Psydelve StakingV1 Audit Report

15th July 2023

Executive Summary

This audit aimed to assess the security of the Psydelve StakingV1 smart contract with a thorough review of the Solidity codebase, testing for common smart contract vulnerabilities, and evaluating the business logic of the contract to ensure it aligned with intended functionality. The code review identified several areas that require attention in order to improve the overall quality and security of the codebase. The most significant issue is the admin sending the reward NFT to zero address, which cannot be recovered later on. Another noteworthy issue is not using SafeMath and SafeCast libraries. Furthermore, There are issues with the timeStaked function and the use of a floating pragma, which does not specify a fixed version, may introduce compatibility issues or unintended behaviour when compiling the code with different Solidity compiler versions. Lastly, there is an informational issue regarding the compiler bugs and the usage of the _cleanupMetadata function.

Severity classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - the technical, economic and reputation damage of a successful attack

Likelihood - the chance that a particular vulnerability gets discovered and exploited

Severity - the overall criticality of the risk

Overview

Project Name	Psydelve StakingV1
Language	Solidity
CodeBase	PsydelveStakingV1.sol

Audit Methodology	Business Logic & Code Review

Code Vulnerability Review Summary

The following number of issues were found, categorised by their severity:

- Medium: 1 issues
- Low: 3 issues
- Informational : 2 issues

ID	Title	Category	Severity
M - 1	**Risk of Airdropping Tokens to the `address(0)` Due to Missing Check**	**Logical**	**Medium**
L - 1	**Not using SafeMath and SafeCast**	**Language Specific**	**Low**
L - 2	**`timeStaked` function returns incorrect value if NFT is not staked**	**Logical**	**Low**
L - 3	**Use of floating pragma**	**Code Style**	**Low**
I - 1	**Compile Errors**	**Code Style**	**Informational**
I - 2	**Redundant use of `_cleanupMetadata` function**	**Code Style**	**Gas Optimisation**

[M - 1] Risk of Airdropping Tokens to the `address(0)` Due to Missing Check

Severity

Impact: High, The rewardsToken NFT cannot be recovered.

Likelihood: Low, as the admin needs to call the airdrop function by passing NFT ids which are currently not staked.

Description

The `airdrop` function poses a potential risk due to a missing check for the zero address (`address(0)`). In the absence of this check, tokens can inadvertently be transferred to the zero address, which is a special address that represents an invalid or non-existent Ethereum address. Transferring tokens to the zero address effectively renders them irrecoverable and permanently locked. It is crucial to include

a validation step within the airdrop function to ensure that the metadata's owner address is not `address(0)` before initiating the token transfer. Implementing this check will help prevent unintended loss of tokens and maintain the integrity of the airdrop process.

The issue arises due to the fact that the owner's address is set to zero but the `lastUpdated` is not as part of the withdraw function(when the user withdraws the staked NFT).

[**Recommendation] :** Add a check to verify that `metadata.owner` is not `address(0)`

```
//Add the following check in the "airdrop" function
if (metadata.owner == address(0)) {
   revert OwnerZeroAddress();
}
```

[**Discussion]**

Issue Fixed.

[L - 1] Not using SafeMath and SafeCast

Description

Throughout the codebase there are math operations that are not checked for overflow or underflow using the SafeMath. Even though overflow/underflow issues are resolved post solidity version 0.8.0, since there are downcasting and other math operations performed between state variables of different types as shown below, It poses a risk.

```
struct StakingMetadata {
  address owner;
  uint40 lastUpdated; //Instance #1
}

uint16 public airdropAmount; //Instance #2

metadata.lastUpdated = uint40(block.timestamp);  //Instance #3

if ((block.timestamp - metadata.lastUpdated)/86400 < airdropInterval){ //Instance #4
  revert NotEnoughTimeElapsed();
}

metadata.lastUpdated = metadata.lastUpdated + (airdropInterval * 86400) //Instance #5
```

[**Recommendation] :** Consider using OpenZeppelin's SafeMath and SafeCast library to prevent unexpected overflows or underflows when casting.

1. SafeCast - https://docs.openzeppelin.com/contracts/3.x/api/utils#SafeCast
2. SafeMath - https://docs.openzeppelin.com/contracts/2.x/api/math#SafeMath

Another solution would be to declare all the state variables as `uint256`

[**Discussion]**

Issue Fixed.

[L - 2] `timeStaked` function returns incorrect value if NFT is not staked

Description

The `timeStaked` is used to find out the time for which a NFT was staked. `metadata.initiallyStaked` is used to track when the NFT was initially staked but it is never set to zero in the `withdraw` function, As a result, If a NFT was withdrawn the `timeStaked` function will return an incorrect value. Furthermore if the `metadata.initiallyStaked` is zero, meaning the NFT is not staked, the function returns the current `block.timestamp` instead of zero.

[**Recommendation] :** Implement the below changes in the code to fix the issues.

```
//Set the metadata.initiallyStaked to zero in the withdraw function
function withdraw(uint256[] calldata _tokenIds) external nonReentrant {

  // ...
  // ...

  // Unfreeze NFTs
  _unstakeNft(_tokenIds[i]);
  // Cleanup the staking metadata for the current tokenId
  metadata.owner = address(0);
  metadata.initiallyStaked = 0;

  // ...
  // ...

}
--------------------------------------------------------------------------
// update the timeStaked to return zero in case the NFT is not staked
function timeStaked(uint32 _tokenId) external view returns (uint256) {
    StakingMetadata memory metadata = stakingMetadata[_tokenId];

    // Check if the NFT is staked (metadata.initiallyStaked is not zero)
    if (metadata.initiallyStaked == 0) {
```

```
            return 0; // NFT is not staked, return zero as the staking time
        }

        // Calculate the time staked and return the result
        return (block.timestamp - metadata.initiallyStaked);
    }
```

[**Discussion]**

Issue Fixed.

[L - 3] Use of floating pragma

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.

Reference - https://swcregistry.io/docs/SWC-103

[**Recommendation] :** Lock the pragma version to version 0.8.19 and also consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

[**Discussion]**

Issue Fixed.

[I - 1] Compile Errors

Description

The `PsydelveStakingV1.sol` fails to compile due to some minor issues as shown below.

1. Missing `admin` state variable definition.
2. Custom error `InvalidManagerAddress()` not defined.
3. Missing semicolon(;) in airdrop function where `metadata.lastUpdated` is calculated

[**Recommendation] :** Fix the bugs

[**Discussion]**

Issue Fixed.

[I - 2] Redundant use of `_cleanupMetadata` function

Description

The `_cleanupMetadata` is called as part of `withdraw` function, where in users un-stake their NFTs. The function checks if `metadata.owner` is set to zero and then updates the `metadata.lastUpdated` to the current timestamp.

Reading from the storage is a costly operation and the same data is being read twice.

[**Recommendation] : Move the** `_cleanupMetadata` to the `withdraw` function.

```
// Unfreeze NFTs
_unstakeNft(_tokenIds[i]);

//Remove the "_cleanupMetadata"
//_cleanupMetadata(_tokenIds[i]);

// Cleanup the staking metadata for the current tokenId
metadata.owner = address(0);

//the "_cleanupMetadata" code added post metadata.owner set to address(0)
metadata.lastUpdated = uint40(block.timestamp);
```

[Discussion]

Issue Fixed.

Conclusion

The Psydelve StakingV1 smart contract exhibits a strong commitment to security. However, the audit has identified several areas where improvements can be made to enhance both the contract's security and the user experience. To ensure the ongoing security and success of the Psydelve Staking project, addressing these findings through code modifications and improvements will help enhance the codebase's security, maintainability, and adherence to best practices. It is recommended to prioritise the resolution of the issues based on their severity and potential impact on the system.

Update: The developer has fixed all the reported vulnerabilities reported. The Psydelve Staking V1 contract is ready to go live.