Neo Tokyo contest
Findings & Analysis Report

2023-04-11

Table of contents

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit contest outlined in this document, C4 conducted an analysis of the Neo Tokyo smart contract system written in Solidity. The audit contest took place between March 8—March 15 2023.

Wardens

136 Wardens contributed reports to the Neo Tokyo contest:

1. 0x1337
2. 0x1f8b

3. 0x52
4. 0x6980
5. 0xAgro
6. 0xSmartContract
7. 0xSolus
8. 0xWeiss
9. 0xhacksmithh
10. 0xkazim
11. 0xnev
12. ABA
13. Angry_Mustache_Man
14. Aymen0909
15. BPZ (pa6221, Bitcoinfever244, and PrasadLak)
16. Bjorn_bug
17. BowTiedOriole
18. ChainReview
19. DadeKuma
20. DeFiHackLabs (SunSec, gbaleee, 0x4non, Aits, and Rappie)
21. Deathstore
22. DevABDee
23. Diana
24. Dravee
25. Dug
26. Englave
27. Flora
28. Go-Langer
29. Haipls
30. IceBear
31. Inspex (Resistor, jokopoppo, DeStinE21, mimic_f, Rugsurely, and ErbaZZ)

32. J4de
33. JCN
34. Jeiwan
35. Josiah
36. Kek
37. Krace
38. Kresh
39. LegendFenGuin
40. Lirios
41. MadWookie
42. Madalad
43. MatricksDeCoder
44. MiniGlome
45. MyFDsYours
46. R-Nemes
47. RaymondFam
48. ReyAdmirado
49. Rolezn
50. Ruhum
51. SAAJ
52. Sathish9098
53. Shubham
54. Taloner
55. Toshii
56. UdarTeam (ahmedov and tourist)
57. Udsen
58. Viktor_Cortess
59. aashar
60. adriro

61. ak1
62. anodaram
63. arialblack14
64. ast3ros
65. atharvasama
66. auditor0517
67. ayden
68. brgltd
69. btk
70. c3phas
71. carlitox477
72. catellatech
73. cccz
74. chaduke
75. codeislight
76. ctrlc03
77. deadrxsezzz
78. descharre
79. dharma09
80. durianSausage
81. erictee
82. fatherOfBlocks
83. favelanky
84. ginlee
85. glcanvas
86. handsomegiraffe
87. hunter_w3b
88. jasonxiale
89. jekapi

90. joestakey
91. juancito
92. kaden
93. kenzo
94. kutugu
95. lemonr
96. leopoldjoy
97. luxartvinsec
98. martin
99. matrix_Owl
100. minhquanym
101. mrpathfindr
102. nadin
103. navinavu
104. nobody2018
105. oyc_109
106. parsely
107. peanuts
108. pfedprog
109. pipoca
110. rbserver
111. rokso
112. saian
113. santipu_
114. schrodinger
115. scokaf (Scoon and jauvany)
116. sinarette
117. slvDev
118. tsvetanovv

119. ubl4nk
120. ulqiorra
121. volodya
122. yamapyblack
123. zaskoh

This contest was judged by hansfriese.

Final report assembled by liveactionllama.

Summary

The C4 analysis yielded an aggregated total of 2 unique vulnerabilities. Of these vulnerabilities, 2 received a risk rating in the category of HIGH severity and 0 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 83 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 46 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the C4 Neo Tokyo contest repository, and is composed of 2 smart contracts written in the Solidity programming language and includes 969 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities based on three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling

- Escalation of privileges
- Arithmetic
- Gas use

For more information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on [the C4 website](), specifically our section on [Severity Categorization]().

High Risk Findings (2)

[H-01] Updating a pool's total points doesn't affect existing stake positions for rewards calculation

Submitted by [adriro](), *also found by* [kutugu](), [joestakey](), [Madalad](), [rbserver](), [minhquanym](), [minhquanym](), [auditor0517](), [sinarette](), [ast3ros](), [ABA](), [Haipls](), [J4de](), *and* [Dug]()

Staking rewards are calculated based on the user's share of total points in the corresponding asset pool, this is the sum of the points associated to the staker's positions divided by the total points from all positions in the pool. We can see this calculation in the `getPoolReward` function:

[NeoTokyoStaker.sol#L1386-L1393]()

```
// Return final shares.
unchecked {
    uint256 share = points * _PRECISION / pool.totalPoints * tot
    uint256 daoShare = share * pool.daoTax / (100 * _DIVISOR);
    share /= _PRECISION;
    daoShare /= _PRECISION;
    return ((share - daoShare), daoShare);
}
```

However, note that `pool.totalPoints` is the current value of the pool's total point at the time the function `getPoolReward` is called. It isn't related to the time the user staked their position, or isn't affected in any way by other stake/unstake actions from potentially other users.

This means that any action that modifies the pool's total points (stake or unstake) won't affect current staking positions, as previously opened staking positions won't accrue their rewards correctly. For stake actions, it will cause rewards from existing staking positions to be reduced, as their calculation of the shares now divided by a higher `pool.totalPoints` value. From unstake actions, it will cause rewards from existing staking positions to be incorrectly increased, as the calculation of the shares is now divided by a lower `pool.totalPoints` value. See section "Proof of Concept" for a more detailed walkthrough.

In a similar way, this could also be used by a griefer to intentionally harm another user. As the `getReward` function present in the `BYTES2` contract is permissionless (anyone can call this on behalf of an arbitrary account), a bad actor can call this when the pool's total points is high, which will have the effect of reducing the user rewards.

Proof of Concept

Let's assume the pool is empty. Alice stakes at `t1` an asset worth 100 points and Bob stakes at `t2` another asset worth 100 points. In order to simplify the examples, let's also consider that all periods fall in the same window, thus having a constant reward rate.

Alice claims after Bob stakes

In this scenario, Alice claims her rewards in `t3` after Bob stakes. She will get less rewards from the `[t1, t2]` period, as the calculation will consider the entire period `[t1, t3]` and calculate the shares using 200 points. Here the correct way would be to calculate the period `[t1, t2]` using 100 total points, and the period `[t2, t3]` using 100 total points.

1. Alice stakes at `t1` and gets 100 points. Total points is 100.
2. Bob stakes at `t2` and gets 100 points. Total points is 200.
3. Alice claims rewards at `t3`. She will get less rewards since the calculation will be done using 200 points.

Alice and Bob stake at same time

Here, `t1 == t2` and Bob and Alice stake at the same time. Alice unstakes at `t3` and Bob claims rewards at `t4`. In this case, Bob will get more rewards, as the calculation will consider the entire period `[t1, t4]` and calculate the shares using 100 points. Here the correct way would be to calculate the period `[t1, t3]` using 200 total points, and the period `[t3, t4]` using 100 total points.

1. Alice and Bob stake at `t1 == t2` and each one gets 100 points. Total points is 200.
2. Alice unstakes at `t3`. Total points is 100.
3. Bob claims rewards at `t4`. He will get more rewards since the calculation will be done using 100 points.

Griefer intentionally claims rewards of Alice

As described in the previous section, a bad actor can intentionally claim the rewards of another user at a time the pool has a high value for total points, since this call as this is a permissionless action.

1. Alice stakes at `t1` and gets 100 points. Total points is 100.
2. Bob stakes at `t2` and gets 100 points. Total points is 200.
3. Bad actor claims rewards of Alice at `t3`. She will get less rewards since the calculation will be done using 200 points.

Recommendation

Rewards calculation should track reward rate according to modifications in the pool's total points caused by stake or unstake actions.

My recommendation for a performant solution would be to follow [this staking example](#) or [the full Staking contract from Synthetix](#). The principal idea here is that every action that affects rewards triggers the `updateReward` modifier, which updates the `rewardPerTokenStored` variable that tracks the reward amount per staked token. A similar idea could be adapted to track the reward per point for the current contract. Stake and unstake actions should update this variable before modifying the pool's total points.

[TimTinkers (Neo Tokyo) confirmed and commented](#):

> Excellent catch, good write-up.

[H-02] Underflow of `lpPosition.points` during withdrawLP causes huge reward minting

Submitted by rokso, *also found by* 0xnev, BPZ, Toshii, Josiah, ak1, ulqiorra, rbserver, juancito, juancito, minhquanym, auditor0517, auditor0517, Jeiwan, cccz, RaymondFam, rokso, MadWookie, kaden, 0x52, carlitox477, DadeKuma, Krace, Haipls, Krace, UdarTeam, kenzo, DadeKuma, ABA, jekapi, J4de, anodaram, LegendFenGuin, aashar, nobody2018, Ruhum, *and* Lirios

NeoTokyoStaking allows to stake and withdraw LPs. User can stake multiple times on same position which simply results in extended lock time and user can withdraw all of these LPs once lock time is passed.

There is a scenario when withdrawing LPs results in overflow of lpPosition.points. After withdraw if attacker calls `getRewards()` then attacker will get more than 1e64 BYTES tokens as reward.

Proof of Concept

Affected code block: NeoTokyoStaker.sol#L1622-L1631

Affected line: L1627

From below POC, you can see that Alice is staking twice and some specific amounts which will trigger underflow when Alice withdraw LP. Once staked LPs are unlocked, Alice can withdraw her LPs and call `getReward()` to trigger minting of more than 1e64 BYTES tokens.

Below test can be added in `NeoTokyoStaker.test.js` test file.

```
it('Unexpected rewards minting due to underflow
        // Configure the LP token contract addre
        await NTStaking.connect(owner.signer).cc
        const amount1 = ethers.utils.parseEther(
        const amount2 = ethers.utils.parseEther(
        const lockingDays = 30
```

```
// Alice stake amount1 LPs for 30 days.
await NTStaking.connect(alice.signer).st
        ASSETS.LP.id,
        TIMELOCK_OPTION_IDS[lockingDays]
        amount1,
        0,
        0
);

// Alice stake amount2 LPs for 30 days.
await NTStaking.connect(alice.signer).st
        ASSETS.LP.id,
        TIMELOCK_OPTION_IDS[lockingDays]
        amount2,
        0,
        0
);

const priorBlockNumber = await ethers.pr
const priorBlock = await ethers.provider
let aliceStakeTime = priorBlock.timestan

// Bob stake 10 LPs for 30 days
await NTStaking.connect(bob.signer).stak
        ASSETS.LP.id,
        TIMELOCK_OPTION_IDS[lockingDays]
        ethers.utils.parseEther('10'),
        0,
        0
);

// Set time to unlock staked lp
await ethers.provider.send('evm_setNextB
        aliceStakeTime + (60 * 60 * 24 *
]);

// Alice withdraw LP
// This transaction will cause underflow
await NTStaking.connect(alice.signer).wi
        ASSETS.LP.id,
        amount1.add(amount2)
);

// Before exploit:: Verify Alice's Bytes
expect(await NTBytes2_0.balanceOf(alice.
```

```
        // Get rewards for Alice. It will mint F
        await NTBytes2_0.getReward(alice.address

        // After exploit:: Verify Alice's Bytes
        expect(await NTBytes2_0.balanceOf(alice.
    });
```

Recommended Mitigation Steps

Consider adding proper precision for `points` and `totalPoints` and also consider checking for under/overflows.

[TimTinkers (Neo Tokyo) commented](#):

> @hansfriese - this attack is a different way of abusing the same rounding bug from **#348**; duplicates?

> I agree with the severity of the underlying issue and really appreciate the test case demonstrating this.

[hansfriese (judge) commented](#):

> Totally, there are 3 kinds of rounding issues.

1. Users can get infinite points by depositing `5e15` twice and withdrawing `1e16`. So `0 * 2 - 1 = -1 = type(uint256).max`
2. Users can get free points by depositing 1e16 and withdrawing 5e15 twice. So `1 - 0 * 2 = 1`
3. Users would lose some LP(or staking reward) due to the rounding.

> After discussing with other judges, I will merge 1 and 2 into one high and mark 3 as QA as it contains a lower impact.

[TimTinkers (Neo Tokyo) confirmed](#)

Low Risk and Non-Critical Issues

For this contest, 76 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by **0xSmartContract** received the top score from the judge.

The following wardens also submitted reports: joestakey, Udsen, Kresh, atharvasama, Deathstore, favelanky, 0x1f8b, ulqiorra, carlitox477, zaskoh, rbserver, saian, 0xkazim, BPZ, btk, Go-Langer, nadin, slvDev, tsvetanovv, rokso, minhquanym, Diana, peanuts, 0x6980, Viktor_Cortess, MyFDsYours, codeislight, Jeiwan, martin, erictee, SAAJ, DadeKuma, BowTiedOriole, glcanvas, handsomegiraffe, 0xSolus, Kek, IceBear, ayden, luxartvinsec, brgltd, Haipls, DevABDee, Dravee, 0xAgro, catellatech, descharre, DeFiHackLabs, Inspex, RaymondFam, ABA, Madalad, jekapi, ChainReview, santipu_, yamapyblack, MatricksDeCoder, oyc_109, matrix_0wl, scokaf, Englave, jasonxiale, ubl4nk, Dug, pfedprog, chaduke, Sathish9098, fatherOfBlocks, mrpathfindr, parsely, Rolezn, lemonr, deadrxsezzz, Taloner, *and* 0xhacksmithh.

Summary

Low Risk Issues List

Number	Issues Details	Context
[L-01]	Low-level calls that are unnecessary for the system should be avoided	2
[L-02]	Insufficient coverage	
[L-03]	Project Upgrade and Stop Scenario should be	
[L-04]	Division before multiplication causing significant loss of precision	1
[L-05]	Update codes to avoid Compile Errors	3
[L-06]	`Claim` event is missing parameters	1
[L-07]	Project has NPM Dependency which uses a vulnerable version : `@openzeppelin`	1
[L-08]	Keccak Constant values should used to immutable rather than constant	7
[L-09]	In the `constructor` , there is no return of incorrect address identification	1

Total 10 issues

Non-Critical Issues List

Number	Issues Details	Context
[N-01]	Omissions in Events	1
[N-02]	`Function writing` that does not comply with the `Solidity Style Guide`	
[N-03]	Tokens accidentally sent to the contract cannot be recovered	All Contracts
[N-04]	Floating pragma	2
[N-05]	Use SMTChecker	
[N-06]	Constants on the left are better	4
[N-07]	`constants` should be defined rather than using magic numbers	3
[N-08]	Use the `delete` keyword instead of assigning a value of `0`	8
[N-09]	Use a single file for all system-wide constants	12
[N-10]	According to the syntax rules, use `=> mapping (` instead of `=> mapping(` using spaces as keyword	2
[N-11]	For modern and more readable code; update import usages	9
[N-12]	Assembly Codes Specific — Should Have Comments	7
[N-13]	Take advantage of Custom Error's return value property	3
[N-14]	Pragma version^0.8.19 version too recent to be trusted	1
[N-15]	Inconsistent Solidity Versions	21

Total 16 issues

Suggestions

Number	Suggestion Details
[S-01]	You can explain the operation of critical functions in NatSpec with an infographic

Total 1 suggestion

[L-01] Low-level calls that are unnecessary for the system should be avoided

Low-level calls that are unnecessary for the system should be avoided whenever possible because low-level calls behave differently from a contract-type call. For example;

`address.call(abi.encodeWithSelector("fancy(bytes32)", mybytes))`` does not verify that a target is actually a contract, while `ContractInterface(address).fancy(mybytes)` does.

Additionally, when calling out to functions declared view/pure, the solidity compiler would actually perform a staticcall providing additional security guarantees while a low-level call does not. Similarly, return values have to be decoded manually when performing low-level calls.

Note: if a low-level call needs to be performed, consider relying on Contract.function.selector instead of encoding using a hardcoded ABI string.

```
2 results

contracts/staking/NeoTokyoStaker.sol:
  772:          (bool success, bytes memory data) =
  773:                  _asset.call(
  774:                          abi.encodeWithSelector(
  775:                                  _TRANSFER_FROM_SELECTOR,
  776:                                  _from,
  777:                                  _to,
  778:                                  _idOrAmount
  779:                          )
  780:                  );
  781:
  782:          // Revert if the low-level call fails.
  783:          if (!success) {
  784:                  revert(string(data));
  785:          }
```

```
  801:          (bool success, bytes memory data) =
```

```
                _asset.call(
                        abi.encodeWithSelector(
                                _TRANSFER_SELECTOR,
                                _to,
                                _amount
                        )
                );

        // Revert if the low-level call fails.
        if (!success) {
                revert(string(data));
        }
}
```

Recommendation

When calling out to a contract known in the system, always prefer typed contract calls (interfaces/contract) instead of low-level calls.

This is to avoid errors, potentially unchecked return values, have security guarantees.

[L-02] Insufficient coverage

Description

The test coverage rate of the project is ~85%. Testing all functions is best practice in terms of security criteria.

Due to its capacity, test coverage is expected to be 100%.

File	% Stmts	% Branch	% Funcs	%
staking/	95	74.53	88.57	
BYTES2.sol	78.95	83.33	75	
NeoTokyoStaker.sol	96.09	74	92.59	

[L-03] Project Upgrade and Stop Scenario should be

At the start of the project, the system may need to be stopped or upgraded, I suggest you have a script beforehand and add it to the documentation. This can also be called an " EMERGENCY STOP (CIRCUIT BREAKER) PATTERN ".

https://github.com/maxwoe/solidity_patterns/blob/master/security/EmergencyStop.sol

[L-04] Division before multiplication causing significant loss of precision

First divides and then multiplies again, there is a significant loss of precision;

```
contracts/staking/NeoTokyoStaker.sol:
  1385
  1386:                         // Return final shares.
  1387:                         unchecked {
  1388:                                 uint256 share = points *
  1389:                                 uint256 daoShare = share
  1390:                                 share /= _PRECISION;
  1391:                                 daoShare /= _PRECISION;
  1392:                                 return ((share - daoShar
  1393:                         }
  1394                  }
```

Recommended Mitigation Steps

Multiply first before dividing to keep the precision.

[L-05] Update codes to avoid Compile Errors

```
Warning: Function state mutability can be restricted to pure
   --> contracts/staking/BYTES2.sol:245:2:
    |
245 |   function updateReward (
    |   ^ (Relevant source part starts here and spans across mul


Warning: Function state mutability can be restricted to pure
   --> contracts/staking/BYTES2.sol:261:2:
```

```
    |
261 |   function updateRewardOnMint (
    |   ^ (Relevant source part starts here and spans across mul

Warning: Contract code size is 63151 bytes and exceeds 24576 byt
   --> contracts/staking/NeoTokyoStaker.sol:190:1:
    |
190 | contract NeoTokyoStaker is PermitControl, ReentrancyGuard
    | ^ (Relevant source part starts here and spans across multi
```

[L-06] `Claim` event is missing parameters

The `NeoTokyoStaker.sol` contract has very important function; `Claim`

However, only amounts are published in emits, whereas msg.sender must be specified in every transaction, a contract or web page listening to events cannot react to users, emit does not serve the purpose.

Basically, this event cannot be used

```
contracts/staking/NeoTokyoStaker.sol:
  1444
  1445:                 // Emit an event.
  1446:                 emit Claim (
+                       msg.sender
  1447:                         _recipient,
  1448:                         totalReward,
  1449:                         totalTax
  1450:                 );
```

Recommended Mitigation Steps

Add `msg.sender` parameter in event-emit

[L-07] Project has NPM Dependency which uses a vulnerable version : `@openzeppelin`

```
1 result - 1 file
```

```
package.json:
  1: {
  9:     "@openzeppelin/contracts-upgradeable": "^4.4.2",
```

```
VULNERABILITY   VULNERABLE VERSION
M
Incorrect Calculation
>=4.8.0 <4.8.2
M
Incorrect Calculation
>=4.8.0 <4.8.2
H
Improper Verification of Cryptographic Signature
<4.7.3
M
Denial of Service (DoS)
>=3.2.0 <4.7.2
L
Incorrect Resource Transfer Between Spheres
>=4.6.0 <4.7.2
H
Incorrect Calculation
>=4.3.0 <4.7.2
H
Information Exposure
>=4.1.0 <4.7.1
H
Information Exposure
>=4.0.0 <4.7.1
```

Proof Of Concept

https://security.snyk.io/package/npm/@openzeppelin%2Fcontracts-upgradeable

Recommended Mitigation Steps

Upgrade OZ to version 4.8.0 or higher

[L-08] Keccak Constant values should used to immutable rather than constant

There is a difference between constant variables and immutable variables, and they should each be used in their appropriate contexts.

While it doesn't save any gas because the compiler knows that developers often make this mistake, it's still best to use the right tool for the task at hand.

```
7 results - 2 files

contracts/staking/BYTES2.sol:
  36    /// The identifier for the right to perform token burns.
  37:   bytes32 public constant BURN = keccak256("BURN");
  38
  39    /// The identifier for the right to perform some contrac
  40:   bytes32 public constant ADMIN = keccak256("ADMIN");
  41

contracts/staking/NeoTokyoStaker.sol:
  205   /// The identifier for the right to configure the LP tok
  206:  bytes32 public constant CONFIGURE_LP = keccak256("CONFIG
  207
  208   /// The identifier for the right to configure timelock c
  209:  bytes32 public constant CONFIGURE_TIMELOCKS = keccak256(

  213   /// The identifier for the right to configure Identity a
  214:  bytes32 public constant CONFIGURE_CREDITS = keccak256("C
  215
  216   /// The identifier for the right to configure emission r
  217:  bytes32 public constant CONFIGURE_POOLS = keccak256("CON
  218
  219   /// The identifier for the right to configure BYTES stak
  220:  bytes32 public constant CONFIGURE_CAPS = keccak256("CONF
```

[L-09] In the `constructor`, there is no return of incorrect address identification

In case of incorrect address definition in the `constructor`, there is no way to fix it because of the variables are immutable.

It is recommended to fix the architecture:

1. Address definitions can be done changeable architecture
2. Because of `owner = address(0)` at the end of the constructor, there is no way to fix it, so the owner's authority can be maintained.

```
contracts/staking/BYTES2.sol:
  74    */
  75:   constructor (
  76:           address _bytes,
  77:           address _s1Citizen,
  78:           address _staker,
  79:           address _treasury
  80:   ) {
  81:           BYTES1 = _bytes;
  82:           S1_CITIZEN = _s1Citizen;
  83:           STAKER = _staker;
  84:           TREASURY = _treasury;
  85:   }
```

[N-01] Omissions in Events

Throughout the codebase, events are generally emitted when sensitive changes are made to the contracts. However, some events are missing important parameters.

```
contracts/staking/NeoTokyoStaker.sol:
  1043          */
  1044:         function _stakeBytes (
  1045:                 uint256
  1046:         ) private {
  1047:                 uint256 amount;
  1048:                 uint256 citizenId;
  1049:                 uint256 seasonId;
  1050:                 assembly{
  1051:                         amount := calldataload(0x44)
  1052:                         citizenId := calldataload(0x64)
  1053:                         seasonId := calldataload(0x84)
  1054:                 }
  1055:
  1056:                 // Attempt to transfer BYTES to escrow.
  1057:                 _assetTransferFrom(BYTES, msg.sender, ac
```

```
                // Handle staking BYTES into an S1 Citiz
                if (seasonId == 1) {
                        StakedS1Citizen storage citizenS
                        uint256 cap = VAULT_CAP;
                        if (!citizenStatus.hasVault) {
                                cap = NO_VAULT_CAP;
                        }
                        if (citizenStatus.stakedBytes +
                                revert AmountExceedsCap(
                        }

                        // Validate that the caller actu
                        if (citizenStatus.timelockEndTim
                                revert CannotStakeIntoUr
                        }

                        PoolData storage pool = _pools[A
                        unchecked {
                                uint256 bonusPoints = (a
                                citizenStatus.stakedByte
                                citizenStatus.points +=
                                pool.totalPoints += bonu
                        }

                // Handle staking BYTES into an S2 Citiz
                } else if (seasonId == 2) {
                        StakedS2Citizen storage citizenS
                        uint256 cap = NO_VAULT_CAP;
                        if (citizenStatus.stakedBytes +
                                revert AmountExceedsCap(
                        }

                        // Validate that the caller actu
                        if (citizenStatus.timelockEndTim
                                revert CannotStakeIntoUr
                        }

                        PoolData storage pool = _pools[A
                        unchecked {
                                uint256 bonusPoints = (a
                                citizenStatus.stakedByte
                                citizenStatus.points +=
                                pool.totalPoints += bonu
                        }

```

```
                // Revert because an invalid season ID h
                } else {
                        revert InvalidSeasonId(seasonId)
                }

                // Emit an event.
                emit Stake(
+               citizenId
                        msg.sender,
                        BYTES,
                        (seasonId << 128) + citizenId,
                        amount
                );
        }
```

[N-02] `Function writing` that does not comply with the `Solidity Style Guide`

Context

All Contracts

Description

Order of Functions; ordering helps readers identify which functions they can call and to find the constructor and fallback definitions easier. But there are contracts in the project that do not comply with this.

https://docs.soliditylang.org/en/v0.8.17/style-guide.html

Functions should be grouped according to their visibility and ordered:

- constructor
- receive function (if exists)
- fallback function (if exists)
- external
- public
- internal

- private
- within a grouping, place the view and pure functions last

[N-03] Tokens accidentally sent to the contract cannot be recovered

Context

contracts/staking/NeoTokyoStaker.sol:

It can't be recovered if the tokens accidentally arrive at the contract address, which has happened to many popular projects, so I recommend adding a recovery code to your critical contracts.

Recommended Mitigation Steps

Add this code:

```
  /**
   * @notice Sends ERC20 tokens trapped in contract to external a
   * @dev Onlyowner is allowed to make this function call
   * @param account is the receiving address
   * @param externalToken is the token being sent
   * @param amount is the quantity being sent
   * @return boolean value indicating whether the operation succe
   *
  */
   function rescueERC20(address account, address externalToken, u
     IERC20(externalToken).transfer(account, amount);
     return true;
   }
}
```

[N-04] 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.
https://swcregistry.io/docs/SWC-103

Floating Pragma List:

```
2 results - 2 files

contracts/staking/BYTES2.sol:
  1  // SPDX-License-Identifier: AGPL-3.0-only
  2: pragma solidity ^0.8.19;
  3

contracts/staking/NeoTokyoStaker.sol:
  1  // SPDX-License-Identifier: AGPL-3.0-only
  2: pragma solidity ^0.8.19;
  3
```

Recommendation

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

[N-05] Use SMTChecker

The *highest* tier of smart contract behavior assurance is formal mathematical verification. All assertions that are made are guaranteed to be true across all inputs → The quality of your asserts is the quality of your verification.

https://twitter.com/0xOwenThurm/status/1614359896350425088?t=dbG9gHFigBX85Rv29lOjlQ&s=19

[N-06] Constants on the left are better

If you use the constant first you support structures that veil programming errors. And one should only produce code either to add functionality, fix an programming error or trying to support structures to avoid programming errors (like design patterns).

https://www.moserware.com/2008/01/constants-on-left-are-better-but-this.html

```
4 results - 1 file

contracts/staking/NeoTokyoStaker.sol:
   910:                 if (citizenVaultId != 0 && vaultId != 0)
   917:                 } else if (citizenVaultId != 0 && vaultI
   926:                 } else if (citizenVaultId == 0 && vaultI
  1834:                                 if (j != 0 && _inputs[i]
```

[N-07] `constants` should be defined rather than using magic numbers

A magic number is a numeric literal that is used in the code without any explanation of its meaning. The use of magic numbers makes programs less readable and hence more difficult to maintain and update.

Even assembly can benefit from using readable constants instead of hex/numeric literals.

```
contracts/staking/BYTES2.sol:
  149           */
  150:          uint256 treasuryShare;
  151:          unchecked {
  152:                  treasuryShare = _amount * 2 / 3;
  153:          }
  154:          _mint(TREASURY, treasuryShare);
  155   }

contracts/staking/NeoTokyoStaker.sol:
  1076                          unchecked {
  1077:                                 uint256 bonusPoints = (a

  1097                          unchecked {
  1098:                                 uint256 bonusPoints = (a
```

[N-08] Use the `delete` keyword instead of assigning a value of `0`

Using the 'delete' keyword instead of assigning a '0' value is a detailed optimization that increases code readability and audit quality, and clearly indicates the intent.

Other hand, if use `delete` instead `0` value assign , it will be gas saved.

```
8 results - 1 file

contracts/staking/NeoTokyoStaker.sol:
  1517:                 stakedCitizen.stakedBytes = 0;
  1518:                 stakedCitizen.timelockEndTime = 0;
  1519:                 stakedCitizen.points = 0;
  1521:                 stakedCitizen.stakedVaultId = 0;
  1582:                 stakedCitizen.stakedBytes = 0;
  1583:                 stakedCitizen.timelockEndTime = 0;
  1584:                 stakedCitizen.points = 0;
  1635:                         lpPosition.multiplier = 0;
```

[N-09] Use a single file for all system-wide constants

There are many addresses and constants used in the system. It is recommended to put the most used ones in one file (for example constants.sol, use inheritance to access these values).

This will help with readability and easier maintenance for future changes. This also helps with any issues, as some of these hard-coded values are admin addresses.

constants.sol
Use and import this file in contracts that require access to these values. This is just a suggestion, in some use cases this may result in higher gas usage in the distribution.

```
12 results - 2 files

contracts/staking/BYTES2.sol:
  37:   bytes32 public constant BURN = keccak256("BURN");
  40:   bytes32 public constant ADMIN = keccak256("ADMIN");

contracts/staking/NeoTokyoStaker.sol:
  191:  bytes4 constant private _TRANSFER_FROM_SELECTOR = 0x23b8
  194:  bytes4 constant private _TRANSFER_SELECTOR = 0xa9059cbb;
  197:  uint256 constant private _PRECISION = 1e12;
```

```
  200:  uint256 constant private _DIVISOR = 100;
  203:  uint256 constant private _BYTES_PER_POINT = 200 * 1e18;
  206:  bytes32 public constant CONFIGURE_LP = keccak256("CONFIG
  209:  bytes32 public constant CONFIGURE_TIMELOCKS = keccak256(
  214:  bytes32 public constant CONFIGURE_CREDITS = keccak256("C
  217:  bytes32 public constant CONFIGURE_POOLS = keccak256("CON
  220:  bytes32 public constant CONFIGURE_CAPS = keccak256("CONF
```

[N-10] According to the syntax rules, use `=> mapping (` instead of `=> mapping(` using spaces as keyword

```
contracts/staking/NeoTokyoStaker.sol:
  280:  mapping ( AssetType => mapping ( uint256 => uint256 )) p
  319:  mapping ( address => mapping ( AssetType => uint256 )) p
  326:  mapping ( uint256 => mapping ( string => string )) publi
```

```
contracts/staking/NeoTokyoStaker.sol:
-  372:         mapping ( address => mapping( uint256 => StakedS
+  372:         mapping ( address => mapping ( uint256 => Stakec

- 405:  mapping ( address => mapping( uint256 => StakedS2Citizen
+ 405:  mapping ( address => mapping ( uint256 => StakedS2Citize
```

[N-11] For modern and more readable code; update import usages

Context

```
9 results - 2 files

contracts/staking/BYTES2.sol:
  4: import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
  5: import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
  7: import "../access/PermitControl.sol";
  8: import "../interfaces/IByteContract.sol";
  9: import "../interfaces/IStaker.sol";
```

```
contracts/staking/NeoTokyoStaker.sol:
  4: import "@openzeppelin/contracts/security/ReentrancyGuard.sc
  6: import "../access/PermitControl.sol";
  7: import "../interfaces/IByteContract.sol";
  8: import "../interfaces/IGenericGetter.sol";
```

Description

Solidity code is also cleaner in another way that might not be noticeable: the struct Point. We were importing it previously with global import but not using it. The Point struct `polluted the source code` with an unnecessary object we were not using because we did not need it. This was breaking the rule of modularity and modular programming: `only import what you need` Specific imports with curly braces allow us to apply this rule better.

Recommendation

```
import {contract1 , contract2} from "filename.sol";
```

A good example from the ArtGobblers project;

```
import {Owned} from "solmate/auth/Owned.sol";
import {ERC721} from "solmate/tokens/ERC721.sol";
import {LibString} from "solmate/utils/LibString.sol";
import {MerkleProofLib} from "solmate/utils/MerkleProofLib.sol";
import {FixedPointMathLib} from "solmate/utils/FixedPointMathLib
import {ERC1155, ERC1155TokenReceiver} from "solmate/tokens/ERC1
import {toWadUnsafe, toDaysWadUnsafe} from "solmate/utils/Signec
```

[N-12] Assembly Codes Specific — Should Have Comments

Since this is a low level language that is more difficult to parse by readers, include extensive documentation, comments on the rationale behind its use, clearly explaining what each assembly instruction does.

This will make it easier for users to trust the code, for reviewers to validate the code, and for developers to build on or update the code.

Note that using Assembly removes several important security features of Solidity, which can make the code more insecure and more error-prone.

```
7 results - 1 file

contracts/staking/NeoTokyoStaker.sol:
   833:                 assembly {
   834:                         let length := mload(a)
   886:                 assembly {
   887:                         citizenId := calldataload(0x44)
  1001:                 assembly {
  1002:                         citizenId := calldataload(0x44)
  1236:                 assembly {
  1237:                         switch _assetType
  1461:                 assembly {
  1462:                         citizenId := calldataload(0x24)
  1536:                 assembly {
  1537:                         citizenId := calldataload(0x24)
  1682:                 assembly {
  1683:                         switch _assetType
```

[N-13] Take advantage of Custom Error's return value property

An important feature of Custom Error is that values such as address, tokenID, msg.value can be written inside the `()` sign, this kind of approach provides a serious advantage in debugging and examining the revert details of dapps such as tenderly.

For Example;

```
3 results - 1 file

contracts/staking/NeoTokyoStaker.sol:
  1149:                         revert MismatchedTimelock();
  1712:                         revert LockedConfigurationOfLP()
  1835:                         revert RewardWindowTimesMustIncr
```

[N-14] Pragma version^0.8.19 version too recent to be trusted.

https://github.com/ethereum/solidity/blob/develop/Changelog.md
0.8.19 (2023-02-22)
0.8.17 (2022-09-08)
0.8.16 (2022-08-08)
0.8.15 (2022-06-15)
0.8.10 (2021-11-09)

Unexpected bugs can be reported in recent versions;

- Risks related to recent releases
- Risks of complex code generation changes
- Risks of new language features
- Risks of known bugs

Use a non-legacy and more battle-tested version
Use 0.8.10

[N-15] Inconsistent Solidity Versions

Description

Different Solidity compiler versions are used, the following contracts mix versions:

```
21 results - 21 files

contracts/access/PermitControl.sol:
  2: pragma solidity ^0.8.19;

contracts/interfaces/IByteContract.sol:
  2: pragma solidity ^0.8.19;

contracts/interfaces/IGenericGetter.sol:
  2: pragma solidity ^0.8.19;

contracts/interfaces/IStaker.sol:
  2: pragma solidity ^0.8.19;
```

```
contracts/s1/beckLoot.sol:
  10: pragma solidity ^0.8.0;

contracts/s1/BYTESContract.sol:
  5: pragma solidity ^0.8.0;

contracts/s1/NTCitizenDeploy.sol:
  10: pragma solidity 0.8.11;

contracts/s1/NTItems.sol:
  10: pragma solidity ^0.8.0;

contracts/s1/NTLandDeploy.sol:
  10: pragma solidity ^0.8.0;

contracts/s1/vaultBox.sol:
  10: pragma solidity ^0.8.0;

contracts/s2/NTOuterCitizenDeploy.sol:
  10: pragma solidity 0.8.11;

contracts/s2/NTOuterIdentity.sol:
  10: pragma solidity ^0.8.0;

contracts/s2/NTS2Items.sol:
  10: pragma solidity ^0.8.0;

contracts/s2/NTS2LandDeploy.sol:
  10: pragma solidity ^0.8.0;

contracts/staking/BYTES2.sol:
  2: pragma solidity ^0.8.19;

contracts/staking/NeoTokyoStaker.sol:
  2: pragma solidity ^0.8.19;
```

Recommendation

Versions must be consistent with each other.

[S-01] You can explain the operation of critical functions in NatSpec with an infographic.

An example project;
https://etherscan.io/address/0xe87a68de82204bfa63e4d626d4c5194481cf3b59#code#F1#L244

```
/**

     *** ---------------------------------- ***
     ***                                    ***
     ***     PUBLIC MINTING FUNCTIONS       ***
     ***                                    ***
     *** ---------------------------------- ***
     ***/
    //                       ,-.
    //                       `_'
    //                       /|\
    //                        |                        ,---------
    //                       / \                       |ERC721Drc
    //                     Caller                      `----+----
    //                       |          purchase()          |
    //                       | ---------------------------->
    //                       |                              |
    //                       |                              |
    //             ___________________________________________________
    //             ! ALT  /  drop has no tokens left for caller to
    //             !_____/      |                              |
    //             !            |    revert Mint_SoldOut()     |
    //             !            | <----------------------------
    //             !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    //             !~[noop]~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    //                          |                              |
    //                          |                              |
    //             ___________________________________________________
    //             ! ALT  /  public sale isn't active?         |
    //             !_____/      |                              |
    //             !            |    revert Sale_Inactive()    |
    //             !            | <----------------------------
    //             !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    //             !~[noop]~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    //                          |                              |
    //                          |                              |
    //             ___________________________________________________
    //             ! ALT  /  inadequate funds sent?            |
    //             !_____/      |                              |
    //             !            | revert Purchase_WrongPrice()|
```

```
//            !             | <----------------------------
//            !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//            !~[noop]~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                          |                              |
//                          |                              |----
//                          |                              |
//                          |                              |<---
//                          |                              |
//                          |                              |----
//                          |                              |
//                          |                              |<---
//                          |                              |
//                          | return first minted token ID|
//                          | <----------------------------
//                        Caller                      ,----+----
//                          ,-.                       |ERC721Drc
//                          `_'                       `---------
//                          /|\
//                           |
//                          / \
/**
```

Gas Optimizations

For this contest, 45 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **JCN** received the top score from the judge.

The following wardens also submitted reports: Udsen, nadin, leopoldjoy, Aymen0909, 0x1f8b, hunter_w3b, c3phas, carlitox477, ulqiorra, atharvasama, Angry_Mustache_Man, slvDev, Diana, dharma09, Viktor_Cortess, glcanvas, Flora, 0xSolus, ayden, SAAJ, durianSausage, ReyAdmirado, descharre, Inspex, RaymondFam, Shubham, Madalad, pipoca, MatricksDeCoder, oyc_109, schrodinger, matrix_0wl, MiniGlome, 0xSmartContract, Sathish9098, volodya, fatherOfBlocks, mrpathfindr, Rolezn, ginlee, R-Nemes, arialblack14, 0xnev, *and* 0xhacksmithh.

Summary

A majority of functions that underwent direct changes for optimziations did not have individual tests that showed gas usage and therefore gas savings were explained via

opcodes and EVM gas costs (with the exception of the first issue).

The gas savings in the first issue are substantial and therefore `stake()` and `withdraw()` were benchmarked as POCs for the optimizations that were done in storage and in the `claimReward()` & `getPoolReward()` functions.

Notes

- Some code snippets may be truncated to save space. Code snippets may also be accompanied by `@audit` tags in comments to aid in explaining the issue.
- The `*` next to the gas savings of an issue indicates that some instances are found within loops, meaning the actual gas savings will be greater depending on the number of iterations in the loop.
- The last issue is soley meant to offer clarification for the issue caught by the c4udit tool.

Gas Optimizations

Number	Issue	Instances	Total Gas Saved
G-01	Refactor mapping to save over 44k gas for new users that stake, 10k gas for recurring users that stake, and 10k gas for users that withdraw	-	54200
G-02	State variables can be packed to use fewer storage slots	1	2000
G-03	State variables can be cached instead of re-reading them from storage	27	*2700
G-04	Avoid emitting constants	5	1875
G-05	Multiple accesses of a mapping/array should use a storage pointer	25	*1000
G-06	`x += y/x -= y` costs more gas than `x = x + y/x = x - y` for state variables	13	260
G-07	Use storage instead of memory for structs/arrays	-	-

[G-01] Refactor mapping to save over 44k gas for new users

that stake, 10k gas for recurring users that stake, and 10k gas for users that withdraw

The `lastRewardTime` nested mapping is only modified in the `claimReward()` function and all asset types for the user are given the same value: `block.timestamp` (i.e. 3 different storage slots undergo an SSTORE with the **same value**). The `lastRewardTime` mapping is then accessed in the `getPoolReward()` function and the same values are read for each `AssetType` (i.e. 3 different storage slots, which hold the **same value**, undergo an SLOAD). Due to this flow, the nested mapping is unecessary and a regular mapping would suffice in recording a single `lastRewardTime` for the user since the time will always be the same for ALL assets, as per the `claimReward()` function.

Refactoring the `lastRewardTime` mapping will stop a user from incurring two `Gsset (20000 gas)` and two `Gcoldsload (2100 gas)` when they call `stake()` for the first time, and two `Gsreset (2900 gas)` and two `Gcoldsload (2100 gas)` each time they call `withdraw()` and each subsequent time they call `stake()`.

Note that the flow in which a new user calls `stake()` and then `withdraw()` will result in ~54,200 gas savings between these two function calls + ~10k gas for each subsequent call to `stake()` / `withdraw()`.

Since the `claimReward()` and `getPoolReward()` functions do not have individual tests that show gas usage, the `stake()` and `withdraw()` functions will be shown below as a POC for this gas optimization (both `stake()` and `withdraw()` result in the invocation of `claimReward()` and `getPoolReward()`).

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L319

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L1310

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L1433-L1435

Gas Savings for `stake()`, obtained via protocol's tests: Avg 26202 gas

Note that the average gas savings do not reflect the actual savings for this function since `stake()` will see the greatest savings when called by a new user. Observe the difference in the `Max` column, which will illustrate the savings a user will see when calling `stake()` for the first time (i.e. avoiding two `Gsset (20000 gas)` and two `Gcoldsload (2100 gas)` : 44537 gas.

In addition, the `MIN` column shows the gas savings of the `stake()` function when called by a recurring user (i.e. avoiding two `Gsreset (2900 gas)` and two `Gcoldsload (2100 gas)`): 10337 gas. | | Min | Max | Avg | # calls | | ------ | ------- | -------- | ------- | -------- | | Before | 154353 | 423836 | 277026 | 71 | | After | 144016 | 379299 | 250824 | 71 |

Gas Savings for `withdraw()` , obtained via protocol's tests: Avg 9883 gas

Note that the gas savings for this function will be similar to the savings recurring users experience for the `stake()` function, since in a normal flow the user would call `stake()` first before they call `withdraw()` . | | Min | Max | Avg | # calls | | ------ | -------- | -------- | ------- | -------- | | Before | 102046 | 267114 | 187080 | 18 | | After | 93794 | 256750 | 177197 | 18 |

```
File: contracts/staking/NeoTokyoStaker.sol
319:            mapping ( address => mapping ( AssetType => uint

1310:           uint256 lastPoolRewardTime = lastRewardTime[_rec

1433:           lastRewardTime[_recipient][AssetType.S1_CITIZEN]
1434:           lastRewardTime[_recipient][AssetType.S2_CITIZEN]
1435:           lastRewardTime[_recipient][AssetType.LP] = block
```

```
diff --git a/contracts/staking/NeoTokyoStaker.sol b/contracts/st
index a54d218..51df065 100644
--- a/contracts/staking/NeoTokyoStaker.sol
+++ b/contracts/staking/NeoTokyoStaker.sol
@@ -316,7 +316,7 @@ contract NeoTokyoStaker is PermitControl, Re
        mapping ( AssetType => PoolData ) private _pools;

        /// Track the last time a caller was granted their rewar
-       mapping ( address => mapping ( AssetType => uint256 )) p
+       mapping ( address => uint256 ) public lastRewardTime;
```

```
        /**
                This admin-configurable double-mapping allows us
@@ -1307,7 +1307,7 @@ contract NeoTokyoStaker is PermitControl,
                                applicable time period.
                        */
                        uint256 totalReward;
-                       uint256 lastPoolRewardTime = lastRewardT
+                       uint256 lastPoolRewardTime = lastRewardT
                        uint256 windowCount = pool.rewardCount;
                        for (uint256 i; i < windowCount; ) {
                                RewardWindow memory window = poo
@@ -1430,9 +1430,7 @@ contract NeoTokyoStaker is PermitControl,
                );

                // Record the current time as the beginning time
-               lastRewardTime[_recipient][AssetType.S1_CITIZEN]
-               lastRewardTime[_recipient][AssetType.S2_CITIZEN]
-               lastRewardTime[_recipient][AssetType.LP] = block
+               lastRewardTime[_recipient] = block.timestamp;

                // Calculate total reward and tax.
                uint256 totalReward;
```

[G-02] State variables can be packed to use fewer storage slots

The EVM works with 32 byte words. Variables less than 32 bytes can be declared next to eachother in storage and this will pack the values together into a single 32 byte storage slot (if the values combined are <= 32 bytes). If the variables packed together are retrieved together in functions we will effectively save ~2000 gas with every subsequent SLOAD for that storage slot. This is due to us incurring a `Gwarmaccess (100 gas)` versus a `Gcoldsload (2100 gas)`.

Gas savings: `2000`

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L232-L511

We are able to pack `lpLocked` and `LP` into one storage slot to save 1 SLOT (~2000 gas).

```
File: contracts/staking/NeoTokyoStaker.sol
231:    /// The address of the LP token contract.
232:    address public LP;
...
510:    /// Whether or not setting the LP token contract address
511:    bool public lpLocked;
```

```
diff --git a/contracts/staking/NeoTokyoStaker.sol b/contracts/st
index a54d218..075cb0c 100644
--- a/contracts/staking/NeoTokyoStaker.sol
+++ b/contracts/staking/NeoTokyoStaker.sol
@@ -228,6 +228,9 @@ contract NeoTokyoStaker is PermitControl, Re
        /// The address of the assembled Neo Tokyo S2 Citizen co
        address immutable public S2_CITIZEN;

+       /// Whether or not setting the LP token contract address
+       bool public lpLocked;
+
        /// The address of the LP token contract.
        address public LP;

@@ -507,8 +510,6 @@ contract NeoTokyoStaker is PermitControl, Re
                LPPosition stakedLPPosition;
        }

-       /// Whether or not setting the LP token contract address
-       bool public lpLocked;

        /**
                This struct records an input to the staker's `co
```

[G-03] State variables can be cached instead of re-reading them from storage

Caching of a state variable replaces each `Gwarmaccess (100 gas)` with a much cheaper stack read.

Total Instances: `27`

Gas savings: `27 * 100 = 2700`

Gas savings will actually be greater since storage slot access is occuring within loops in some instances

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L710-L752

Cache `_stakerS1Position[_staker].length` and `_stakerS2Position[_staker].length` to save 2 SLOADs

More SLOADs would actually be saved since storage slot access is occuring within a loop.

```
File: staking/NeoTokyoStaker.sol
710:    function getStakerPositions (
711:            address _staker
712:    ) external view returns (StakerPosition memory) {
713:
714:            // Compile the S1 Citizen details.
715:            StakedS1CitizenOutput[] memory stakedS1Details =
716:                    new StakedS1CitizenOutput[](_stakerS1Pos
717:            for (uint256 i; i < _stakerS1Position[_staker].l
718:                    uint256 citizenId = _stakerS1Position[_s
719:                    StakedS1Citizen memory citizenDetails =
720:                    stakedS1Details[i] = StakedS1CitizenOutp
721:                            citizenId: citizenId,
722:                            stakedBytes: citizenDetails.stak
723:                            timelockEndTime: citizenDetails.
724:                            points: citizenDetails.points,
725:                            hasVault: citizenDetails.hasVaul
726:                            stakedVaultId: citizenDetails.st
727:                    });
728:                    unchecked { i++; }
729:            }
730:
731:            // Compile the S2 Citizen details.
732:            StakedS2CitizenOutput[] memory stakedS2Details =
733:                    new StakedS2CitizenOutput[](_stakerS2Pos
734:            for (uint256 i; i < _stakerS2Position[_staker].l
735:                    uint256 citizenId = _stakerS2Position[_s
736:                    StakedS2Citizen memory citizenDetails =
737:                    stakedS2Details[i] = StakedS2CitizenOutp
738:                            citizenId: citizenId,
739:                            stakedBytes: citizenDetails.stak
```

```
740:                            timelockEndTime: citizenDetails.
741:                            points: citizenDetails.points
742:                    });
743:                    unchecked { i++; }
744:            }
745:
746:            // Return the final output position struct.
747:            return StakerPosition({
748:                    stakedS1Citizens: stakedS1Details,
749:                    stakedS2Citizens: stakedS2Details,
750:                    stakedLPPosition: stakerLPPosition[_stak
751:            });
752:    }
```

```
diff --git a/contracts/staking/NeoTokyoStaker.sol b/contracts/st
index a54d218..8d8ccc4 100644
--- a/contracts/staking/NeoTokyoStaker.sol
+++ b/contracts/staking/NeoTokyoStaker.sol
@@ -712,9 +712,10 @@ contract NeoTokyoStaker is PermitControl, P
        ) external view returns (StakerPosition memory) {

                // Compile the S1 Citizen details.
+               uint256 s1Length = _stakerS1Position[_staker].le
                StakedS1CitizenOutput[] memory stakedS1Details =
-                       new StakedS1CitizenOutput[](_stakerS1Pos
-               for (uint256 i; i < _stakerS1Position[_staker].l
+                       new StakedS1CitizenOutput[](s1Length);
+               for (uint256 i; i < s1Length; ) {
                        uint256 citizenId = _stakerS1Position[_s
                        StakedS1Citizen memory citizenDetails =
                        stakedS1Details[i] = StakedS1CitizenOutp
@@ -729,9 +730,10 @@ contract NeoTokyoStaker is PermitControl, P
                }

                // Compile the S2 Citizen details.
+               uint256 s2Length = _stakerS2Position[_staker].le
                StakedS2CitizenOutput[] memory stakedS2Details =
-                       new StakedS2CitizenOutput[](_stakerS2Pos
-               for (uint256 i; i < _stakerS2Position[_staker].l
+                       new StakedS2CitizenOutput[](s2Length);
+               for (uint256 i; i < s2Length; ) {
                        uint256 citizenId = _stakerS2Position[_s
                        StakedS2Citizen memory citizenDetails =
                        stakedS2Details[i] = StakedS2CitizenOutp
```

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L967-L978

Cache the value from the expression given to `citizenStatus.points` and use that stack variable in other expression to save 1 SLOAD.

For example: `stackVar = (expression)` **=>** `citizenStatus.points = stackVar` **=>** `pool.totalPoints += stackVar`.

```
File: contracts/staking/NeoTokyoStaker.sol
967:            unchecked {
968:                    citizenStatus.points = // @audit: can ca
969:                            identityPoints * vaultMultiplier
970:                            _DIVISOR / _DIVISOR;
971:                    citizenStatus.timelockEndTime = block.ti
972:
973:                    // Record the caller's staked S1 Citizer
974:                    _stakerS1Position[msg.sender].push(citiz
975:
976:                    // Update the pool point weights for rew
977:                    pool.totalPoints += citizenStatus.points
978:            }
```

```diff
diff --git a/contracts/staking/NeoTokyoStaker.sol b/contracts/st
index a54d218..0469c3f 100644
--- a/contracts/staking/NeoTokyoStaker.sol
+++ b/contracts/staking/NeoTokyoStaker.sol
@@ -964,17 +964,19 @@ contract NeoTokyoStaker is PermitControl,

                // Update caller staking information and asset c
                PoolData storage pool = _pools[AssetType.S1_CIT]
+               uint256 points;
                unchecked {
-                       citizenStatus.points =
+                       points =
                                identityPoints * vaultMultiplier
                                _DIVISOR / _DIVISOR;
+                       citizenStatus.points = points;
                        citizenStatus.timelockEndTime = block.ti

                        // Record the caller's staked S1 Citizer
                        _stakerS1Position[msg.sender].push(citiz
```

```
                        // Update the pool point weights for rev
-                       pool.totalPoints += citizenStatus.points
+                       pool.totalPoints += points;
                }
```

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L1021-L1030

Optimization is identical to the one above. Saves 1 SLOAD.

```
File: contracts/staking/NeoTokyoStaker.sol
1021:           unchecked {
1022:                   citizenStatus.points = 100 * timelockMul
1023:                   citizenStatus.timelockEndTime = block.ti
1024:
1025:                   // Record the caller's staked S2 Citizen
1026:                   _stakerS2Position[msg.sender].push(citiz
1027:
1028:                   // Update the pool point weights for rev
1029:                   pool.totalPoints += citizenStatus.points
1030:           }
```

```
diff --git a/contracts/staking/NeoTokyoStaker.sol b/contracts/st
index a54d218..7f03c33 100644
--- a/contracts/staking/NeoTokyoStaker.sol
+++ b/contracts/staking/NeoTokyoStaker.sol
@@ -1018,15 +1018,17 @@ contract NeoTokyoStaker is PermitControl

                // Update caller staking information and asset c
                PoolData storage pool = _pools[AssetType.S2_CIT]
+               uint256 points;
                unchecked {
-                       citizenStatus.points = 100 * timelockMul
+                       points = 100 * timelockMultiplier / _DIV
+                       citizenStatus.points = points;
                        citizenStatus.timelockEndTime = block.ti

                        // Record the caller's staked S2 Citizen
                        _stakerS2Position[msg.sender].push(citiz
```

```
                        // Update the pool point weights for rew
-                       pool.totalPoints += citizenStatus.points
+                       pool.totalPoints += points;
                }
```

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L1044-L1102

Cache `citizenStatus.stakedBytes` to save 4 SLOADs.

```
File: contracts/staking/NeoTokyoStaker.sol
1060:           if (seasonId == 1) {
1061:                   StakedS1Citizen storage citizenStatus =
1062:                   uint256 cap = VAULT_CAP;
1063:                   if (!citizenStatus.hasVault) {
1064:                           cap = NO_VAULT_CAP;
1065:                   }
1066:                   if (citizenStatus.stakedBytes + amount >
1067:                           revert AmountExceedsCap(citizenS
1068:                   }
1069:
1070:                   // Validate that the caller actually sta
1071:                   if (citizenStatus.timelockEndTime == 0)
1072:                           revert CannotStakeIntoUnownedCit
1073:                   }
1074:
1075:                   PoolData storage pool = _pools[AssetType
1076:                   unchecked {
1077:                           uint256 bonusPoints = (amount *
1078:                           citizenStatus.stakedBytes += amo
1079:                           citizenStatus.points += bonusPoi
1080:                           pool.totalPoints += bonusPoints;
1081:                   }
1082:
1083:           // Handle staking BYTES into an S2 Citizen.
1084:           } else if (seasonId == 2) {
1085:                   StakedS2Citizen storage citizenStatus =
1086:                   uint256 cap = NO_VAULT_CAP;
1087:                   if (citizenStatus.stakedBytes + amount >
1088:                           revert AmountExceedsCap(citizenS
1089:                   }
1090:
1091:                   // Validate that the caller actually sta
```

```
1092:                     if (citizenStatus.timelockEndTime == 0)
1093:                             revert CannotStakeIntoUnownedCit
1094:                     }
1095:
1096:                     PoolData storage pool = _pools[AssetType
1097:                     unchecked {
1098:                             uint256 bonusPoints = (amount *
1099:                             citizenStatus.stakedBytes += amo
1100:                             citizenStatus.points += bonusPoi
1101:                             pool.totalPoints += bonusPoints;
1102:                     }
```

```
diff --git a/contracts/staking/NeoTokyoStaker.sol b/contracts/st
index a54d218..93a190d 100644
--- a/contracts/staking/NeoTokyoStaker.sol
+++ b/contracts/staking/NeoTokyoStaker.sol
@@ -1059,12 +1059,13 @@ contract NeoTokyoStaker is PermitControl
                // Handle staking BYTES into an S1 Citizen.
                if (seasonId == 1) {
                        StakedS1Citizen storage citizenStatus =
+                       uint256 s1StakedBytes = citizenStatus.st
                        uint256 cap = VAULT_CAP;
                        if (!citizenStatus.hasVault) {
                                cap = NO_VAULT_CAP;
                        }
-                       if (citizenStatus.stakedBytes + amount >
-                               revert AmountExceedsCap(citizenS
+                       if (s1StakedBytes + amount > cap) {
+                               revert AmountExceedsCap(s1Stakec
                        }

                        // Validate that the caller actually sta
@@ -1075,7 +1076,7 @@ contract NeoTokyoStaker is PermitControl,
                        PoolData storage pool = _pools[AssetType
                        unchecked {
                                uint256 bonusPoints = (amount *
-                               citizenStatus.stakedBytes += amo
+                               citizenStatus.stakedBytes = s1St
                                citizenStatus.points += bonusPoi
                                pool.totalPoints += bonusPoints;
                        }
@@ -1083,9 +1084,10 @@ contract NeoTokyoStaker is PermitControl,
                // Handle staking BYTES into an S2 Citizen.
                } else if (seasonId == 2) {
```

```
                        StakedS2Citizen storage citizenStatus =
+                       uint256 s2StakedBytes = citizenStatus.st
                        uint256 cap = NO_VAULT_CAP;
-                       if (citizenStatus.stakedBytes + amount >
-                               revert AmountExceedsCap(citizenS
+                       if (s2StakedBytes + amount > cap) {
+                               revert AmountExceedsCap(s2Stakec
                        }

                        // Validate that the caller actually sta
@@ -1096,7 +1098,7 @@ contract NeoTokyoStaker is PermitControl,
                        PoolData storage pool = _pools[AssetType
                        unchecked {
                                uint256 bonusPoints = (amount *
-                               citizenStatus.stakedBytes += amc
+                               citizenStatus.stakedBytes = s2St
                                citizenStatus.points += bonusPoi
                                pool.totalPoints += bonusPoints;
                        }
```

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L1137-L1174

Cache `LP` and `stakerLPPosition[msg.sender].multiplier` to save 2 SLOADs.

```
File: contracts/staking/NeoTokyoStaker.sol
1137:           _assetTransferFrom(LP, msg.sender, address(this)
1138:
1139:           // Decode the timelock option's duration and mul
1140:           uint256 timelockDuration = _timelock >> 128;
1141:           uint256 timelockMultiplier = _timelock & type(ui
1142:
1143:           // If this is a new stake of this asset, initial
1144:           if (stakerLPPosition[msg.sender].multiplier == (
1145:                   stakerLPPosition[msg.sender].multiplier
1146:
1147:           // If a multiplier exists already, we must match
1148:           } else if (stakerLPPosition[msg.sender].multipli
1149:                   revert MismatchedTimelock();
1150:           }
1151:
1152:           // Update caller staking information and asset c
```

```
1153:            PoolData storage pool = _pools[AssetType.LP];
1154:            unchecked {
1155:                    uint256 points = amount * 100 / 1e18 * t
1156:
1157:                    // Update the caller's LP token stake.
1158:                    stakerLPPosition[msg.sender].timelockEnc
1159:                            block.timestamp + timelockDurati
1160:                    stakerLPPosition[msg.sender].amount += a
1161:                    stakerLPPosition[msg.sender].points += p
1162:
1163:                    // Update the pool point weights for rew
1164:                    pool.totalPoints += points;
1165:            }
1166:
1167:            // Emit an event recording this LP staking.
1168:            emit Stake(
1169:                    msg.sender,
1170:                    LP, // @audit: 2nd sload for `LP`
1171:                    _timelock,
1172:                    amount
1173:            );
1174:    }
```

```
diff --git a/contracts/staking/NeoTokyoStaker.sol b/contracts/st
index a54d218..3c7343e 100644
--- a/contracts/staking/NeoTokyoStaker.sol
+++ b/contracts/staking/NeoTokyoStaker.sol
@@ -1134,18 +1134,20 @@ contract NeoTokyoStaker is PermitControl
                        contract. This transfer will fail if the
                        tokens.
                */
-               _assetTransferFrom(LP, msg.sender, address(this)
+               address lp = LP;
+               _assetTransferFrom(lp, msg.sender, address(this)

                // Decode the timelock option's duration and mul
                uint256 timelockDuration = _timelock >> 128;
                uint256 timelockMultiplier = _timelock & type(ui

                // If this is a new stake of this asset, initial
-               if (stakerLPPosition[msg.sender].multiplier == (
+               uint256 multiplier = stakerLPPosition[msg.sender
+               if (multiplier == 0) {
                        stakerLPPosition[msg.sender].multiplier
```

```
                // If a multiplier exists already, we must match
-               } else if (stakerLPPosition[msg.sender].multipli
+               } else if (multiplier != timelockMultiplier) {
                        revert MismatchedTimelock();
                }

@@ -1167,7 +1169,7 @@ contract NeoTokyoStaker is PermitControl,
                // Emit an event recording this LP staking.
                emit Stake(
                        msg.sender,
-                       LP,
+                       lp,
                        _timelock,
                        amount
                );
```

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L1264-L1396

Cache `pool.totalPoints` to save 1 SLOAD.

```
File: contracts/staking/NeoTokyoStaker.sol
1273:           PoolData storage pool = _pools[_assetType];
1274:           if (pool.totalPoints != 0) { // @audit: 1st sloa
...
1276:                   // Calculate the total number of points
...
1386:                   // Return final shares.
1387:                   unchecked {
1388:                           uint256 share = points * _PRECIS
1389:                           uint256 daoShare = share * pool.
1390:                           share /= _PRECISION;
1391:                           daoShare /= _PRECISION;
1392:                           return ((share - daoShare), daoS
1393:                   }
1394:           }
1395:           return (0, 0);
1396:   }
```

```
diff --git a/contracts/staking/NeoTokyoStaker.sol b/contracts/st
```

```diff
index a54d218..9431d53 100644
--- a/contracts/staking/NeoTokyoStaker.sol
+++ b/contracts/staking/NeoTokyoStaker.sol
@@ -1271,7 +1271,8 @@ contract NeoTokyoStaker is PermitControl,
                        this case, do not attempt to grant any r
                */
                PoolData storage pool = _pools[_assetType];
-               if (pool.totalPoints != 0) {
+               uint256 totalPoints = pool.totalPoints;
+               if (totalPoints != 0) {

                        // Calculate the total number of points
                        uint256 points;
@@ -1385,7 +1387,7 @@ contract NeoTokyoStaker is PermitControl,

                        // Return final shares.
                        unchecked {
-                               uint256 share = points * _PRECIS
+                               uint256 share = points * _PRECIS
                                uint256 daoShare = share * pool.
                                share /= _PRECISION;
                                daoShare /= _PRECISION;
```

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L1459-L1529

Cache `stakedCitizen.timelockEndTime`, `stakedCitizen.stakedBytes`, `stakedCitizen.stakedVaultId` and `oldPosition.length` to save 6 SLOADs.

More SLOADs would actually be saved since storage slot access is occuring within a loop.

```
File: contracts/staking/NeoTokyoStaker.sol
1459:   function _withdrawS1Citizen () private {
1460:           uint256 citizenId;
1461:           assembly {
1462:                   citizenId := calldataload(0x24)
1463:           }
1464:
1465:           // Validate that the caller has cleared their as
1466:           StakedS1Citizen storage stakedCitizen = stakedS1
1467:           if (block.timestamp < stakedCitizen.timelockEndT
```

```
                revert TimelockNotCleared(stakedCitizen.
        }

        // Validate that the caller actually staked this
        if (stakedCitizen.timelockEndTime == 0) { // @au
                revert CannotWithdrawUnownedS1(citizenIc
        }

        // Return any staked BYTES.
        if (stakedCitizen.stakedBytes > 0) { // @audit:
                _assetTransfer(BYTES, msg.sender, stakec
        }

        // Return any non-component Vault if one is pres
        if (stakedCitizen.stakedVaultId != 0) { // @audi
                _assetTransferFrom(
                        VAULT,
                        address(this),
                        msg.sender,
                        stakedCitizen.stakedVaultId // @
                );
        }

        // Return the S1 Citizen.
        _assetTransferFrom(S1_CITIZEN, address(this), ms

        /*
                Check each citizen ID to find its index
                staked item array of its old position.
        */
        uint256[] storage oldPosition = _stakerS1Positic
        for (uint256 stakedIndex; stakedIndex < oldPosit

                // Remove the element at the matching ir
                if (citizenId == oldPosition[stakedIndex
                        if (stakedIndex != oldPosition.l
                                oldPosition[stakedIndex]
                        }
                        oldPosition.pop();
                        break;
                }
                unchecked { stakedIndex++; }
        }
```

```diff
diff --git a/./contracts/staking/NeoTokyoStaker.sol b/./contract
index a54d218..2fa62b4 100644
--- a/./contracts/staking/NeoTokyoStaker.sol
+++ b/./contracts/staking/NeoTokyoStakerNew.sol
@@ -1464,27 +1464,30 @@ contract NeoTokyoStaker is PermitControl

                // Validate that the caller has cleared their as
                StakedS1Citizen storage stakedCitizen = stakedS1
-               if (block.timestamp < stakedCitizen.timelockEndT
-                       revert TimelockNotCleared(stakedCitizen.
+               uint256 timelockEndTime = stakedCitizen.timelock
+               if (block.timestamp < timelockEndTime) {
+                       revert TimelockNotCleared(timelockEndTin
                }

                // Validate that the caller actually staked this
-               if (stakedCitizen.timelockEndTime == 0) {
+               if (timelockEndTime == 0) {
                        revert CannotWithdrawUnownedS1(citizenIc
                }

                // Return any staked BYTES.
-               if (stakedCitizen.stakedBytes > 0) {
-                       _assetTransfer(BYTES, msg.sender, stakec
+               uint256 stakedBytes = stakedCitizen.stakedBytes;
+               if (stakedBytes > 0) {
+                       _assetTransfer(BYTES, msg.sender, stakec
                }

                // Return any non-component Vault if one is pres
-               if (stakedCitizen.stakedVaultId != 0) {
+               uint256 stakedVaultId = stakedCitizen.stakedVaul
+               if (stakedVaultId != 0) {
                        _assetTransferFrom(
                                VAULT,
                                address(this),
                                msg.sender,
-                               stakedCitizen.stakedVaultId
+                               stakedVaultId
                        );
                }

@@ -1496,12 +1499,13 @@ contract NeoTokyoStaker is PermitControl
                        staked item array of its old position.
                */
```

```
                uint256[] storage oldPosition = _stakerS1Positic
-               for (uint256 stakedIndex; stakedIndex < oldPosit
+               uint256 length = oldPosition.length;
+               for (uint256 stakedIndex; stakedIndex < length;

                        // Remove the element at the matching ir
                        if (citizenId == oldPosition[stakedIndex
-                               if (stakedIndex != oldPosition.l
-                                       oldPosition[stakedIndex]
+                               if (stakedIndex != length - 1) {
+                                       oldPosition[stakedIndex]
                                }
                                oldPosition.pop();
                                break;
```

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L1534-L1592

Cache `stakedCitizen.timelockEndTime`, `stakedCitizen.stakedBytes`, and `oldPosition.length` as stack variables to save 5 SLOADs.

More SLOADs would actually be saved since storage slot access is occuring within a loop.

```
File: contracts/staking/NeoTokyoStaker.sol
1542:           if (block.timestamp < stakedCitizen.timelockEndT
1543:                   revert TimelockNotCleared(stakedCitizen.
1544:           }
1545:
1546:           // Validate that the caller actually staked this
1547:           if (stakedCitizen.timelockEndTime == 0) { // @au
1548:                   revert CannotWithdrawUnownedS2(citizenIc
1549:           }
1550:
1551:           // Return any staked BYTES.
1552:           if (stakedCitizen.stakedBytes > 0) { // @audit:
1553:                   _assetTransfer(BYTES, msg.sender, stakec
1554:           }
1555:
1556:           // Return the S2 Citizen.
1557:           _assetTransferFrom(S2_CITIZEN, address(this), ms
1558:
1559:           /*
```

```
1560:                   Check each citizen ID to find its index
1561:                   staked item array of its old position.
1562:           */
1563:           uint256[] storage oldPosition = _stakerS2Positic
1564:           for (uint256 stakedIndex; stakedIndex < oldPosit
1565:
1566:                   // Remove the element at the matching ir
1567:                   if (citizenId == oldPosition[stakedIndex
1568:                           if (stakedIndex != oldPosition.l
1569:                                   oldPosition[stakedIndex]
1570:                           }
1571:                           oldPosition.pop();
1572:                           break;
1573:                   }
1574:                   unchecked { stakedIndex++; }
1575:           }
```

```
diff --git a/contracts/staking/NeoTokyoStaker.sol b/contracts/st
index a54d218..f975466 100644
--- a/contracts/staking/NeoTokyoStaker.sol
+++ b/contracts/staking/NeoTokyoStaker.sol
@@ -1539,18 +1539,20 @@ contract NeoTokyoStaker is PermitControl

                // Validate that the caller has cleared their as
                StakedS2Citizen storage stakedCitizen = stakedS2
-               if (block.timestamp < stakedCitizen.timelockEndT
-                       revert TimelockNotCleared(stakedCitizen.
+               uint256 timelockEndTime = stakedCitizen.timelock
+               if (block.timestamp < timelockEndTime) {
+                       revert TimelockNotCleared(timelockEndTim
                }

                // Validate that the caller actually staked this
-               if (stakedCitizen.timelockEndTime == 0) {
+               if (timelockEndTime == 0) {
                        revert CannotWithdrawUnownedS2(citizenIc
                }

                // Return any staked BYTES.
-               if (stakedCitizen.stakedBytes > 0) {
-                       _assetTransfer(BYTES, msg.sender, stakec
+               uint256 stakedBytes = stakedCitizen.stakedBytes;
+               if (stakedBytes > 0) {
+                       _assetTransfer(BYTES, msg.sender, stakec
```

```
                }

                // Return the S2 Citizen.
@@ -1561,12 +1563,13 @@ contract NeoTokyoStaker is PermitControl
                        staked item array of its old position.
                */
                uint256[] storage oldPosition = _stakerS2Positic
-               for (uint256 stakedIndex; stakedIndex < oldPosit
+               uint256 length = oldPosition.length;
+               for (uint256 stakedIndex; stakedIndex < length;

                        // Remove the element at the matching ir
                        if (citizenId == oldPosition[stakedIndex
-                               if (stakedIndex != oldPosition.l
-                                       oldPosition[stakedIndex]
+                               if (stakedIndex != length - 1) {
+                                       oldPosition[stakedIndex]
                                }
                                oldPosition.pop();
                                break;
```

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L1605-L1644

Cache `lpPosition.timelockEndTime`, `lpPosition.amount`, and `LP` as stack variables to save 5 SLOADs.

```
File: contracts/staking/NeoTokyoStaker.sol
1605:           if (block.timestamp < lpPosition.timelockEndTime
1606:                   revert TimelockNotCleared(lpPosition.tim
1607:           }
1608:
1609:           // Validate that the caller has enough staked LP
1610:           if (lpPosition.amount < amount) { // @audit 1st
1611:                   revert NotEnoughLPTokens(amount, lpPosit
1612:           }
1613:
1614:           /*
1615:                   Attempt to transfer the LP tokens held i
1616:                   back to the caller.
1617:           */
1618:           _assetTransfer(LP, msg.sender, amount); // @audi
1619:
```

```
1620:            // Update caller staking information and asset c
1621:            PoolData storage pool = _pools[AssetType.LP];
1622:            unchecked {
1623:                    uint256 points = amount * 100 / 1e18 * l
1624:
1625:                    // Update the caller's LP token stake.
1626:                    lpPosition.amount -= amount; // @audit:
1627:                    lpPosition.points -= points;
1628:
1629:                    // Update the pool point weights for rev
1630:                    pool.totalPoints -= points;
1631:            }
1632:
1633:            // If all LP tokens are withdrawn, we must clear
1634:            if (lpPosition.amount == 0) { // @audit 4th sloa
1635:                    lpPosition.multiplier = 0;
1636:            }
1637:
1638:            // Emit an event recording this LP withdraw.
1639:            emit Withdraw(
1640:                    msg.sender,
1641:                    LP, // @audit: 2nd sload for `LP`
1642:                    amount
1643:            );
1644:    }
```

```
diff --git a/./contracts/staking/NeoTokyoStaker.sol b/./contract
index a54d218..a7c34a9 100644
--- a/./contracts/staking/NeoTokyoStaker.sol
+++ b/./contracts/staking/NeoTokyoStakerNew.sol
@@ -1602,20 +1602,23 @@ contract NeoTokyoStaker is PermitControl

                // Validate that the caller has cleared their as
                LPPosition storage lpPosition = stakerLPPositior
-               if (block.timestamp < lpPosition.timelockEndTime
-                       revert TimelockNotCleared(lpPosition.tin
+               uin256 timelockEndTime = lpPosition.timelockEndT
+               if (block.timestamp < timelockEndTime) {
+                       revert TimelockNotCleared(timelockEndTin
                }

                // Validate that the caller has enough staked LP
-               if (lpPosition.amount < amount) {
-                       revert NotEnoughLPTokens(amount, lpPosit
```

```
+                uint256 lpAmount = lpPosition.amount;
+                if (lpAmount < amount) {
+                        revert NotEnoughLPTokens(amount, lpAmour
                 }

                 /*
                         Attempt to transfer the LP tokens held i
                         back to the caller.
                 */
-                _assetTransfer(LP, msg.sender, amount);
+                address lp = LP;
+                _assetTransfer(lp, msg.sender, amount);

                 // Update caller staking information and asset c
                 PoolData storage pool = _pools[AssetType.LP];
@@ -1623,7 +1626,7 @@ contract NeoTokyoStaker is PermitControl,
                         uint256 points = amount * 100 / 1e18 * l

                         // Update the caller's LP token stake.
-                        lpPosition.amount -= amount;
+                        lpPosition.amount = lpAmount - amount;
                         lpPosition.points -= points;

                         // Update the pool point weights for rev
@@ -1631,14 +1634,14 @@ contract NeoTokyoStaker is PermitControl
                 }

                 // If all LP tokens are withdrawn, we must clear
-                if (lpPosition.amount == 0) {
+                if (lpAmount == 0) {
                         lpPosition.multiplier = 0;
                 }

                 // Emit an event recording this LP withdraw.
                 emit Withdraw(
                         msg.sender,
-                        LP,
+                        lp,
                         amount
                 );
         }
```

[G-04] Avoid emitting constants.

A log topic (declared with `indexed`) has a gas cost of `Glogtopic (375 gas)`. The `Stake` and `Withdraw` events' second indexed parameter is a constant for a majority of events emitted (with the exception of the events emitted in the `_stakeLP()` and `_withdrawLP()` functions) and is unecessary to emit since the value will never change. Alternatively, you can avoid incurring the `Glogtopic (375 gas)` per call to any function that emits `Stake` / `Withdraw` (with the exception of `_stakeLP()` and `_withdrawLP()`) by creating separate events for each staking/withdraw function and opt out of using the current `indexed asset` topic in each event. This way you can still query the different staking/withdraw events and will save 375 gas for each staking/withdraw function (with the exception of `_stakeLP()` and `_withdrawLP()`).

Note that the events emitted in the `_stakeLP()` and `withdrawLP()` functions are not considered for this issue since the second indexed parameter is for the `LP` storage variable, which can be changed via the `configureLP()` function.

Instances: 5

Gas Savings: `5 * 375 = 1875`

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L981-L986

`S1_CITIZEN` *is a constant*

```
File: contracts/staking/NeoTokyoStaker.sol
981:            emit Stake(
982:                    msg.sender,
983:                    S1_CITIZEN,
984:                    _timelock,
985:                    citizenId
986:            );
```

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L1033-L1038

`S2_CITIZEN` *is a constant*

```
File: contracts/staking/NeoTokyoStaker.sol
1033:           emit Stake(
1034:                   msg.sender,
1035:                   S2_CITIZEN,
1036:                   _timelock,
1037:                   citizenId
1038:           );
```

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L1110-L1115

`BYTES` *is a constant*

```
File: contracts/staking/NeoTokyoStaker.sol
1110:           emit Stake(
1111:                   msg.sender,
1112:                   BYTES,
1113:                   (seasonId << 128) + citizenId,
1114:                   amount
1115:           );
```

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L1524-L1528

```
File: contracts/staking/NeoTokyoStaker.sol
1524:           emit Withdraw(
1525:                   msg.sender,
1526:                   S1_CITIZEN,
1527:                   citizenId
1528:           );
```

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L1587-L1591

```
File: contracts/staking/NeoTokyoStaker.sol
1587:           emit Withdraw(
1589:                   msg.sender,
1590:                   S2_CITIZEN,
```

```
                    citizenId
            );
```

[G-05] Multiple accesses of a mapping/array should use a storage pointer

Caching a mapping's value in a storage pointer when the value is accessed multiple times saves ~40 gas per access due to not having to perform the same offset calculation every time. Help the Optimizer by saving a storage variable's reference instead of repeatedly fetching it.

To achieve this, declare a storage pointer for the variable and use it instead of repeatedly fetching the reference in a map or an array. As an example, instead of repeatedly calling `_stakerS1Position[_staker]`, save its reference via a storage pointer: `uint256[] storage s1Array = _stakerS1Position[_staker]` and use the pointer instead.

Total instances: 25

Gas savings: `25 * 40 = 1000`

Gas savings will actually be greater since storage slot offset calculation is occuring within loops in some instances

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L715-L744

Cache storage pointers for `_stakerS1Position[_staker]`, `_stakerS2Position[_staker]`, `stakedS1[_staker]`, and `stakedS2[_staker]`.

More gas would actually be saved since storage slot offset calculation is occuring within a loop.

```
File: contracts/staking/NeoTokyoStaker.sol
715:            StakedS1CitizenOutput[] memory stakedS1Details =
716:                    new StakedS1CitizenOutput[](_stakerS1Pos
717:            for (uint256 i; i < _stakerS1Position[_staker].l
```

```
718:                    uint256 citizenId = _stakerS1Position[_s
719:                    StakedS1Citizen memory citizenDetails =
720:                    stakedS1Details[i] = StakedS1CitizenOutp
721:                            citizenId: citizenId,
722:                            stakedBytes: citizenDetails.stak
723:                            timelockEndTime: citizenDetails.
724:                            points: citizenDetails.points,
725:                            hasVault: citizenDetails.hasVaul
726:                            stakedVaultId: citizenDetails.st
727:                    });
728:                    unchecked { i++; }
729:            }
730:
731:            // Compile the S2 Citizen details.
732:            StakedS2CitizenOutput[] memory stakedS2Details =
733:                    new StakedS2CitizenOutput[](_stakerS2Pos
734:            for (uint256 i; i < _stakerS2Position[_staker].l
735:                    uint256 citizenId = _stakerS2Position[_s
736:                    StakedS2Citizen memory citizenDetails =
737:                    stakedS2Details[i] = StakedS2CitizenOutp
738:                            citizenId: citizenId,
739:                            stakedBytes: citizenDetails.stak
740:                            timelockEndTime: citizenDetails.
741:                            points: citizenDetails.points
742:                    });
743:                    unchecked { i++; }
744:            }
```

```
diff --git a/contracts/staking/NeoTokyoStaker.sol b/contracts/st
index a54d218..e078701 100644
--- a/contracts/staking/NeoTokyoStaker.sol
+++ b/contracts/staking/NeoTokyoStaker.sol
@@ -712,11 +712,13 @@ contract NeoTokyoStaker is PermitControl,
        ) external view returns (StakerPosition memory) {

                // Compile the S1 Citizen details.
+               uint256[] storage s1Array = _stakerS1Position[_s
+               mapping (uint256 => StakedS1Citizen) storage s1C
                StakedS1CitizenOutput[] memory stakedS1Details =
-                       new StakedS1CitizenOutput[](_stakerS1Pos
-               for (uint256 i; i < _stakerS1Position[_staker].l
-                       uint256 citizenId = _stakerS1Position[_s
-                       StakedS1Citizen memory citizenDetails =
+                       new StakedS1CitizenOutput[](s1Array.leng
```

```
+                for (uint256 i; i < s1Array.length; ) {
+                        uint256 citizenId = s1Array[i];
+                        StakedS1Citizen memory citizenDetails =
                         stakedS1Details[i] = StakedS1CitizenOutp
                                 citizenId: citizenId,
                                 stakedBytes: citizenDetails.stak
@@ -729,11 +731,13 @@ contract NeoTokyoStaker is PermitControl,
                 }

                 // Compile the S2 Citizen details.
+                uint256[] storage s2Array = _stakerS2Position[_s
+                mapping (uint256 => StakedS2Citizen) storage s2C
                 StakedS2CitizenOutput[] memory stakedS2Details =
-                        new StakedS2CitizenOutput[](_stakerS2Pos
-                for (uint256 i; i < _stakerS2Position[_staker].l
-                        uint256 citizenId = _stakerS2Position[_s
-                        StakedS2Citizen memory citizenDetails =
+                        new StakedS2CitizenOutput[](s2Array.lenc
+                for (uint256 i; i < s2Array.length; ) {
+                        uint256 citizenId = s2Array[i];
+                        StakedS2Citizen memory citizenDetails =
                         stakedS2Details[i] = StakedS2CitizenOutp
                                 citizenId: citizenId,
                                 stakedBytes: citizenDetails.stak
```

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L1144-L1165

Cache storage pointer for `stakerLPPosition[msg.sender]`.

```
File: contracts/staking/NeoTokyoStaker.sol
1144:           if (stakerLPPosition[msg.sender].multiplier == (
1145:                   stakerLPPosition[msg.sender].multiplier
1146:
1147:           // If a multiplier exists already, we must match
1148:           } else if (stakerLPPosition[msg.sender].multipli
1149:                   revert MismatchedTimelock();
1150:           }
1151:
1152:           // Update caller staking information and asset c
1153:           PoolData storage pool = _pools[AssetType.LP];
1154:           unchecked {
1155:                   uint256 points = amount * 100 / 1e18 * t
```

```
1156:
1157:                   // Update the caller's LP token stake.
1158:                   stakerLPPosition[msg.sender].timelockEnc
1159:                           block.timestamp + timelockDurati
1160:                   stakerLPPosition[msg.sender].amount += a
1161:                   stakerLPPosition[msg.sender].points += p
1162:
1163:                   // Update the pool point weights for rew
1164:                   pool.totalPoints += points;
1165:           }
```

```
diff --git a/contracts/staking/NeoTokyoStaker.sol b/contracts/st
index a54d218..7993a96 100644
--- a/contracts/staking/NeoTokyoStaker.sol
+++ b/contracts/staking/NeoTokyoStaker.sol
@@ -1141,11 +1141,12 @@ contract NeoTokyoStaker is PermitControl
                uint256 timelockMultiplier = _timelock & type(ui

                // If this is a new stake of this asset, initial
-               if (stakerLPPosition[msg.sender].multiplier == (
-                       stakerLPPosition[msg.sender].multiplier
+               LPPosition storage stakerPosition = stakerLPPosi
+               if (stakerPosition.multiplier == 0) {
+                       stakerPosition.multiplier = timelockMult

                // If a multiplier exists already, we must match
-               } else if (stakerLPPosition[msg.sender].multipli
+               } else if (stakerPosition.multiplier != timelock
                        revert MismatchedTimelock();
                }

@@ -1155,10 +1156,10 @@ contract NeoTokyoStaker is PermitControl
                        uint256 points = amount * 100 / 1e18 * t

                        // Update the caller's LP token stake.
-                       stakerLPPosition[msg.sender].timelockEnc
+                       stakerPosition.timelockEndTime =
                                block.timestamp + timelockDurati
-                       stakerLPPosition[msg.sender].amount += a
-                       stakerLPPosition[msg.sender].points += p
+                       stakerPosition.amount += amount;
+                       stakerPosition.points += points;

                        // Update the pool point weights for rew
```

```
                        pool.totalPoints += points;
```

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L1210-L1217

Cache storage pointer for `_pools[_assetType]`.

```
File: contracts/staking/NeoTokyoStaker.sol
1210:           if (_pools[_assetType].rewardCount == 0) { // @a
1211:                   revert UnconfiguredPool(uint256(_assetTy
1212:           }
1213:
1214:           // Validate that the asset being staked matches
1215:           if (_pools[_assetType].rewardWindows[0].startTim
1216:                   revert InactivePool(uint256(_assetType))
1217:           }
```

```
diff --git a/contracts/staking/NeoTokyoStaker.sol b/contracts/st
index a54d218..b81a227 100644
--- a/contracts/staking/NeoTokyoStaker.sol
+++ b/contracts/staking/NeoTokyoStaker.sol
@@ -1207,12 +1207,13 @@ contract NeoTokyoStaker is PermitControl
                }

                // Validate that the asset being staked matches
-               if (_pools[_assetType].rewardCount == 0) {
+               PoolData storage pool = _pools[_assetType];
+               if (pool.rewardCount == 0) {
                        revert UnconfiguredPool(uint256(_assetTy
                }

                // Validate that the asset being staked matches
-               if (_pools[_assetType].rewardWindows[0].startTim
+               if (pool.rewardWindows[0].startTime >= block.tim
                        revert InactivePool(uint256(_assetType))
                }
```

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L1277-L1368

Cache storage pointers for `_stakerS1Position[_recipient]`, `_stakerS2Position[_recipient]`, `stakedS1[_recipient]`, `stakedS2[_recipient]`, and `pool.rewardWindows`.

More gas would actually be saved since storage slot offset calculation is occuring within a loop.

```
File: contracts/staking/NeoTokyoStaker.sol
1277:                   uint256 points;
1278:                   if (_assetType == AssetType.S1_CITIZEN)
1279:                           for (uint256 i; i < _stakerS1Pos
1280:                                   uint256 citizenId = _sta
1281:                                   StakedS1Citizen memory s
1282:                                   unchecked {
1283:                                           points += s1Citi
1284:                                           i++;
1285:                                   }
1286:                           }
1287:                   } else if (_assetType == AssetType.S2_CI
1288:                           for (uint256 i; i < _stakerS2Pos
1289:                                   uint256 citizenId = _sta
1290:                                   StakedS2Citizen memory s
1291:                                   unchecked {
1292:                                           points += s2Citi
1293:                                           i++;
1294:                                   }
1295:                           }
1296:                   } else if (_assetType == AssetType.LP) {
1297:                           unchecked {
1298:                                   points += stakerLPPositi
1299:                           }
1300:                   } else {
1301:                           revert InvalidAssetType(uint256(
1302:                   }
1303:
1304:                   /*
1305:                           Determine the reward for the `_r
1306:                           Iterate through the entire array
1307:                           applicable time period.
1308:                   */
1309:                   uint256 totalReward;
1310:                   uint256 lastPoolRewardTime = lastRewardT
1311:                   uint256 windowCount = pool.rewardCount;
1312:                   for (uint256 i; i < windowCount; ) {
```

```
                        RewardWindow memory window = poo

                        /*
                                If the last reward time
                                window, then the reward
                        */
                        if (lastPoolRewardTime < window.
                                uint256 currentRewardRat

                                /*
                                        Iterate forward
                                        windows.
                                */
                                for (uint256 j = i; j <

                                        // If the curren
                                        if (block.timest
                                                unchecke


                                                }

                                                // We ha
                                                i = winc
                                                break;

                                        // Otherwise, ac
                                        } else {
                                                unchecke


                                                }
                                                currentR
                                                lastPool

                                                /*


                                                */
                                                if (j ==
```

```
1360:
1361:
1362:
1363:
1364:
1365:                                                    // Other
1366:                                                    } else {
1367:
1368:                                                    }
1369:                                            }
```

```
diff --git a/contracts/staking/NeoTokyoStaker.sol b/contracts/st
index a54d218..0a58815 100644
--- a/contracts/staking/NeoTokyoStaker.sol
+++ b/contracts/staking/NeoTokyoStaker.sol
@@ -1276,18 +1276,22 @@ contract NeoTokyoStaker is PermitControl
                        // Calculate the total number of points
                        uint256 points;
                        if (_assetType == AssetType.S1_CITIZEN)
-                               for (uint256 i; i < _stakerS1Pos
-                                       uint256 citizenId = _sta
-                                       StakedS1Citizen memory s
+                               uint256[] storage s1Array = _sta
+                               mapping (uint256 => StakedS1Citi
+                               for (uint256 i; i < s1Array.lenc
+                                       uint256 citizenId = s1Ar
+                                       StakedS1Citizen memory s
                                        unchecked {
                                                points += s1Citi
                                                i++;
                                        }
                               }
                        } else if (_assetType == AssetType.S2_CI
-                               for (uint256 i; i < _stakerS2Pos
-                                       uint256 citizenId = _sta
-                                       StakedS2Citizen memory s
+                               uint256[] storage s2Array = _sta
+                               mapping (uint256 => StakedS2Citi
+                               for (uint256 i; i < s2Array.lenc
+                                       uint256 citizenId = s2Ar
+                                       StakedS2Citizen memory s
                                        unchecked {
                                                points += s2Citi
                                                i++;
```

```
@@ -1309,15 +1313,16 @@ contract NeoTokyoStaker is PermitControl
                        uint256 totalReward;
                        uint256 lastPoolRewardTime = lastRewardT
                        uint256 windowCount = pool.rewardCount;
+                       mapping (uint256 => RewardWindow) storag
                        for (uint256 i; i < windowCount; ) {
-                               RewardWindow memory window = poo
+                               RewardWindow memory window = rew

                                /*
                                        If the last reward time
                                        window, then the reward
                                */
                                if (lastPoolRewardTime < window.
-                                       uint256 currentRewardRat
+                                       uint256 currentRewardRat

                                        /*
                                                Iterate forward
@@ -1363,7 +1368,7 @@ contract NeoTokyoStaker is PermitControl,

                                                        // Other
                                                        } else {
-
+
                                                        }
                                                }
                                                unchecked { j++;
```

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L1432-L1435

Cache storage pointer for `lastRewardTime[_recipient]`.

```
File: contracts/staking/NeoTokyoStaker.sol
1432:           // Record the current time as the beginning time
1433:           lastRewardTime[_recipient][AssetType.S1_CITIZEN]
1434:           lastRewardTime[_recipient][AssetType.S2_CITIZEN]
1435:           lastRewardTime[_recipient][AssetType.LP] = block
```

```
diff --git a/contracts/staking/NeoTokyoStaker.sol b/contracts/st
```

```diff
index a54d218..4621b10 100644
--- a/contracts/staking/NeoTokyoStaker.sol
+++ b/contracts/staking/NeoTokyoStaker.sol
@@ -1430,9 +1430,10 @@ contract NeoTokyoStaker is PermitControl,
                );

                // Record the current time as the beginning time
-               lastRewardTime[_recipient][AssetType.S1_CITIZEN]
-               lastRewardTime[_recipient][AssetType.S2_CITIZEN]
-               lastRewardTime[_recipient][AssetType.LP] = block
+               mapping (AssetType => uint256) storage recipient
+               recipient[AssetType.S1_CITIZEN] = block.timestan
+               recipient[AssetType.S2_CITIZEN] = block.timestan
+               recipient[AssetType.LP] = block.timestamp;

                // Calculate total reward and tax.
                uint256 totalReward;
```

[G-06] `x += y/x -= y` costs more gas than `x = x + y/x = x - y` for state variables

Total Instances: 13

Gas Savings: `13 * 20 = 260`

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L875-L987

```
File: contracts/staking/NeoTokyoStaker.sol
967:            unchecked {
968:                    citizenStatus.points =
969:                            identityPoints * vaultMultiplier
970:                            _DIVISOR / _DIVISOR;
971:                    citizenStatus.timelockEndTime = block.ti
972:
973:                    // Record the caller's staked S1 Citizer
974:                    _stakerS1Position[msg.sender].push(citiz
975:
976:                    // Update the pool point weights for rew
977:                    pool.totalPoints += citizenStatus.points
978:            }
```

```diff
diff --git a/contracts/staking/NeoTokyoStaker.sol b/contracts/st
index a54d218..f7c4a13 100644
--- a/contracts/staking/NeoTokyoStaker.sol
+++ b/contracts/staking/NeoTokyoStaker.sol
@@ -974,7 +974,7 @@ contract NeoTokyoStaker is PermitControl, Re
                        _stakerS1Position[msg.sender].push(citiz

                        // Update the pool point weights for rew
-                       pool.totalPoints += citizenStatus.points
+                       pool.totalPoints = pool.totalPoints + ci
                }
```

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L1021-L1030

```
File: contracts/staking/NeoTokyoStaker.sol
1021:           unchecked {
1022:                   citizenStatus.points = 100 * timelockMul
1023:                   citizenStatus.timelockEndTime = block.ti
1024:
1025:                   // Record the caller's staked S2 Citizen
1026:                   _stakerS2Position[msg.sender].push(citiz
1027:
1028:                   // Update the pool point weights for rew
1029:                   pool.totalPoints += citizenStatus.points
1030:           }
```

```diff
diff --git a/contracts/staking/NeoTokyoStaker.sol b/contracts/st
index a54d218..e940f8e 100644
--- a/contracts/staking/NeoTokyoStaker.sol
+++ b/contracts/staking/NeoTokyoStaker.sol
@@ -1026,7 +1026,7 @@ contract NeoTokyoStaker is PermitControl,
                        _stakerS2Position[msg.sender].push(citiz

                        // Update the pool point weights for rew
-                       pool.totalPoints += citizenStatus.points
+                       pool.totalPoints = pool.totalPoints + ci
                }
```

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L1097-L1102

```
File: contracts/staking/NeoTokyoStaker.sol
1097:                   unchecked {
1098:                           uint256 bonusPoints = (amount *
1099:                           citizenStatus.stakedBytes += amo
1100:                           citizenStatus.points += bonusPoi
1101:                           pool.totalPoints += bonusPoints;
1102:                   }
```

```
diff --git a/contracts/staking/NeoTokyoStaker.sol b/contracts/st
index a54d218..af3d3de 100644
--- a/contracts/staking/NeoTokyoStaker.sol
+++ b/contracts/staking/NeoTokyoStaker.sol
@@ -1096,9 +1096,9 @@ contract NeoTokyoStaker is PermitControl,
                        PoolData storage pool = _pools[AssetType
                        unchecked {
                                uint256 bonusPoints = (amount *
-                               citizenStatus.stakedBytes += amo
-                               citizenStatus.points += bonusPoi
-                               pool.totalPoints += bonusPoints;
+                               citizenStatus.stakedBytes = citi
+                               citizenStatus.points = citizenSt
+                               pool.totalPoints = pool.totalPoi
                        }
```

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L1154-L1165

```
File: contracts/staking/NeoTokyoStaker.sol
1154:           unchecked {
1155:                   uint256 points = amount * 100 / 1e18 * t
1156:
1157:                   // Update the caller's LP token stake.
1158:                   stakerLPPosition[msg.sender].timelockEnc
1159:                           block.timestamp + timelockDurati
1160:                   stakerLPPosition[msg.sender].amount += a
1161:                   stakerLPPosition[msg.sender].points += p
1162:
1163:                   // Update the pool point weights for rew
```

```
1164:                   pool.totalPoints += points;
1165:           }
```

```
diff --git a/contracts/staking/NeoTokyoStaker.sol b/contracts/st
index a54d218..7f52897 100644
--- a/contracts/staking/NeoTokyoStaker.sol
+++ b/contracts/staking/NeoTokyoStaker.sol
@@ -1157,11 +1157,11 @@ contract NeoTokyoStaker is PermitControl
                        // Update the caller's LP token stake.
                        stakerLPPosition[msg.sender].timelockEnc
                                block.timestamp + timelockDurati
-                       stakerLPPosition[msg.sender].amount += a
-                       stakerLPPosition[msg.sender].points += p
+                       stakerLPPosition[msg.sender].amount = st
+                       stakerLPPosition[msg.sender].points = st

                        // Update the pool point weights for rew
-                       pool.totalPoints += points;
+                       pool.totalPoints = pool.totalPoints + pc
                }
```

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L1514-L1516

```
File: contracts/staking/NeoTokyoStaker.sol
1514:           unchecked {
1515:                   pool.totalPoints -= stakedCitizen.points
1516:           }
```

```
diff --git a/contracts/staking/NeoTokyoStaker.sol b/contracts/st
index a54d218..89a1f6d 100644
--- a/contracts/staking/NeoTokyoStaker.sol
+++ b/contracts/staking/NeoTokyoStaker.sol
@@ -1512,7 +1512,7 @@ contract NeoTokyoStaker is PermitControl,
                // Update caller staking information and asset c
                PoolData storage pool = _pools[AssetType.S1_CIT]
                unchecked {
-                       pool.totalPoints -= stakedCitizen.points
+                       pool.totalPoints = pool.totalPoints - st
                }
```

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L1579-L1581

```
File: contracts/staking/NeoTokyoStaker.sol
1579:           unchecked {
1580:                   pool.totalPoints -= stakedCitizen.points
1581:           }
```

```
diff --git a/contracts/staking/NeoTokyoStaker.sol b/./contracts/
index a54d218..f93562f 100644
--- a/contracts/staking/NeoTokyoStaker.sol
+++ b/contracts/staking/NeoTokyoStaker.sol
@@ -1577,7 +1577,7 @@ contract NeoTokyoStaker is PermitControl,
                // Update caller staking information and asset c
                PoolData storage pool = _pools[AssetType.S2_CIT]
                unchecked {
-                       pool.totalPoints -= stakedCitizen.points
+                       pool.totalPoints = pool.totalPoints - st
                }
```

https://github.com/code-423n4/2023-03-neotokyo/blob/main/contracts/staking/NeoTokyoStaker.sol#L1622-L1631

```
File: contracts/staking/NeoTokyoStaking.sol
1622:           unchecked {
1623:                   uint256 points = amount * 100 / 1e18 * l
1624:
1625:                   // Update the caller's LP token stake.
1626:                   lpPosition.amount -= amount;
1627:                   lpPosition.points -= points;
1628:
1629:                   // Update the pool point weights for rew
1630:                   pool.totalPoints -= points;
1631:           }
```

```
diff --git a/contracts/staking/NeoTokyoStaker.sol b/contracts/st
index a54d218..2513548 100644
--- a/contracts/staking/NeoTokyoStaker.sol
+++ b/contracts/staking/NeoTokyoStaker.sol
```

```
@@ -1623,11 +1623,11 @@ contract NeoTokyoStaker is PermitControl
                        uint256 points = amount * 100 / 1e18 * l

                        // Update the caller's LP token stake.
-                       lpPosition.amount -= amount;
-                       lpPosition.points -= points;
+                       lpPosition.amount = lpPosition.amount -
+                       lpPosition.points = lpPosition.points -

                        // Update the pool point weights for rev
-                       pool.totalPoints -= points;
+                       pool.totalPoints = pool.totalPoints - pc
                }
```

[G-07] Use storage instead of memory for structs/arrays

If you are not returning an entire struct (all fields) in a function then it is more gas efficient to use a storage pointer rather than using a memory pointer (which copies all the values in the struct from storage into memory). If you use a memory pointer and do not access all the struct fields in your function, then you are performing unecessary sloads. With this in mind, only the 4th, 6th, 13th, 14th, and 15th instances caught by the `c4udit tool` are instances where a struct is being copied from storage into memory and therefore have the hightest potential for gas savings. Below are some clarifications regarding those instances:

- Modifying the 4th and 6th instances to use a storage pointer will not yield any substantial gas savings since all of the struct values are accessed (meaning the same amount of sloads would occur if you used a storage pointer) in the function. In addition, all the struct values are also being returned in memory.
- Modifying the 15th instance to use a storage pointer would actually be less efficient in some cases since the struct values can potentially be read more than once during the function call. When using a memory pointer, the values would be loaded (MLOAD) from memory each time. When using a storage pointer, the values would be loaded (SLOAD) from storage each time. If you were to use a storage pointer in this instance you should consider caching the `window.startTime` and `window.reward` values to avoid re-reading those values from storage.
- Modifying the 13th and 14th instances to use a storage pointer will result in substantial gas savings due to the fact that only **one** struct value is intended to

be read during each loop. Although the function only reads one struct value from memory, the result of using a memory pointer is that ALL of the struct values are loaded from storage (5 unecessary SLOADs for the `StakedS1Citizen` struct and 2 unecessary SLOADs from the `StakedS2Citizen` struct) into memory. Using a storage pointer instead of a memory pointer for these two instances would result in the following gas savings:

Gas savings for `stake()`, obtained via protocol's tests: Avg 4463 gas

	Min	Max	Avg	# calls
Before	154353	423836	277026	71
After	149743	423836	272563	71

Gas Savings for `withdraw()`, obtained via protocol's tests: Avg 5346 gas

	Min	Max	Avg	# calls
Before	102046	267114	187080	18
After	102046	257972	181734	18

Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top