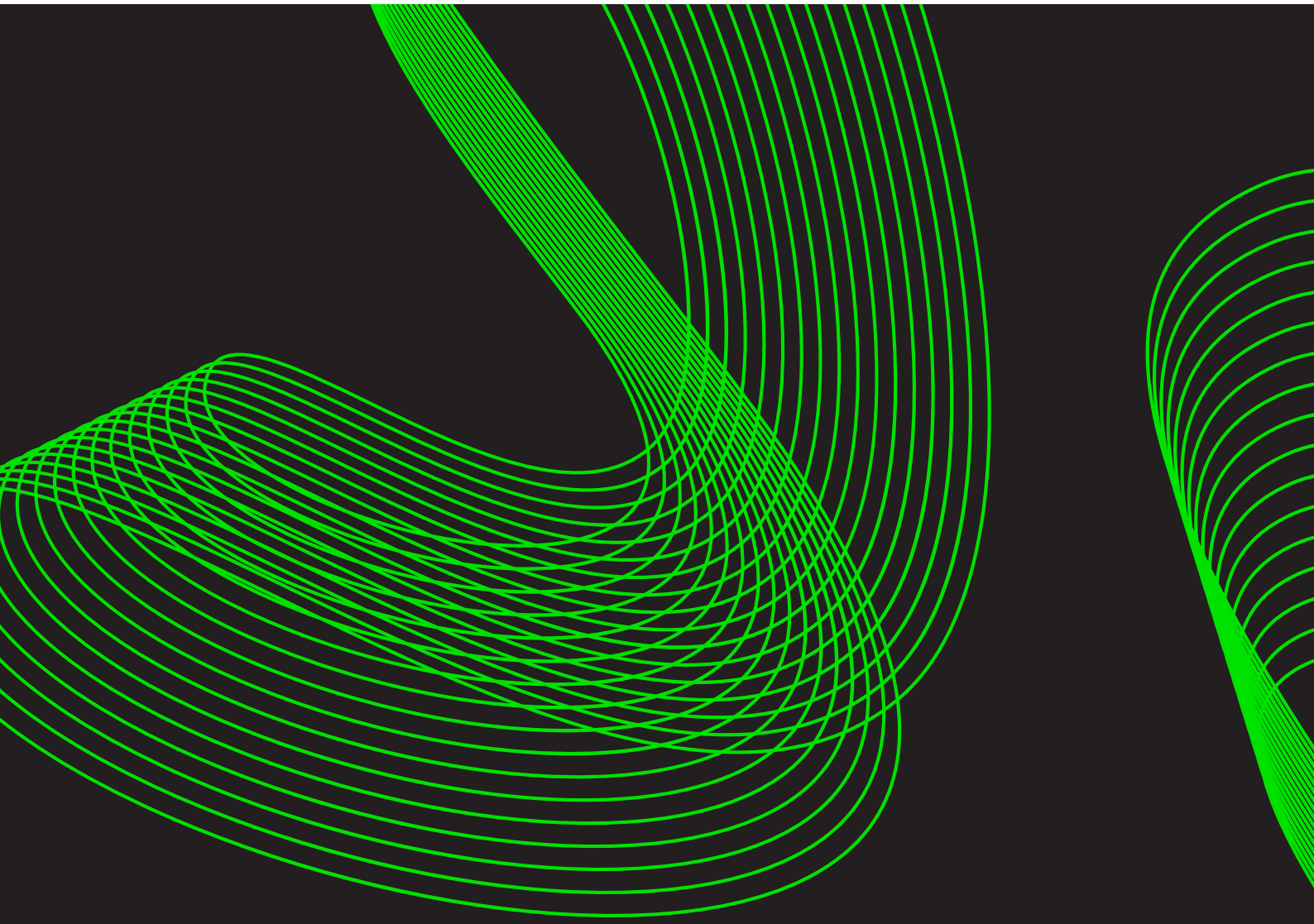
Guild Audits

JANUARY
16TH, 2025



PODY PROTOCOL FINAL AUDIT REPORT

By Guild Audits

TABLE OF CONTENTS

Disclaimer	2
Executive Summary	2
Project Summary	2
Project Audit Scope & Findings	3-5
Mode of Audit and Methodologies	6
Types of Severity	7
Types of Issues	8
Report of Findings	9-19
Closing Summary	20
Appendix	20
Guild Audit	20

DISCLAIMER

The Guild Audit team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

EXECUTIVE SUMMARY

This section will represent the summary of the whole audit.

PROJECT SUMMARY

Project Name: Pody Network Description: Pody Network has two different contracts,the PodyGift contract allows users to send gifts in the form of tokens or Ether, with configurable commission fees and gift limits and the PodyPassport contract allows users to mint nft based on the user level, users can also claim points which is determined off chain. Codebase: https://github.com/PodyNetwork/contracts Commit:https://github.com/PodyNetwork/contracts/commit/2afd5b98b 0a7106260b75bd88d7330d006212525

PROJECT AUDIT SCOPE AND FINDINGS

The motive of this audit is to review the codebase of Pody Network contracts for the purpose of achieving secured, correctness and quality smart contracts. Number of Contracts in Scope: All the contract (185 SLOC) Duration for Audit: 7 days.

Vulnerability Summary:
Total issues: 10
Total High: 0
Total Medium: 5
Total Low: 3
Total Informational: 2

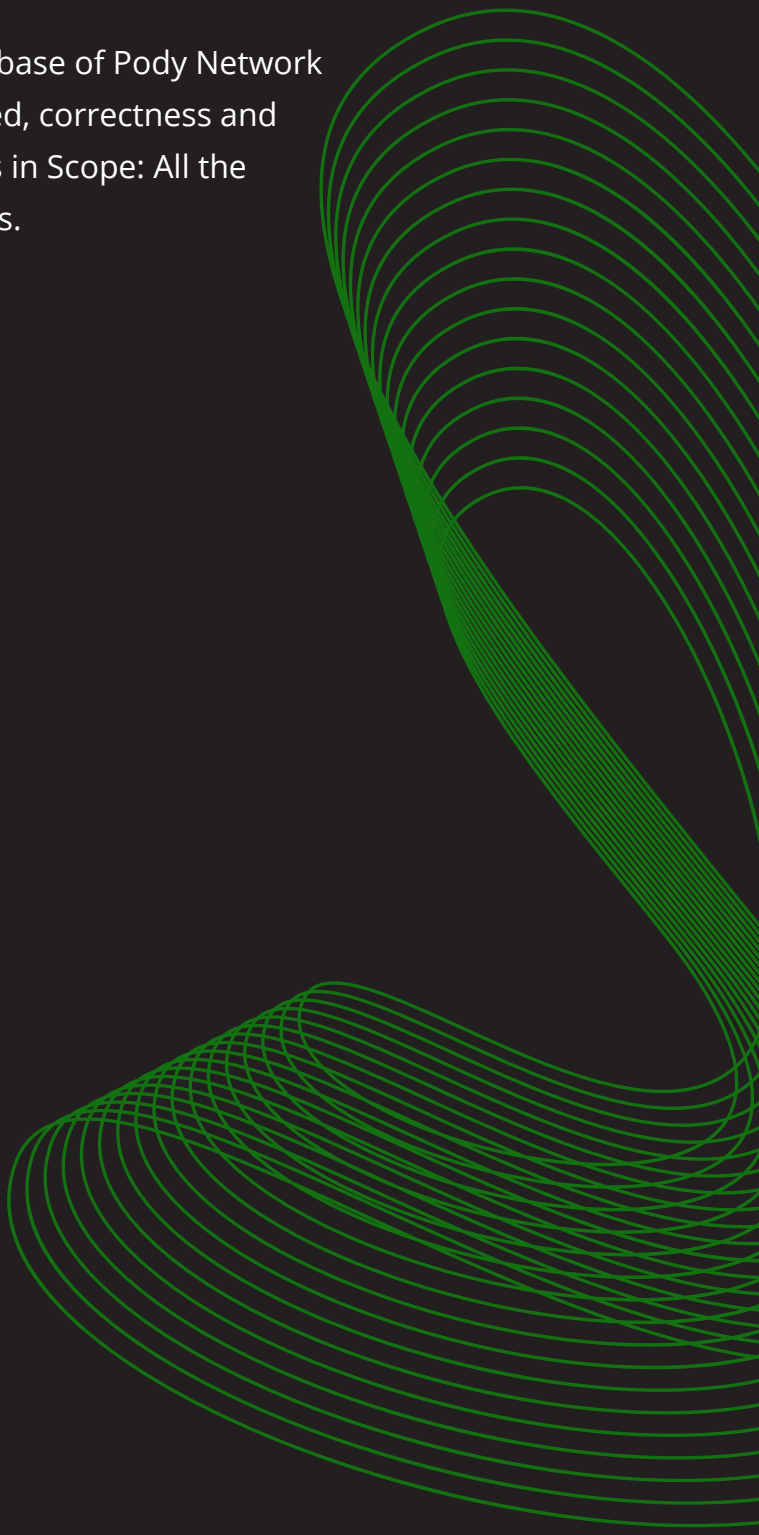
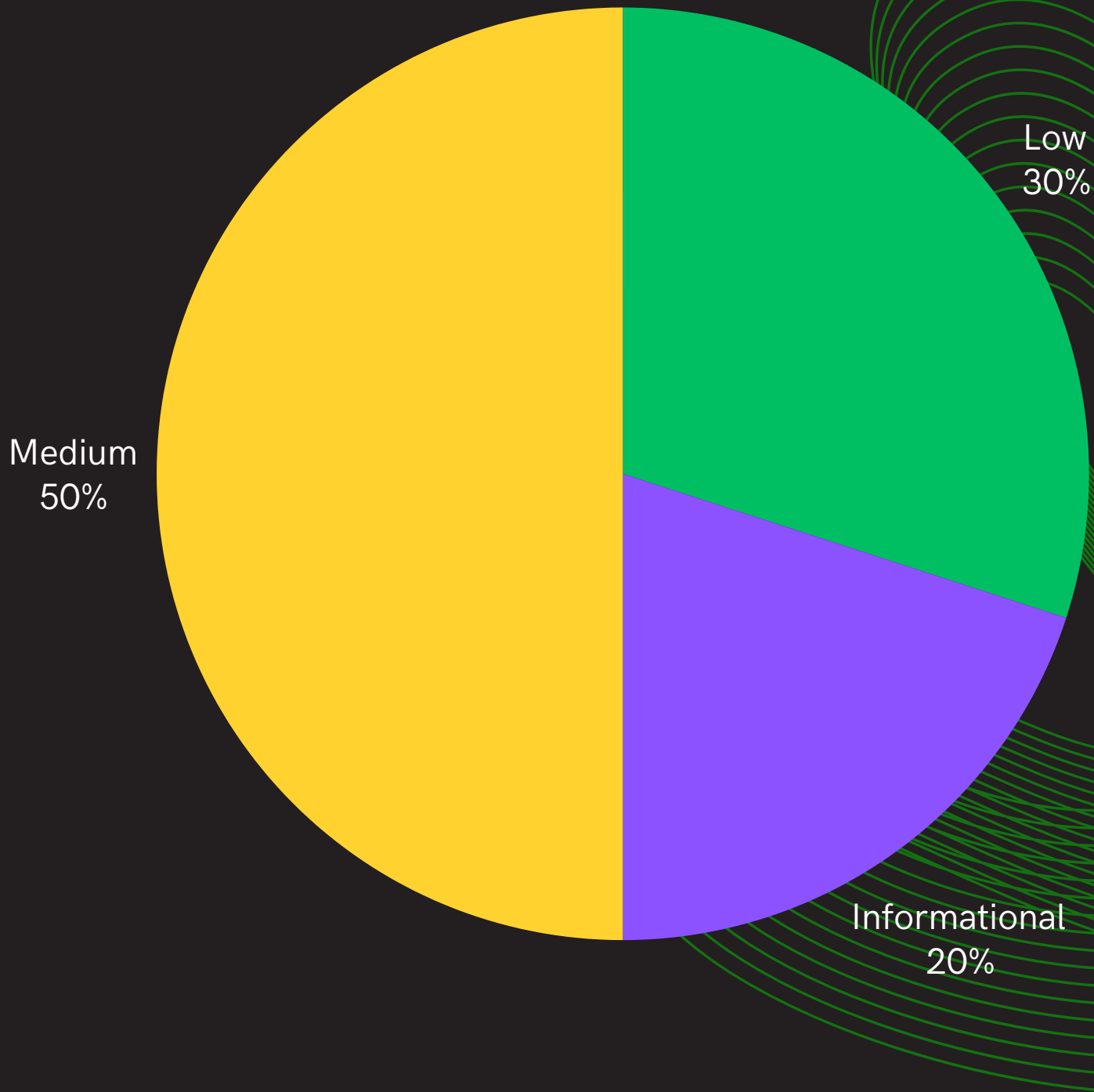


Chart Illustration



FINDINGS

Index	Title	Severity	Statusof Issues
01	[M-1] Users will lose funds if they send ether when minting for the first time	Medium	Resolved
02	[M-2] Admin can change podyToken at any time, thereby making the old podyToken worthless.	Medium	Resolved
03	[M-3] PodyGift contract can receive ether but no withdraw ether function	Medium	Resolved
04	[M-4] PodyPassport does not implement URI for ERC1155	Medium	Resolved
05	[M-5] PodyPassport::setMultiSigWallet does not check if address is a contract or multisig	Medium	Acknowledge
06	[L-1] PodyGift::giftEther uses transfer instead of call	Low	Resolved
07	[L-2] PodyPassport::mint revert with incorrect error message when msg.value is greater than price	Low	Resolved
08	[L-3] PodyPassport::withdrawERC20 assumes all erc20 token return bool	Low	Resolved
09	[L-4] Lack of Safe Transfer Handling on PodyGift::giftTokens().	Low	Acknowledge
10	[I-1] ClaimPointsOperationFailed error not used	Info	Resolved
11	[I-2] GiftEther emits address(0) instead of the standard native ether Address	Info	Resolved

MODE OF AUDIT AND METHODOLOGIES

The mode of audit carried out in this audit process is as follows:

- **Manual Review:** This is the first and principal step carried out to understand the business logic behind a project. At this point, it involves research, discussion and establishment of familiarity with contracts. Manual review is critical to understand the nitty-gritty of the contracts.
- **Automated Testing:** This is the running of tests with audit tools to further cement discoveries from the manual review. After a manual review of a contract, the audit tools used are Slither, Echidna and others.
- **Functional Testing:** Functional testing involves manually running unit, static, and dynamic testing. This is done to find out possible exploit scenarios that could be used to exploit the contracts.

The methodologies for establishing severity issues:

High Level Severity Issues
Medium Level Severity Issues
Low Level Severity Issues
Informational Level Severity Issues

Types of Severity

Every issue in this report has been assigned a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

These are critical issues that can lead to a significant loss of funds, compromise of the contract's integrity, or core function of the contract not working. Exploitation of such vulnerabilities could result in immediate and catastrophic consequences, such as:

- Complete depletion of funds.
- Permanent denial of service (DoS) to contract functionality.
- Unauthorized access or control over the contract.

Medium Severity Issues

These issues pose a significant risk but require certain preconditions or complex setups to exploit. They may not result in immediate financial loss but can degrade contract functionality or pave the way for further exploitation. Exploitation of such vulnerabilities could result in partial denial of service for certain users, leakage of sensitive data or unintended contract behaviour under specific circumstances.

Low Severity Issues

These are minor issues that have a negligible impact on the contract or its users. They may affect efficiency, readability, or clarity but do not compromise security or lead to financial loss. Impacts are minor degradation in performance,confusion for developers or users interacting with the contract and low risk of exploitation with limited consequences.

Informational

These are not vulnerabilities in the strict sense but observations or suggestions for improving the contract's code quality, readability, and adherence to best practices. There is no direct impact on the contract’s functionality or security, it is aimed at improving code standards and developer understanding.

TYPES OF ISSUES

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the audit and have been successfully fixed.

Acknowledged

Vulnerabilities that have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

REPORT OF FINDINGS

MEDIUM SEVERITY ISSUES

- **Issue:** [M-1] Users will lose funds if they send ether when minting for the first time.

- **Description:**
- When users mint for the first time, the price of the nft is zero but if a user adds ether to the transaction, the ether will be sent to the PodyPassport contract. The user would lose the ether and the ether will be stuck in the contract since there is no withdrawEther function.

- **Impact of the vulnerability**
- 1. Users minting for the first will lose ether if they send any ether.
- 2. Ether sent will be stuck in the contract.

```
function mint(address account, bytes memory data) external payable {
        User storage user = users[account];
        require(user.level < 5, "You have reached the maximum level");
        uint256 nextLevel = user.level + 1;
        if (user.level != 0) {
            require(msg.value == prices[nextLevel], "Insufficient funds
sent");
            (bool success, ) = multiSigWallet.call{value: msg.value}("");
            require(success, "Failed to send funds to multisig wallet");
            emit FundsWithdrawn(address(0), multiSigWallet, msg.value);
        }
        _mint(account, nextLevel, 1, data);
        user.hashRate = hashRates[nextLevel];
        user.level = nextLevel;
        emit NFTMinted(account, nextLevel);
    }
```

Impact

1. Users minting for the first will lose ether if they send any ether.
2. Ether sent will be stuck in the contract.

Recommendation:

Add a check to make sure the transaction reverts if a user minting for the first time sends ether in the transaction.

```
  function mint(address account, bytes memory data) external payable {
        User storage user = users[account];
        require(user.level < 5, "You have reached the maximum level");
        uint256 nextLevel = user.level + 1;
        if (user.level == 0 && msg.value > 0){
            revert();
         }
        if (user.level != 0) {
            require(msg.value == prices[nextLevel], "Insufficient funds
sent");
            (bool success, ) = multiSigWallet.call{value: msg.value}("");
            require(success, "Failed to send funds to multisig wallet");
            emit FundsWithdrawn(address(0), multiSigWallet, msg.value);
        }
        _mint(account, nextLevel, 1, data);
        user.hashRate = hashRates[nextLevel];
        user.level = nextLevel;
        emit NFTMinted(account, nextLevel);
    }
```

Status: Resolved

- **Issue:** [M-2] Admin can change podyToken at any time, thereby making the old podyToken worthless.
- **Severity:** Medium

- **Description:**

The updateToken function allows the contract owner to update the podyToken address at any time. While this provides flexibility for the contract to support a new podyToken, it introduces a significant centralization risk. Holders of the old polyToken will not be able to use it and the token will become worthless resulting in financial loss for the holders of the old polyToken.

```
function updateToken(address newTokenAddress) external onlyOwner {
        require(newTokenAddress != address(0), "Token address cannot be
zero");
        token = PodyToken(newTokenAddress);
        whitelistedTokens[newTokenAddress] = true;
        emit TokenUpdated(newTokenAddress);
        emit TokenWhitelisted(newTokenAddress, true);
    }
```

- **Impact of the vulnerability**

1. Financial Loss for holders of old podyToken

Recommendation

The owner should not be able to replace the podyToken

Status: Resolved

- **Issue:** [M-3] PodyGift contract can receive ether but no withdraw ether functionSeverity:

- **Description:**

The PodyGift contract contains both receive and fallback functions, which allow it to accept Ether. However, there is no functionality to withdraw Ether from the contract. This results in the following issues: 1. Locked Funds: Ether sent to the contract remains permanently locked, as no mechanism exists to transfer it out. 2. Accidental Deposits: Users who accidentally send Ether to the contract (via transfer, send, or call) will not be able to recover their funds.

- **Impact of the vulnerability**

1. Users or external systems may unintentionally transfer Ether to the contract, leading to irretrievable loss.

R**ecommendation**

Implement a withdrawEther function to allow authorized entities (e.g., the contract owner) to withdraw Ether from the contract.

Status: **Resolved**

- **Issue:** [M-4] PodyPassport does not implement URI for ERC1155

- **Description:**

The contract inherits the ERC1155 standard but does not implement the uri function for token metadata. The uri function is a key component of the ERC1155 standard, allowing external systems to fetch metadata for tokens using a template URI. If the uri function is not implemented or set, external tools, wallets, or DApps that interact with the contract may be unable to display token metadata correctly. This oversight results in incomplete functionality of the ERC1155 standard.

- **Impact of the vulnerability**

1. The contract fails to comply fully with the ERC1155 standard, which may lead to integration issues with marketplaces and other platforms.
2. Users or systems attempting to retrieve token metadata may encounter errors or invalid responses.

- **Recommendation**:

Implement the uri function to return a valid metadata URI. Use a placeholder with a token ID substitution mechanism such as {id} to comply with the standard.

Status: Resolved

- **Issue:** [M-5] PodyPassport::setMultiSigWallet does not check if address is a contract or multisig
- **Description:**

The setMultiSigWallet function in the PodyPassport contract allows the admin to update the multiSigWallet address. However, the function does not validate whether the provided address is a contract or a legitimate multi-signature wallet. This lack of validation introduces the following risks:

1. **Improper Address Assignment:** The multiSigWallet could be set to an externally owned account (EOA) or an invalid contract, which defeats the purpose of using a multi-signature wallet for security.
2. **Centralization Risk:** If an EOA is set, the multi-signature protection intended for critical operations is bypassed.

- **Impact of the vulnerability**

Using an EOA as the multiSigWallet undermines the additional security provided by a multi-signature setup.

- **Recommendation**:

Add Validation to Ensure the Address Is a Contract Integrate a mechanism to ensure that the address corresponds to a multi-signature wallet, such as requiring it to adhere to a known standard (e.g., Gnosis Safe).

Status: Acknowledged

LOW SEVERITY ISSUES

- **Issue**: [L-1] PodyGift::giftEther uses transfer instead of call

- **Description**:

The giftEther function uses the Solidity transfer method to send Ether. The transfer method has been known to introduce risks, particularly in the context of gas limit changes. Since the transfer method imposes a fixed 2300 gas stipend for the recipient, it can lead to reverts if the recipient is a smart contract with higher gas requirements for its fallback or receive function.

- **Impact of the vulnerability :**

1. If the recipient is a smart contract that requires more than 2300 gas to process the transfer, the transaction will fail, potentially halting the contract's functionality.

- **Recommendation:**

Replace the use of transfer with the call method to ensure greater flexibility and compatibility with recipient contracts. The call method allows specifying gas and handles the risks associated with the 2300 gas stipend limitation.

Status: Resolved

- **Issue:** [L-2] PodyPassport::mint revert with incorrect error message when msg.value is greater than price

- **Description:**

The mint function in the PodyPassport contract reverts with an incorrect error message when the msg.value exceeds the required price. This creates confusion for users interacting with the function, as the error message does not correctly represent the actual issue.

- **Impact of the vulnerability**

When msg.value is greater than price,the function reverts with Insufficient funds sent error.

- **Recommendation:**

The Bounty rewards state should be updated at the point of depositing bounty rewards.

Status: **Resolved**

- **Issue:** [L-3] PodyPassport::withdrawERC20 assumes all erc20 token return bool
- **Description:**

The withdrawERC20 function assumes that all ERC-20 tokens conform to the ERC-20 standard by returning bool values for transfer and transferFrom functions. However, not all tokens strictly adhere to this standard. For example, tokens like Tether (USDT) and Binance-Pegged tokens do not return a bool and instead rely solely on transaction success or failure. This assumption can lead to compatibility issues, missed failures, or unexpected behavior.

- **Impact of the vulnerability :**

Lack of reference to actual depositors when recording deposited bounty rewards can have significant implications in the context of a bounty contract. It can lead to a lack of transparency and accountability.

- **Recommendation**
- The actual depositor should be accounted for when updating the state of the bounty rewards storage. Also, the recording should be made on transfer.

```
function withdrawERC20(IERC20 token, address to, uint256 amount) external
onlyOwner {
        require(address(token) != address(0), "Invalid token address");
        require(to != address(0), "Invalid recipient address");
        require(amount > 0, "Amount must be greater than zero");
        uint256 balance = token.balanceOf(address(this));
        require(balance >= amount, "Insufficient token balance");
        bool success = token.transfer(to, amount);
        require(success, "Token transfer failed");
        emit FundsWithdrawn(address(token), to, amount);
    }
```

- **Recommendation**

Use SafeERC20 from the OpenZeppelin library

Status: Resolved

- **Issue:** [L-4] Lack of Safe Transfer Handling on PodyGift::giftTokens().
- **Description:**

The giftTokens function allows a user to send tokens to a recipient with a commission deducted for the contract owner. While the design includes basic checks for token whitelisting, amount constraints, and valid addresses, there is a vulnerability due to unsafe assumptions about the behavior of whitelisted ERC20 tokens. The issue arises from the assumption that all ERC20 tokens revert on failure. However, non-standard ERC20 tokens do not revert but instead return a boolean (false) on failure. The current implementation fails to check this return value, leading to a false assumption of successful transfers.

Additionally, the approve function in some tokens may allow zero-value approvals.

- **Impact:**

A user can approve an amount of tokens and attempt to send tokens with an amount exceeding the approved balance. If the transfer does not revert on insufficient allowance, the transaction will fail silently without any token transfer, yet the GiftSent event will still be emitted, giving a false impression of a successful transfer.

- **Recommendation**

1. Use of openzeppelin SafeERC20 library.
2. Whitelisted tokens must match Pody Network's token integration.

Status: Acknowledged

INFORMATIONAL ISSUES

- **Issue:** [I-1] ClaimPointsOperationFailed error not used

- **Description**:

The ClaimPointsOperationFailed error was not used in the contract.

- **Impact of the vulnerability**

1. Takes unnecessary space in the contract.

- **Recommendation**

Remove the error message

Status: Resolved

- **Issue:** [I-2] GiftEther emits address(0) instead of the standard native ether

- **Address:**

In the giftEther function, the contract emits address(0) as the token address to represent native Ether payments. However, this approach deviates from the convention of using a standard placeholder, such as 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE, to represent native Ether in the context of smart contracts.

- **Impact of the vulnerability**

1. External tools, wallets, or DApps may fail to recognize address(0) as a representation of Ether, leading to integration issues.

- **Recommendation**

Use the standard placeholder address 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE to represent native Ether in the emitted event.

Status: Resolved

Closing Summary

There were discoveries of some high, medium, low and informational issues after the audit. The audit team thereafter suggested some remediation to help remedy the issues found in the contract.

Appendix

- Audit: The review and testing of contracts to find bugs.
- Issues: These are possible bugs that could lead exploits or help redefine the contracts better.
- Slither: A tool used to automatically find bugs in a contract. Severity: This explains the status of a bug.

Guild Audits

Guild Audits is geared towards providing blockchain and smart contract security in the web3 world. The firm is passionate about remedying the constant hacks and exploits that deters the web3 motive.