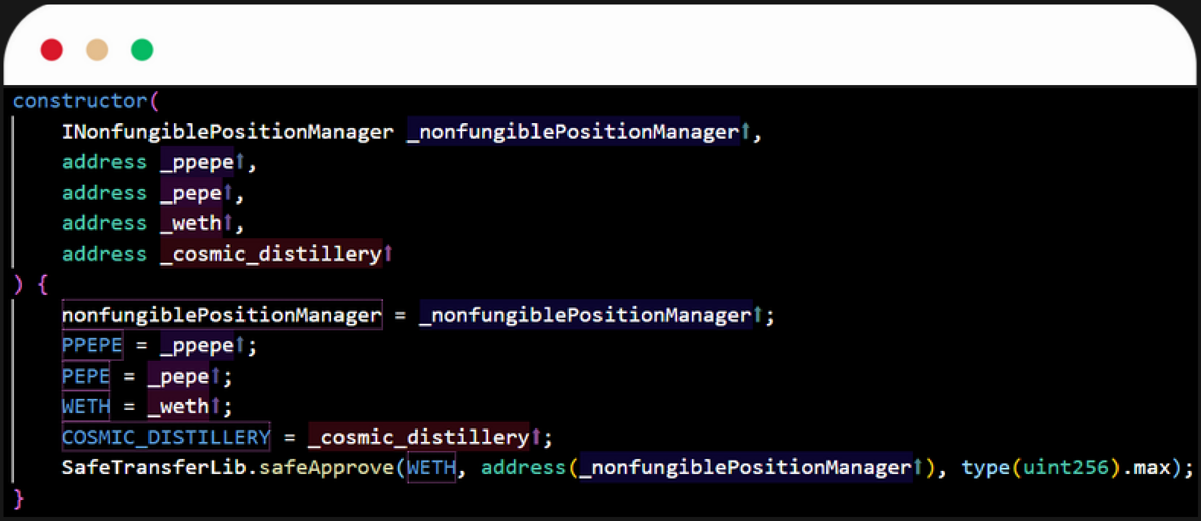
Contract: MiningRig.sol

Issue: Missing Zero Address Validation

Severity: Low

Location: L43 & 49, L44 & 50, L45 & 51, L46 & 52

Description: Lack of zero address validation. Checking for the zero-address can help to prevent errors and vulnerabilities that may arise from passing an invalid address to a function. For example, if a function transfers funds to an invalid address, the funds will be irretrievably lost.



```
constructor(
    INonfungiblePositionManager _nonfungiblePositionManager↑,
    address _ppepe↑,
    address _pepe↑,
    address _weth↑,
    address _cosmic_distillery↑
) {
    nonfungiblePositionManager = _nonfungiblePositionManager↑;
    PPEPE = _ppepe↑;
    PEPE = _pepe↑;
    WETH = _weth↑;
    COSMIC_DISTILLERY = _cosmic_distillery↑;
    SafeTransferLib.safeApprove(WETH, address(_nonfungiblePositionManager↑), type(uint256).max);
}
```

Comment: It is generally recommended to include a zero-address check in functions that expect an Ethereum address as a parameter. Therefore, we recommend making sure that the address is not zero by adding checks.

Allevation ✓: This measure has been implemented to optimize gas expenditure when conducting contract tests on the mainnet.

TESTING STANDARDS

The goal of the audit was to find any potential smart contract security problems and vulnerabilities. The information in this report should be used to understand the smart contract's risk exposure and as a guide to improving the smart contract's security posture by addressing the concerns that were discovered.

The blockchain platform is used to deploy and execute smart contracts. The platform, its programming language, and other smart contract-related applications may all have vulnerabilities that may be exploited. As a result, the audit cannot completely ensure the audited smart contract(s) explicit security on its own. Audits can't make warranties on security of the code. It also cannot be deemed a complete adequate assessment of the code's utility and safety, bug-free status, or any statements of the smart contract. While we did our best in completing the study and publishing this report, it is crucial to emphasize that you should not rely only on it; we advocate all projects doing many independent audits and participating in a public bug bounty program to assure smart contract security.

- **Gather all relevant data.**
- **Perform a preliminary visual examination of all documents and contracts.**
- **Find security holes with specialist tools & manual review with independent experts.**
- **Create and distribute a report.**

SAULIDITY

Smart Contract Audit

saulidity.com
Saulidity
@Saulidity