SMART CONTRACT AUDIT

zokyo

March 14th 2023 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

TECHNICAL SUMMARY

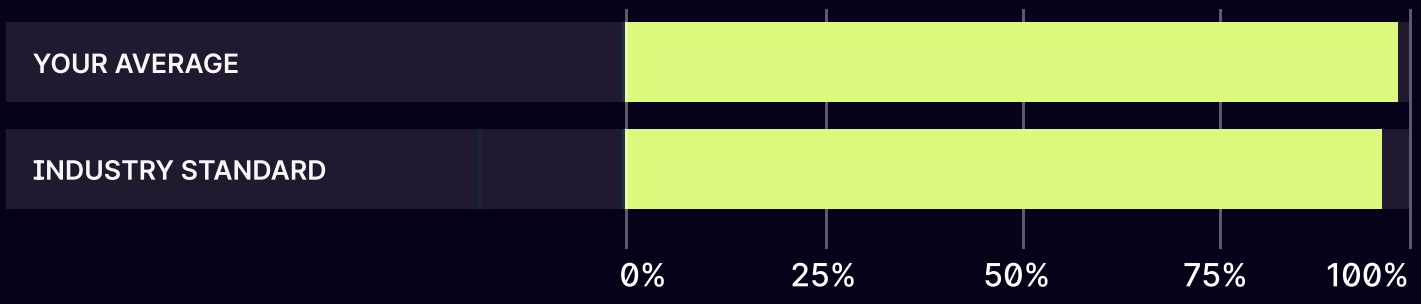
This document outlines the overall security of the DPNMDEFI smart contracts evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the DPNMDEFI smart contracts codebase for quality, security, and correctness.

Contract Status

There were 0 critical issues found during the audit. (See Complete Analysis)

Testable Code



96.28% of the code is testable, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contracts but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the DPNMDEFI team put in place a bug bounty program to encourage further active analysis of the smart contracts.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the DPNMDEFI repository: https://github.com/tguy6/dpnm_audit

Last commit: 8e4592d30fcbeef8af31bd895fc22160fbeb9558

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- dpnm_sc.sol
- gwt_new.sol
- phenomanelTree.sol

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most resent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of DPNMDEFI smart contracts. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. Part of this work includes writing a test suite using the Hardhat testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.	03	Testing contracts logic against common and uncommon attack vectors.
02	Cross-comparison with other, similar smart contracts by industry leaders.	04	Thorough manual review of the codebase line by line.

Executive Summary

Contracts are well written and structured. There was no critical issue found during the audit . All the mentioned findings may have an effect only in case of specific conditions performed by the contract owner and the investors interacting with it. They are described in detail in the “Complete Analysis” section.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the DPNMDEFI team and the DPNMDEFI team is aware of it, but they have chosen to not solved it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk	Status
1	Transferring Ownership can lead to undefined behavior	High	Resolved
2	Tree Level incompatible	High	Resolved
3	Exposure to sandwich attack	High	Acknowledged
4	Modifier onlyAllowed potentially causes DoS	High	Resolved
5	totalUsers of tree is missing one	High	Resolved
6	Buying limits can be broken with a significant impact	High	Acknowledged
7	Underflow/overflow cases are not avoided	High	Resolved
8	Tree does not extend as expected	High	Resolved
9	Transfer of busd to same recipient is unnecessarily repeated in loops	Medium	Resolved
10	Optimize gas cost needed for transfer of busd to users	Low	Acknowledged
11	Exposure to revert due to underflow	Medium	Resolved
12	Centralization Risk	Medium	Acknowledged
13	Unchecked return	Medium	Resolved
14	Pragma version lock	Low	Resolved
15	Some public visibilities are unneeded	Low	Resolved

#	Title	Risk	Status
16	allowedContracts is publicly viewable while it should not	Low	Resolved
17	Unnecessary assignment of array in treeuserlevels	Low	Resolved
18	Dangerous nested loops	Low	Acknowledged
19	Data representing tree users, mismatch can take place	Low	Acknowledged
20	Interactions preceding the effects	Low	Resolved
21	No uint value input validation applied on methods	Low	Resolved
22	Wasteful array representation for variable	Low	Resolved
23	Variable to be declared as constant	Informational	Resolved
24	Misleading variable name	Informational	Resolved
25	No validation on input address	Informational	Resolved
26	Code logic repeated can be replaced by function	Informational	Resolved
27	Replace a frequently used number by a constant	Informational	Acknowledged
28	Unused variable	Informational	Resolved
29	Gas Optimization	Informational	Resolved
30	Wrong Function description	Informational	Resolved

HIGH | RESOLVED

Transferring Ownership can lead to undefined behavior

phenomanelTree.sol - Transferring ownership after deployment can cause a class of issues because the logic is built upon about the assumption that the root of the tree is the owner of the contract. Effects will follow when transferOwnership is used: 1- isUserExist will return faulty result because it assumes owner is the root, hence old owner might show that he does not exist in the tree anymore and be added to the tree again. 2- Similarly, getUserDistance will return a faulty result because it assumes owner to be the root of the tree.

Recommendation:
Rather than assuming owner to be the root of the tree, introduce a new address variable that is immutable and assigned to the root of the tree on deployment.
Address immutable public rootOfTree;

Fix
In commit `60761c1`, new variable `rootOfTree` is assigned to the first user added to the tree. Issue is resolved, provided that the deployer of phenomanelTree is the same deployer of dpnm_sc.

HIGH | RESOLVED

Tree Level incompatible

dpnm_sc.sol / phenomanelTree.sol - tree depth is incompatible between the dpnmMain and phenomanelTree. Contract dpnmMain assumes depth of tree is 10 while phenomanelTree is built to be 15.

Recommendation:
Match the tree depth.

Fix
It is meant to be as a spec in order to leave room for future products as according to developers depth will be adjustable up to 15.

HIGH | ACKNOWLEDGED

Exposure to sandwich attack

dpnm_sc.sol - buydPNM & sellDPNM are exposed to slippage sudden price change, hence an attacker can take advantage of that by manipulating price to turn against a victim.

Recommendation:
A suggestion proposed, add an argument to the methods to represent the slippage in price like this: buydPNM(uint BUSDamount, uint slippage) or buydPNM(uint BUSDamount, uint minAmountdPNMReceived). This new argument will be validated by a require statement like require(userTotaldPNMdeposit >= minAmountdPNMReceived). In that case an undesirable change in price would make the trade revert.

Fix
Issue has been acknowledged by developers, and they showed that incentive of the attacker to exploit such vulnerability is estimated to be insignificant, hence we consider this is part of the game theory/business logic of the product.

HIGH | RESOLVED

Modifier onlyAllowed potentially causes DoS

phenomanelTree.sol - modifier onlyAllowed() is applying an exhaustive search on array elements of allowedContracts . Given that allowedContracts can be appended indefinitely without even an option to remove elements, this can potentially lead to a denial of service on the methods using the modifier.

Recommendation:
use a map for allowedContracts

mapping(address ⇒ bool) allowedContractsMap;

HIGH | RESOLVED

totalUsers of tree is missing one

phenomanelTree.sol - totalUsers of tree reflects the number of participants in the tree. Despite that, right on deployment on constructor call, we have the owner of the contract added as one of the participants but totalUsers is not incremented and stays holding a zero value.

```
    constructor() {
        treeUsers[msg.sender] = Tree(address(0),address(0),address(0),address(0));
        treeuserlevels[msg.sender] = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];

    }

treeUsers[msg.sender] =
Tree(address(0),address(0),address(0),address(0));
treeuserlevels[msg.sender] = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
// NB: missing initializing totalUsers
```

What makes the issue of a High severity is that it messes the synchronization with the corresponding value of totalUsers in dpnm_sc contract.

Recommendation:

- Ensure totalUsers is equal to one on deployment to match the number of actual users.
- A better recommendation is to refactor the code in a way that you rely on a single source of truth. In this situation, better rely on totalUsers in phenomanelTree contract after correcting its value.

Fix
Issue fixed in 60761c1.

HIGH | ACKNOWLEDGED

Buying limits can be broken with a significant impact

dpnm_sc.sol - In method getMaxDailyBuy a user can break the limits significantly by applying a certain pattern of buys and sells within a 2-day window. It is understood though that paying fees on those buy/sell operations serve as a disincentive for the users (i.e., attackers in this case). Attackers can still apply the pattern if the wins overcome the fee loss. This can also serve the attackers to make an impact on price against a victim buyer, which is another issue mentioned in this report.

Pattern:

```
Day-1 :   | buy x -> sell x -> buy x -> sell x ... 100 times          |
Day-2:    | Wait for buy period to pass
|
Day-3 :   | buy 100x   (breaking the limit)
|
```

The issue mainly arises from the fact that the sell period is bigger than the buy period.

Recommendation
Choose the sell period to be at least 35 seconds less than the buy period (24 hours). In other instead of having soldLast48Hours let it be like soldLast24HoursMinus35Seconds by applying this check users[user].tokenData.lastSellTime + 24 hours – 35 seconds > block.timestamp

Instead of:

```
if (users[user].tokenData.lastSellTime + 48 hours > block.timestamp) {
    soldLast48Hours = users[user].tokenData.lastSellAmount;
}
```

Fix
Issue has been acknowledged by developers and the incentive of the attacker has been estimated by a real world example and shown to be insignificant, hence we consider this is part of the game theory/business logic of the product.

HIGH | RESOLVED

Underflow/overflow cases are not avoided

phenomanelTree.sol - Testing has shown that in cases where we are filling trees for different users, sometimes unexpected revert messages are received (i.e. related to underflow/overflow) while the expected message according to the use case is `Tree is filled`. By investigating this we see that underflow cases are not avoided, for instance we have the following (i.e. on calling `positionUser`):

- `lvlsDeep ->` take any value `0`-255.
- Triggers `findPositionSpot`, we find that `lvlsDeep` should have been validated to be non-zero because of `lvlsDeep-lvl` and `lvl` starts by `1`.
- Triggers `calcTreeNetwork`, we find `_depth` which should be at least `1` to have a proper for loop. Since we have `lvlsDeep-lvl=_depth` while maximum value of `lvl` can be `lvl+1` we see it ends up having `_depth = -1` which is not possible as it reverts.
- Also, based on previous point, `lvlsDeep-lvl` should be less than `15`. Otherwise, it goes out of bounds of array index. Therefore, `lvlsDeep < 16` because `lvl >= 1`.

Underflow/Overflow might be more severe if we face those errors in valid states; we need to be careful to avoid them.

Recommendation:

After Refactoring the data representation of `phenomanelTree` as recommended in other issues. This part needs to require statements that validate input to be added in the right places in order to avoid unintended revert messages. Reimplementing this method in order to avoid that is also needed.

Fix

Require statement has been added in order to validate that input is within an acceptable bound. Code fix is presented in `60761c1`.

HIGH | RESOLVED

Tree does not extend as expected

phenomanelTree.sol - Starting from `positionUser`, testing showed that the tree of a given referrer does not get filled as expected (i.e. according to given depth). For instance, if depth is 11 we expect to have 265720 nodes but only 1093 are added.

Recommendation:
Algorithm and parameters used in checks might need to be readjusted. It is noticed that the capacity of nodes gets decreased according to a pattern to fit a smaller depth.

Fix
Tree of each individual user is shown in test to be filled up to the expected capacity after fix is carried out on commit `05ec745` .

MEDIUM | RESOLVED

Transfer of busd to same recipient is unnecessarily repeated in loops

dpnm_sc.sol - `busd.safeTransfer(feeCollector, ...)` is repeated in loop in both methods: `_TreePayment` and `depositFordPNMBuy`, `safeTransfer` is a costly operation, and it can be called only once on the total amount to be transferred, rather than repeating it unnecessarily.

Recommendation:
Declare a variable for the `amountToBeTransferred` and increase it each time you need to make the transfer. After reaching the target `amountToBeTransferred` do the actual transfer.

```
uint256 amountToBeTransferred;

for (uint i=0; i <10; i++)
   amountToBeTransferred += amountToBeAdded
busd.safeTransfer(feeCollector, amountToBeTransferred);
```

Fix
Codefix in `60761c1` introduced `amountToBeTransferredToFeeCollector` in `_TreePayment`. Also we have `feeCollectorBonus` in method `depositBonusFordPNMBuy`.

LOW | ACKNOWLEDGED

Optimize gas cost needed for transfer of busd to users

dpnm_sc.sol - in `depositBonus` Issue is similar to `Transfer of busd to same recipient is unnecessarily repeated in loops`, but it is dealt with in a different way. Method `depositBonus` includes `busd.safeTransfer(userAddress, ...)`. Mehtod `depositBonus` is repeated in loops in the implementations of `_TreePayment` and `depositBonusFordPNMBuy`.

Recommendation:

This kind of use case is mostly implemented in the following way in DeFi:
Instead of transferring the busd right away, add the amount of reward to be paid for the users in a mapping state variable like:

```
mapping(address => uint256) bonusToBePaid;
```

while the actual busd rewards (i.e., bonuses) are sent as a whole to a reward pool (or the same contract). Users later claim their rewards from the reward pool in a separate transaction triggered by them, like this:

```
bonusToBePaid[userAddress] -= amount;
rewardPool.safeTransferBUSDBonusTo(userAddress, amount);
```

Fix

Developers acknowledged the issue and they showed their awareness about it. They preferred accessibility for users (i.e. simplicity to deal with their ecosystem) than following the more common practice.

MEDIUM | RESOLVED

Exposure to revert due to underflow

dpnm_sc.sol - in `getMaxDailyBuy` this line:

```
buyLimit = buyLimit - uint(last24BuyAmount) + uint(soldLast48Hours);
```

Is exposed to a panic revert (leads to Denial of service) in some valid cases due to underflow.

Recommendation:

Just switch the ordering of last two quantities.

```
buyLimit = buyLimit + uint(soldLast48Hours) - uint(last24BuyAmount);
```

Fix

Issue fixed in `60761c1`.

MEDIUM | ACKNOWLEDGED

Centralization Risk

Admin enjoys too much authority in contracts. The general theme is that admin has power to call several state changing functions also affecting the tokenomics of the project. Some functions can be highly severe to be left out, controlled by one wallet.

Recommendation:

Apply governance methods or use multisig wallets.

Fix

Developers stated they are planning to utilize multisig wallets.

MEDIUM | RESOLVED

Unchecked return

dpnm_sc.sol - In `buyTurnoverWithGWT(uint)` & `buyEarnLimitWithGWT(uint)`, the return value from the burning operation of `gwt` is not validated `gwt.burn(msg.sender, gwtCost)`.
In `buydPNM(uint)` & `selldPNM(uint)`, the return value from the minting operation of `gwt` is not validated `gwt.mint(msg.sender, gwtCost)`.

Recommendation:
Wrap in require statement:
`require(gwt.burn(msg.sender, gwtCost), ...).`
`require(gwt.mint(msg.sender, gwtCost), ...).`

Fix
Issue fixed in `60761c1`.

LOW | RESOLVED

Pragma version lock

It's recommended to have the same compiler and flags that the contracts were tested with the most. This way, it reduces the risk of introducing unknown bugs by accidentally deploying with the latest version that might not be stable yet.

Recommendation:
Lock pragma versions

Fix
Resolved in 05ec745

Some public visibilities are unneeded

phenomanelTree.sol / dpnm_sc.sol / gwt_new.sol - functions declared as `public` while they are not used internally in the contracts.

```
function getUserData(address account) public view returns (uint, uint,
uint, uint, uint)
function getUserBuySellData(address account) public view returns (uint,
uint, uint, uint, uint)
function getLvlsLockStatus(address user) public view returns(bool[10]
memory)
function getLostProfit(address account) public view returns (uint[10]
memory)
function promoter() public view onlyOwner returns(address)
function changeLock() public onlyOwner()
function changeFeeCollector(address newCollector) public onlyOwner
function changePromoter(address newPromoter) public onlyOwner
function setDailyBuyLimit (uint amount) public onlyPromoter
function setGWTforActivation (uint amount) public onlyPromoter
function setDaysForTree(uint amount) public onlyPromoter
function setMaxDaysForTree(uint amount) public onlyPromoter
function setbuyFeeToLiquidity(uint amount) public onlyPromoter
function setsellFeeToLiquidity(uint amount) public onlyPromoter
function setgwtTransFeeLiquidity(uint amount) public onlyPromoter
function setturnoverForOneGWT(uint amount) public onlyPromoter
function setdPNMbuyTurnoverIncrease(uint amount) public onlyPromoter
function setgwtBuyIncrease(uint amount) public onlyPromoter
function setgwtSellIncrease(uint amount) public onlyPromoter
function setdPNMsellTurnoverIncrease(uint amount) public onlyPromoter
function setearnLimitDepositedPerc(uint amount) public onlyPromoter
```

```
function setStakingDailyProfit(uint amount, uint stakingID) public
onlyOwner
function addAllowedContract(uint allowedContract) public onlyOwner
function returnAllowedContract(uint index) public view onlyOwner returns
(address)
```

Recommendation:
Declare as `external`.

Fix
Developers fixed the access modifier of methods in `60761c1`, but still exist methods declared as public and not invoked internally:

- `dpnm_sc: changeFeeCollector, changePromoter, makeTreePayment, buydPNM, selldPNM, transfer, buyTurnoverWithGWT, buyEarnLimitWithGWT, changeLock, name, symbol, decimals, totalSupply, balanceOf, disablePrestartMode`
- `gwt_new: changeFeeCollector, returnAllowedContract`
- `phaneomanelTree: getTreeRefs, getUserDistance`

As of commit `05ec745`, dev team left this issue unaddressed.
The issue was resolved in commit 8e4592d

LOW | RESOLVED

allowedContracts is publicly viewable while it should not

phenomanelTree.sol - `allowedContracts` array is publicly available to be viewed, while in the same time `returnAllowedContract` which shows an element of the aforementioned array is only accessible by owner ! If `allowedContracts` should be a secret, then it should not be available to read for the public.

Recommendation:

To be consistent, we either have `returnAllowedContract` available for all callers with no `onlyOwner` modifier (i.e. no secret). Or limit the access of `allowedContracts` and make it `internal` to avoid revealing it.

Fix
`allowedContracts` array is replaced by `allowedContractsMap` mapping in commit `60761c1`. Therefore, the issue is no longer relevant and considered resolved.

LOW | RESOLVED

Unnecessary assignment of array in treeuserlevels

phenomanelTree.sol & dpnm_sc.sol - `treeuserlevels[newUser]` is actually an array of zeros by default.

```
treeuserlevels[newUser] = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
```

Similarly, in constructor

```
treeuserlevels[msg.sender] = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
```

Also in dpnm_sc:

```
treeUserLostProfits[msg.sender] = [0,0,0,0,0,0,0,0,0,0];
```

```
treeUserLostProfits[_userAddress] = [0,0,0,0,0,0,0,0,0,0];
```

Recommendation:
remove the all zero array initialization lines.

Post-audit.
Issue fixed in `60761c1`.

LOW | ACKNOWLEDGED

Dangerous nested loops

phenomanelTree.sol - In `findPositionSpot, while` loop is including a `for` loop (inside `calcTreeNetwork`). Nested loops in solidity are not recommended in blockchain.

Recommendation:
Rethink the data representation and use mappings whenever possible in order to avoid nested loops

From the client.
Issue acknowledged by dev team.

LOW | ACKNOWLEDGED

Data representing tree users, mismatch can take place

dpnm_sc.sol - in `init, msg.sender` is added to users of the tree in the contract dpnmMain, but there is no guarantee that the deployer `msg.sender` of dpnmMain is the root of the tree in `phenomanelTree`.

In regard to that is apparent from the context of the project that tree users should be matching in `dpnmMain` and `phenomanelTree` but in that mentioned simple scenario they can be mismatched.

Recommendation:
retrieve the root of the tree in `phenomanelTree` and make it be the first added user in the body of method `init()`.
better solution is to represent the address of the root in a new variable `treeRoot` not using `owner()` (PS. This is related to another highly severe issue mentioned in this draft)

```
users[phenomanelTree(_treeAddress).treeRoot()] = user
// Apply same idea on the rest of user related tree data
```

It will be better though to rely solely on tree Users from the phenomanelTree contract to have a single source of truth.

Fix
Acknowledged by dev team as they ensure that deployment goes the way it is supposed to be carried out in order to avoid this issue.

Interactions preceding the effects

dpnm_sc.sol - in `activate`, dealing with smart contracts requires care, that's why we are obliged to abide by certain patterns. One such pattern is checks-effects-interactions. Method `activate` does not follow checks-effects-interactions pattern, as it creates the user after interacting with the phenomanelTree.

```
contractTree.positionUser(newUser,referrerAddress,treeDepth);

createUser(newUser, referrerAddress);
```

Similarly, in the body of `buydPNM(...)`: we have interactions `busd.safeTransferFrom(...) and gwt.mint(...)` better be called after the effects. As for: `buyTurnoverWithGWT(...)` and `buyEarLimitWithGWT(...)` we have `gwt.burn` preceding state changes (i.e. effects):

```
//add turnover
users[msg.sender].totalTurnover += turnoverAmount;
```

And,

```
//increase earn limit
users[msg.sender].earnLimitLeft += earnlimitAmount;
users[msg.sender].totalEarnLimit += earnlimitAmount;
```

Recommendation:
Effects like `createUser(...)` which update the internal state of contract should precede interactions like `contractTree.positionUser(...)`.

From the client.
Issue fixed in `60761c1`.

LOW | RESOLVED

No uint value input validation applied on methods

dpnm_sc.sol - in `selldPNM` No assurance that the value of `BUSDamount` should be non-zero value. Also, it will be safer to assure that `dPNMprice` is non-zero. Also, same applies to:
`function buyEarnLimitWithGWT(uint earnlimitAmount) public onlyActivated notPrestart`
`function buyTurnoverWithGWT(uint turnoverAmount) public onlyActivated notPrestart`

Recommendation:

- add a validation statement `require(BUSDamount > 0 && dPNMprice >0, ...)`
- add a validation statement `require(earnlimitAmount > 0, ...)`
- add a validation statement `require(turnoverAmount > 0, ...)`

Fix
Issue fixed in `60761c1`.

LOW | RESOLVED

Wasteful array representation for variable

dpnm_sc.sol - `treeuserlevelslock` is represented as a mapping of an array. The array represents the last level reached by the `userAddress` that maps to that array. The pattern that it takes is in the form of a stream of trues followed by a stream of false till the end of the array (e.g., `[true, true, true, false, false]`). So the only information needed is to know the index of the last true right before the first false. This shall save some computation and space as there will be no need to use loops in order to process it.

Recommendation:
make it `mapping(address => uint8) treeuserlevelslock` instead of an array. `uint8` represents the index of the level reached.

Fix
Issue fixed in `60761c1` by introducing mapping variable `firstLockedLvl` to replace `treeuserlevelslock`.

INFORMATIONAL | RESOLVED

Variable to be declared as constant

dpnm_sc.sol - `minDailyBuy` is shown to be immutable, therefore it is better to be declared as a constant.

Recommendation:

```
uint constant public minDailyBuy = 50e18;
```

INFORMATIONAL | RESOLVED

Misleading variable name

dpnm_sc.sol - name choice of `minDailyBuy` refers to something completely different to what it actually is. Since it should be referring to the default lower bound of `maxDailyBuy` not the minimum amount to buy daily

Recommendation:

Rename it to something like: `lowerBoundMaxDailyBuy, defaultMaxDailyBuy,` ... etc.

Fix

Issue fixed in `60761c1`.

INFORMATIONAL | RESOLVED

No validation on input address

dpnm_sc.sol - input address in `init, changePromoter` & `changeFeeCollector` might be zero and assignment of such improper address would be set in this case.
gwt_new.sol - input address in init, `addAllowedContract` & `changeFeeCollector` might be zero and assignment of such improper address would be set in this case.

Recommendation:

add a require statement to ensure the input addresses are non-zero addresses.

Fix

Issue fixed in `60761c1`.

INFORMATIONAL | RESOLVED

Code logic repeated can be replaced by function

dpnm_sc.sol - in init(...), the logic of creating a new user is already implemented in private function createUser(address,address).

Recommendation:
use :
`createUser(treeRoot, address(0));`

instead of the bulky logic of user creation inside the `init(...)` method.

Fix
Issue fixed in `60761c1`.

INFORMATIONAL | ACKNOWLEDGE

Replace a frequently used number by a constant

phenomanelTree.sol & dpnm_sc.sol - depth of tree, which is assigned to be 15. This number is repeated throughout the contract, and it's not considered a good coding pattern. Similarly, the same is occurring in dpnm_sc.sol in `treeDepth` is declared to represent the depth of tree (i.e. 10 in this case) but it is almost never used where it should be used.

Recommendation:
Declare a constant that represents depth of tree in `phenomanelTree (i.e. TREE_DEPTH=15)` and use the constant instead. Same with `dpnmMain`.

Fix
Acknowledged by dev team.

INFORMATIONAL | RESOLVED

Unused variable

phenomanelTree.sol - In `positionUser`, variable `lvlDistance` is unused.

Recommendation:
Put the variable in a require statement to check if it is within the expected bound that should be returned from the method.

Fix
Issue fixed in `60761c1`.

INFORMATIONAL | RESOLVED

Gas Optimization.

dpnm_sc.sol - In the `depositBonus`
Values of the *treeuserlevelslock* mapping are all booleans. So there is no need to compare with true or false. It will cause more gas consumption.

```
if(treeuserlevelslock[userAddress[7] == true){
```

Recommendation:
remove the true.
`if(treeuserlevelslock[userAddress[7]){`

Fix
Issue fixed in `60761c1`

INFORMATIONAL | RESOLVED

Wrong Function description.

dpnm_sc.sol - In the `depositBonus`
For dPNM bonus requires that token value less than earn limit left.
But in the description, dev wrote that token value should be greater than earn limit left.

```
* For dPNM bonus require that token value greater then earn limit left
```

Recommendation:
Change `greater` to `less`.\

Fix
Issue fixed in 60761c1

	pnm_sc.sol gwt_new.sol phenomenalTree.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security

As a part of our work assisting DPNMDEFI in verifying the correctness of their contracts code, our team was responsible for writing integration tests using the Hardhat testing framework.

The tests were based on the functionality of the code, as well as a review of the DPNMDEFI contracts requirements for details about issuance amounts and how the system handles these.

dpnmMain

- ✓ Should be able to deploy correctly (63ms)
- ✓ SHould be able to get promoter (103ms)
- ✓ SHould be able to change promoter (131ms)
- ✓ Should be able to get tree active until (64ms)
- ✓ Should be able to get user buy sell data
- ✓ Should be able to get lvls lock status (42ms)
- ✓ Should be able to get lost profit (38ms)
- ✓ Should be able to get user data (62ms)
- ✓ SHould allow users to be activated (675ms)
- ✓ Should check if user is qualified for bonus
- ✓ Should not allow transfers
- ✓ SHould be able to set daily buy limit (147ms)
- ✓ Should be able to buy dpnm (611ms)
- ✓ Should be able to buy dpnm (448ms)
- ✓ Should be able to sell dpnm (707ms)
- ✓ Should be able to sell dpnm with turn over increase (439ms)
- ✓ Should be able to buy turn over with gwt (291ms)
- ✓ Should be able to buy earn limit with gwt (617ms)
- ✓ Should be able to set gwt for activation (166ms)
- ✓ Should be able to set days for tree (164ms)
- ✓ Should be able to set max days for tree (162ms)
- ✓ Should be able to set buy fee to liquidity (152ms)
- ✓ Should be able to set sell fee to liquidity (156ms)
- ✓ Should be able to set gwt trans fee liquidity (180ms)
- ✓ Should be able to set turn over for one gwt (169ms)
- ✓ Should be able to set dpnm buy turn over increase (179ms)
- ✓ Should be able to set gwt buy increase (182ms)
- ✓ Should be able to set gwt sell increase (175ms)
- ✓ Should be able to set dpnm sell turn over increase (173ms)

✓ Should be able to set earn limit deposited perc (171ms)
✓ Should be able to change lock (85ms)
✓ Should be able to get name
✓ Should be able to get symbol (43ms)
✓ Should be able to get decimals
✓ Should be able to change fee collector (124ms)
✓ Should be able to disable prestart mode (182ms)
✓ Should be able to get total supply (115ms)
✓ Should be able to get balance of (122ms)
✓ Should be able to make tree payment (897ms)
✓ Should be able to get max daily buy (286ms)

GwtNew

✓ Should be able to deploy correctly (74ms)
✓ Should be able to init (164ms)
✓ Should be able to add to allowed contract (166ms)
✓ Should be able to mint tokens (152ms)
✓ Should be able to burn tokens (179ms)
✓ Should be able to change staking enabled (87ms)
✓ Should be able to stake gwt (521ms)
✓ SHoud be able to get user pool slots (275ms)
✓ Should be able to claim staking (317ms)
✓ Should be able to get user pool data (249ms)
✓ Should allow transfers to be made (260ms)
✓ Should set gwt trans fee liquidity (185ms)
✓ Should be able to set staking daily profit (206ms)
✓ Should be allowed to change fee collector (139ms)

phenomenalTree

✓ Should be able to add allowed contracts (130ms)
✓ Should be able to return allowed contracts (150ms)
✓ Should be able to position user (539ms)
✓ Should be able to position user with different branch (268ms)
✓ Should be able to find position spot through different tree branches (253ms)
✓ Should be able to calculate tree network (219ms)
✓ Should be able to find position spot (192ms)
✓ Should be able to get user distance (404ms)
✓ Should be to get lvls up (436ms)
✓ SHould be able to get tree refs (62ms)

64 passing (1m)

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	% UNCOVERED LINES
dpnm_sc.sol	95.27	88.83	100	95.49	... 449, 606, 720
gwt_new.sol	100	95.83	100	97.44	264, 266
phenomenalTree.sol	96.55	92.31	100	95.45	138,142,269,271
FILE	**96.28**	**90.52**	**100**	**95.83**	

We are grateful for the opportunity to work with the DPNMDEFI team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the DPNMDEFI team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo