

# Canto Identity Subprotocols contest Findings & Analysis Report

2023-05-09

## Table of contents

- [Overview](#)
  - [About C4](#)
  - [Wardens](#)
- [Summary](#)
- [Scope](#)
- [Severity Criteria](#)
- [High Risk Findings \(1\)](#)
  - [\[H-01\] Users will be able to purchase fewer NFTs than the project had anticipated](#)
- [Medium Risk Findings \(9\)](#)
  - [\[M-01\] ProfilePicture subprotocol is immutably linked by `subprotocolName` to the CID protocol](#)
  - [\[M-02\] The range of `iteratePRNG` limits the number of Zalgo distortions](#)
  - [\[M-03\] `characterModifier` is `uint8` but encodes `1.38e24` different Zalgo distortions](#)
  - [\[M-04\] Bio Protocol - `tokenURI` JSON injection](#)

- [M-05] If the underlying NFT is burned, getPFP may return incorrect results
- [M-06] Incorrect emoji displaying
- [M-07] Users can end up buying and paying for a different Tray than the one they were trying to acquire
- [M-08] Bio lines will overflow the buffer for repeated “continuation” characters
- [M-09] Bio NFT incorrectly breaks SVG lines and doesn’t support more than 120 characters effectively
- Low Risk and Non-Critical Issues
  - L-01 AVOID USING TX.ORIGIN
  - L-02 LOSS OF PRECISION DUE TO ROUNDING
  - L-03 Consider using OpenZeppelin’s SafeCast library to prevent unexpected overflows when casting from various type int/uint values
  - L-04 Prevent division by 0
  - N-01 Named imports can be used
  - N-02 AVOID HARDCODED VALUES
  - N-03 Include return parameters in NatSpec comments
  - N-04 ADD PARAMETER TO EVENT-EMIT
  - N-05 NOT USING THE NAMED RETURN VARIABLES ANYWHERE IN THE FUNCTION IS CONFUSING
  - N-06 Interchangeable usage of uint and uint256
  - N-07 Immutables should be in uppercase
  - N-08 Move IF/validation statements to the top of the function when validating input parameters
  - N-09 FOR FUNCTIONS, FOLLOW SOLIDITY STANDARD NAMING CONVENTIONS
  - N-10 Don’t use numbers without explanation, it’s not a good code practice. It’s possible to use constants.
  - N-11 Use underscores for number literals
  - N-12 NO SAME VALUE INPUT CONTROL

- [N-13 Need Fuzzing test](#)
- [N-14 NatSpec comments should be increased in contracts](#)
- [N-15 Function writing that does not comply with the Solidity Style Guide](#)
- [N-16 Add a timelock to critical functions](#)
- [N-17 Long lines are not suitable for the 'Solidity Style Guide'](#)
- [N-18 Assembly Codes Specific — Should Have Comments](#)
- [N-19 Critical changes should use two-step procedure](#)
- [N-20 Missing NATSPEC](#)
- [N-21 Imports can be grouped together](#)
- [S-01 Project Upgrade and Stop Scenario should be](#)
- [Gas Optimizations](#)
  - [Summary](#)
  - [G-01 Gas overflow during iteration \(DoS\)](#)
  - [G-02 State variables only set in the constructor should be declared immutable](#)
  - [G-03 Failure to check the zero address in the constructor causes the contract to be deployed again](#)
  - [G-04 By updating `\_emit` , extra emit usage can be reduced](#)
  - [G-05 Emit can be rearranged](#)
  - [G-06 Use `assembly` to write \*address storage values\*](#)
  - [G-07 Functions \*guaranteed to revert\* when called by normal users can be marked `payable`](#)
  - [G-08 Multiple `address` /ID mappings can be combined into a single `mapping` of an `address` /ID to a `struct` , where appropriate](#)
  - [G-09 Use a more recent version of solidity](#)
  - [G-10 The result of function calls should be cached rather than re-calling the function](#)
  - [G-11 Avoid contract existence checks by using low level calls](#)
  - [G-12 `bytes` constants are more efficient than `string` constans](#)

- [G-13 Change `public` function visibility to `external`](#)
- [G-14 Usage of uints/ints smaller than 32 bytes \(256 bits\) incurs overhead](#)
- [G-15 Setting the `constructor` to `payable`](#)
- [G-16 Use nested if and, avoid multiple check combinations](#)
- [G-17 Sort Solidity operations using short-circuit mode](#)
- [G-18 Use ERC721A instead ERC721](#)
- [G-19 Optimize names to save gas](#)
- [G-20 Upgrade Solidity's optimizer](#)

- [Disclosures](#)



## 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 Canto Identity Subprotocols smart contract system written in Solidity. The audit contest took place between March 17—March 20 2023.



## Wardens

101 Wardens contributed reports to the Canto Identity Subprotocols contest:

1. [OxAgro](#)
2. [OxSmartContract](#)
3. [Oxdaydream](#)
4. [Oxnev](#)
5. Awesome

6. [Aymen0909](#)
7. BRONZEDISC
8. [Bauchibred](#)
9. [Chom](#)
10. Deathstore
11. Deekshith99
12. Diana
13. [Emmanuel](#)
14. Englave
15. EvanW
16. Fanz
17. Haipls
18. IceBear
19. IgorZuk
20. J4de
21. [JCN](#)
22. JerryOx
23. [K42](#)
24. Kek
25. Kresh
26. Madalad
27. Matin
28. MiniGlome
29. ParadOx
30. Polaris\_tow
31. Rageur
32. [Rappie](#)
33. [Respx](#)
34. ReyAdmirado

- 35. Rolezn
- 36. [Ruhum](#)
- 37. SAAJ
- 38. SaeedAlipoor01988
- 39. [Sathish9098](#)
- 40. [Shubham](#)
- 41. [Stryder](#)
- 42. SunSec
- 43. TIMOH
- 44. [Udsen](#)
- 45. Viktor\_Cortess
- 46. Walter
- 47. Wander (xAlismx, ubl4nk, and mahdikarimi)
- 48. [adriro](#)
- 49. [alejandrocovrr](#)
- 50. anodaram
- 51. arialblack14
- 52. atharvasama
- 53. [bin2chen](#)
- 54. caspersolangii
- 55. cccz
- 56. chaduke
- 57. codeslide
- 58. cryptonue
- 59. d3e4
- 60. dec3ntraliz3d
- 61. descharre
- 62. dicethedev
- 63. [dingo2077](#)

- 64. djsxloit
- 65. [fatherOfBlocks](#)
- 66. [felipe](#)
- 67. fs0c
- 68. ginlee
- 69. glcanvas
- 70. [hihen](#)
- 71. igungu
- 72. jack
- 73. jasonxiale
- 74. [joestakey](#)
- 75. [juancito](#)
- 76. [leopoldjoy](#)
- 77. libratus
- 78. lukris02
- 79. luxartvinsec
- 80. m9800
- 81. mjmoonwalker
- 82. mojito\_auditor
- 83. [nadin](#)
- 84. nasri136
- 85. pipoca
- 86. popular00
- 87. reassor
- 88. rokso
- 89. scokaf (Scoon and jauvany)
- 90. shark
- 91. [slvDev](#)
- 92. tnevler

93. [turvy\\_fuzz](#)

94. ulqiorra

95. vagrant

96. [viking71](#)

97. volodya

98. wait

This contest was judged by [Oxleastwood](#).

Final report assembled by [liveactionllama](#).



## Summary

The C4 analysis yielded an aggregated total of 10 unique vulnerabilities. Of these vulnerabilities, 1 received a risk rating in the category of HIGH severity and 9 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 48 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 41 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 Canto Identity Subprotocols contest repository](#), and is composed of 1 library and 4 smart contracts written in the Solidity programming language and includes 687 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 (1)



### [H-01] Users will be able to purchase fewer NFTs than the project had anticipated

*Submitted by [volodya](#), also found by [m9800](#), [Emmanuel](#), [IgorZuk](#), [dec3ntraliz3d](#), [igingu](#), [adriro](#), [Rappie](#), and [descharre](#)*

Users will be able to purchase fewer NFTs than the project had anticipated. The project had expected that users would be able to purchase a range of variations using both text and emoji characters. However, in reality, users will only be able to purchase a range of variations using emoji characters.

For example, the list of characters available for users to choose from is as follows  
image

For instance, if a user chooses to mint an NFT namespace using font class 2 and the single letter *a*, then theoretically all other users should be able to mint font class 0 using the first emoji in the list, font class 1 using the single letter “a,” font class 3 using the single letter *α*, and so on, the first letter on every class will be. However, in reality, they will not be able to do so.

I consider this to be a critical issue because the project may not be able to sell as many NFTs as expected, potentially resulting in a loss of funds.

Here is an how nft name and their svg will look like from what I described above. As you can see emoji replaced letters in the name.



This is a function that creates namespace out of tray.

```
canto-namespace-protocol/src/Namespace.sol#L110
function fuse(CharacterData[] calldata _characterList) external
    uint256 numCharacters = _characterList.length;
    if (numCharacters > 13 || numCharacters == 0) revert InvalidLength();
    uint256 fusingCosts = 2**(13 - numCharacters) * 1e18;
    SafeTransferLib.safeTransferFrom(note, msg.sender, revert(), fusingCosts);
    uint256 namespaceIDToMint = ++nextNamespaceIDToMint;
    Tray.TileData[] storage nftToMintCharacters = nftCharacters;
    bytes memory bName = new bytes(numCharacters * 33); // 33 bytes per character
    uint256 numBytes;
    // Extract unique trays for burning them later on
    uint256 numUniqueTrays;
    uint256[] memory uniqueTrays = new uint256[](_characterList.length);
    for (uint256 i; i < numCharacters; ++i) {
        bool isLastTrayEntry = true;
        uint256 trayID = _characterList[i].trayID;
        uint8 tileOffset = _characterList[i].tileOffset;
        // Check for duplicate characters in the provided list
        for (uint256 j = i + 1; j < numCharacters; ++j) {
            if (_characterList[j].trayID == trayID) {
                isLastTrayEntry = false;
                if (_characterList[j].tileOffset == tileOffset) {
                    continue;
                }
            }
        }
        Tray.TileData memory tileData = tray.getTile(trayID, tileOffset);
        uint8 characterModifier = tileData.characterModifier;

        if (tileData.fontClass != 0 && _characterList[i].skinToneModifier != 0) {
            revert CannotFuseCharacterWithSkinTone();
        }

        if (tileData.fontClass == 0) {
            // Emoji
            characterModifier = _characterList[i].skinToneModifier;
        }
        bytes memory charAsBytes = Utils.characterToUnicodeF
    ...
```

[canto-namespace-protocol/src/Namespace.sol#L110](#)

There is a bug in this line of code where a character is retrieved from tile data. Instead of passing `tileData.fontClass` , we are passing `0` .

```
bytes memory charAsBytes = Utils.characterToUnicodeF
```

Due to this bug, the names for all four different font classes will be the same. As a result, they will point to an existing namespace, and later, there will be a check for the existence of that name (token) using `NameAlreadyRegistered`.

```
string memory nameToRegister = string(bName);
uint256 currentRegisteredID = nameToToken[nameToRegister]
if (currentRegisteredID != 0) revert NameAlreadyRegistered
```



## Proof of Concept

Here is the test that you can run

```
function testFailMintSameCharacterIndex() public {
    address user = user1;
    note.mint(user, 10000e18);
    endPrelaunchAndBuyOne(user);

    uint256[] memory trayIds = buyTray(user, 3);
    vm.startPrank(user);
    note.approve(address(ns), type(uint256).max);
    Namespace.CharacterData[] memory list = new Namespace.CharacterData[
        1
    ];
    // fuse tile with fontClass=8,characterIndex=1
    list[0] = Namespace.CharacterData(trayIds[1], 4, 0);
    Tray.TileData memory tileData = tray.getTile(trayIds[1],
    console.log(tileData.characterIndex); //1
    console.log(tileData.fontClass); //8
    ns.fuse(list);

    // fuse tile with fontClass=4,characterIndex=1
    list[0] = Namespace.CharacterData(trayIds[2], 3, 0);
    tileData = tray.getTile(trayIds[2], 3);
    console.log(tileData.characterIndex); //1
    console.log(tileData.fontClass); //4
```

```
vm.expectRevert(  
    abi.encodeWithSelector(namespace.NameAlreadyRegister  
);  
ns.fuse(list);  
}
```



## Tools Used

Manual review, forge tests



## Recommended Mitigation Steps

Pass font class instead of 0

```
- bytes memory charAsBytes = Utils.characterToUnicode  
+ bytes memory charAsBytes = Utils.characterToUnicode
```

[OpenCoreCH \(Canto Identity\)](#) confirmed and commented:

Agree, leftover from earlier, will be changed.



## Medium Risk Findings (9)



[M-01] ProfilePicture subprotocol is immutably linked by  
subprotocolName to the CID protocol

Submitted by [d3e4](#)

<https://github.com/code-423n4/2023-03-canto-identity/blob/077372297fc419ea7688ab62cc3fd4e8f4e24e66/canto-pfp-protocol/src/ProfilePicture.sol#L59>

<https://github.com/code-423n4/2023-03-canto-identity/blob/077372297fc419ea7688ab62cc3fd4e8f4e24e66/canto-pfp-protocol/src/ProfilePicture.sol#L99>

Besides having to re-register the protocol, it will also have to be redeployed.



## Proof of Concept

A protocol is registered by name in the `SubprotocolRegistry`. Quoting the [Canto Identity Protocol contest details](#):

“In theory, someone can front-run a call to `SubprotocolRegistry.register` with the same name, causing the original call to fail. There is a registration fee ( `100 $NOTE` ) and the damage is very limited (original registration call fails, user can just re-register under a different name), so this attack is not deemed feasible.”

However, the `ProfilePicture` subprotocol suffers from the additional issue of also having to be redeployed, making this exploit more severe.

In `ProfilePicture`, `subprotocolName` [is set on construction](#).

```
constructor(address _cidNFT, string memory _subprotocolName) ERC20(
    _cidNFT, "Canto Identity Protocol", 18) {
    cidNFT = ICidNFT(_cidNFT);
    subprotocolName = _subprotocolName;
```

`subprotocolName` is used to [retrieve the `cidNFTID` to which the subprotocols NFT is added](#).

```
uint256 cidNFTID = cidNFT.getPrimaryCIDNFT(subprotocolName, _profilePicture);
IAddressRegistry addressRegistry = cidNFT.addressRegistry();
if (cidNFTID == 0 || addressRegistry.getAddress(cidNFTID) != ERC20) {
    nftContract = address(0);
    nftID = 0; // Strictly not needed because nftContract has to be set
}
```

This means that this subprotocol *must* be registered under this name, or it will not work. So if `ProfilePicture` is deployed but someone else manages to register it in the `SubprotocolRegistry` before the owner (recipient of fees) of `ProfilePicture` (by frontrunning or otherwise), it cannot simply be re-registered under a different name, but would have to be redeployed with a new name under which it then can be registered.



## Recommended Mitigation Steps

Since the subprotocol is meant to be registered with the CID protocol, its deployment and registration should be atomic. Or the name can be initialized after deployment and registration, and set by a (temporary) owner to the name it then has been registered with.

### [OpenCoreCH \(Canto Identity\) acknowledged and commented:](#)

We will do deployment and registration in one transaction, but the issue is generally true.



## [M-02] The range of `iteratePRNG` limits the number of Zalgo distortions

Submitted by [d3e4](#)

<https://github.com/code-423n4/2023-03-canto-identity/blob/077372297fc419ea7688ab62cc3fd4e8f4e24e66/canto-namespace-protocol/src/Utils.sol#L136-L174>  
<https://github.com/code-423n4/2023-03-canto-identity/blob/077372297fc419ea7688ab62cc3fd4e8f4e24e66/canto-namespace-protocol/src/Utils.sol#L256-L261>

Only a tiny fraction of all Zalgo distortions are accessible.



## Proof of Concept

In `characterToUnicodeBytes`, [for font class 7 \(i.e. Zalgo\)](#), the `_characterModifier` determines the Zalgo distortion. The distortion is pseudo-randomly calculated by using `_characterModifier` as a seed and iteratively passing it through `iteratePRNG`, each intermediate value being used to set, in turn, each component of the distortion.

```
} else if (_fontClass == 7) {  
    // Zalgo  
    uint8 asciiStartingIndex = 97;  
    uint256 numAbove = (_characterModifier % 7) + 1;
```

```
// We do not reuse the same seed for the following generatic
_characterModifier = iteratePRNG(_characterModifier);
uint256 numMiddle = _characterModifier % 2;
_characterModifier = iteratePRNG(_characterModifier);
uint256 numBelow = (_characterModifier % 7) + 1;
// ...
```

This has the effect that the initial seed value generates a stream of values, equidistributed within their respective ranges.

A Zalgo tile is defined by a letter and a modification consisting of a combination of characters above, over and below the letter. The modification consists of a selection of 1 up to 7 characters from a set of 46 characters above the letter, 0 or 1 from 5 over the letter, and 1 to 7 from 47 below the letter. Thus there is a total of  $(46^1 + 46^2 + \dots + 46^7) \cdot (5^0 + 5^1) \cdot (47^1 + 47^2 + \dots + 47^7) = 1.38e24$  different Zalgo modifications possible per letter.

However, `iteratePRNG` modulates its return value by 2038074743 :

```
function iteratePRNG(uint256 _currState) internal pure returns (
    unchecked {
        iteratedState = _currState * 15485863;
        iteratedState = (iteratedState * iteratedState * iterate
    }
}
```

This means that after the first pass through `iteratePRNG` we only have 2038074743 different possibilities which will determine the rest of the distortion.

The original `_characterModifier` is first used to set to `uint256 numAbove = (_characterModifier % 7) + 1;`. The remaining combinations are therefore  $1.38e24 / 7 = 1.98e+23$ , which is vastly greater than 2038074743.

`characterToUnicodeBytes` can thus only return  $7 * 2038074743 = 14266523201$  different distortions per letter, which is only a tiny fraction of the actual number of distortions.

Note that it is similarly an issue that the range of `_characterModifier` is only `0..255`, which issue is reportedly separately.



## Recommended Mitigation Steps

`iteratePRNG` must have a range of at least  $1.98e+23$  different values.

[OpenCoreCH \(Canto Identity\) acknowledged](#)



**[M-03]** `characterModifier` is `uint8` but encodes  $1.38e24$  different Zalgo distortions

Submitted by [d3e4](#)

<https://github.com/code-423n4/2023-03-canto-identity/blob/077372297fc419ea7688ab62cc3fd4e8f4e24e66/canto-namespace-protocol/src/Tray.sol#L163>

<https://github.com/code-423n4/2023-03-canto-identity/blob/077372297fc419ea7688ab62cc3fd4e8f4e24e66/canto-namespace-protocol/src/Tray.sol#L264-L268>

<https://github.com/code-423n4/2023-03-canto-identity/blob/077372297fc419ea7688ab62cc3fd4e8f4e24e66/canto-namespace-protocol/src/Utils.sol#L136-L174>

Only 256 Zalgo distortions are possible, which is a miniscule fraction of the actual combinations possible.



## Proof of Concept

A Zalgo tile is defined by a letter and a modification consisting of a combination of characters above, over and below the letter. The modification consists of a selection of 1 up to 7 characters from a set of 46 characters above the letter, 0 or 1 from 5 over the letter, and 1 to 7 from 47 below the letter. Thus there is a total of  $(46^1 + 46^2 + \dots + 46^7) * (5^0 + 5^1) * (47^1 + 47^2 + \dots + 47^7) = 1.38e24$  different Zalgo modifications possible per letter.

However, the distortions are uniquely determined by `characterModifier` which is a `uint8`. So only 256 different distortions per letter are possible.



When a tray is bought, [\\_drawing](#) **sets the tile**,

```
function buy(uint256 _amount) external {
// ...
    trayTiledata[j] = _drawing(uint256(lastHash));
// ...
}
```

and, if luck would have it that it is a Zalgo tile, [characterModifier](#) **is set to a pseudo-random** [uint8](#), i.e. in the range 0..255.

```
function _drawing(uint256 _seed) private pure returns (TileData
// ...
    if (tileData.fontClass == 7) {
        // Set seed for Zalgo to ensure same characters
        uint256 zalgoSeed = Uutils.iteratePRNG(_seed);
        tileData.characterModifier = uint8(zalgoSeed % 2
// ...
    }
```

This `characterModifier` uniquely determines the distortion in `characterToUnicodeBytes` at [L136-L174](#):

```
} else if (_fontClass == 7) {
    // Zalgo
    uint8 asciiStartingIndex = 97;
    uint256 numAbove = (_characterModifier % 7) + 1;
    // We do not reuse the same seed for the following generatic
    _characterModifier = iteratePRNG(_characterModifier);
    uint256 numMiddle = _characterModifier % 2;
    _characterModifier = iteratePRNG(_characterModifier);
    uint256 numBelow = (_characterModifier % 7) + 1;
    bytes memory character = abi.encodePacked(bytes1(asciiStarti
    for (uint256 i; i < numAbove; ++i) {
        _characterModifier = iteratePRNG(_characterModifier);
        uint256 characterIndex = (_characterModifier % ZALGO_NUM
        character = abi.encodePacked(
            character,
            ZALGO_ABOVE_LETTER[characterIndex],
```

```

        ZALGO_ABOVE_LETTER[characterIndex + 1]
    );
}
for (uint256 i; i < numMiddle; ++i) {
    _characterModifier = iteratePRNG(_characterModifier);
    uint256 characterIndex = (_characterModifier % ZALGO_NUM
    character = abi.encodePacked(
        character,
        ZALGO_OVER_LETTER[characterIndex],
        ZALGO_OVER_LETTER[characterIndex + 1]
    );
}
for (uint256 i; i < numBelow; ++i) {
    _characterModifier = iteratePRNG(_characterModifier);
    uint256 characterIndex = (_characterModifier % ZALGO_NUM
    character = abi.encodePacked(
        character,
        ZALGO_BELOW_LETTER[characterIndex],
        ZALGO_BELOW_LETTER[characterIndex + 1]
    );
}
return character;
} else {

```

Note that there is also a similar bottleneck issue with `iteratePRNG`, which is reported as a separate issue.



## Recommended Mitigation Steps

Let `characterModifier` have a range of at least `0..1.38e24`. Since it therefore cannot be a `uint8`, simply let it be a `uint256` (or at least a `uint88`).

## [OpenCoreCH \(Canto Identity\) acknowledged and commented:](#)

Addressing [M-02 \(issue 282\)](#) and [M-03 \(issue 277\)](#) here because they are very similar: Both issues are technically true and well written, but 256 different distortions are more than enough in my opinion and there is no need to have `1.38e24`.

## [d3e4 \(warden\) commented:](#)

@OpenCoreCH - It's of course up to you what you eventually implement, but I would just like to offer my perspective on why you might actually want to make the corrections suggested in #277 and #282 .

256 distortions may seem plenty, but there are already 98 combinations just in terms of the number of diacritics above, in the middle, and below (727). Currently five such combinations are not even generated at all, viz. (2,1,7), (3,0,6), (3,1,1), (5,0,6), (6,1,5), where the combinations are represented as (numAbove, numMiddle, numBelow).

Even if all possible distortions were reachable, a user couldn't feasibly obtain a specific distortion of his choosing, but he might desire some class of distortions. If this class is any of the above missing combinations of numbers of diacritics he evidently cannot obtain it. But even if 14266523201 distortions are available ( #282 ) the chances that these overlap with his own desired class is minuscule, as these are only 0.0000000000001% of all possible Zalgo distortions promised by the parameters.

And with only 256 distortions they will have to be reused. Different users will share identical distortions, and there may even be duplicates in the same tray. This would be exceedingly unlikely if the full range of distortions were available.

If the goal is simply to offer only 256 different distortions, then the protocol currently achieves this. (But in this case I wonder if it wouldn't be more gas efficient to simply hardcode these and pick one randomly instead of generating them in the current complicated manner.) But the protocol gives the impression (if not promise?) of offering a distortion synthesised from a range of diacritics, which might leave the user feeling cheated. For example, none of the Zalgos in the docs are available.



## [M-04] Bio Protocol - tokenURI JSON injection

Submitted by [reassor](#), also found by [Wander](#), [ulqiorra](#), [ulqiorra](#), [mjmoonwalker](#), [reassor](#), [dingo2077](#), [JerryOx](#), [ParadOx](#), [adriro](#), [Kek](#), [Kek](#), [Respx](#), [Viktor\\_Cortess](#), [Viktor\\_Cortess](#), [Haipls](#), [J4de](#), [Haipls](#), [Englave](#), [bin2chen](#), [MiniGlome](#), [djxploit](#), [jasonxiale](#), and [rokso](#)

The Bio Protocol allows users to mint Bio NFTs that represent user's bio. Once NFT is minted anyone can trigger `tokenURI` to retrieve JSON data with the bio and generated svg image. Example JSON content (decoded from Base64):

```
{"name": "Bio #1", "description": "Test", "image": "data:image/svg+xml;base64,PHN0aGVudGU="}
```

The issue is that function `Bio.tokenURI` is vulnerable to JSON injection and it is possible to inject `"` character into bio completely altering the integrity of JSON data. User sends following data as `bio`:

```
Test\", \"name\": \"Bio #999999999
```

This will result in following JSON data (new `name` key was injected):

```
{"name": "Bio #1", "description": "Test", "name": "Bio #999999999", "image": "data:image/svg+xml;base64,PHN0aGVudGU="}
```

Depending on the usage of Bio Protocol by frontend or 3rd party applications this will have serious impact on the security. Some examples of the attacks that can be carried:

1. Attacker might create a Bio NFT that mimics copies the name of different Bio NFT.
2. Attacker might change the generated svg `image` to completely different one.
3. Attacker might point `image` to external `URL`.
4. Attacker might trigger cross-site scripting attacks if the data from JSON is not properly handled.



Proof of Concept

<https://github.com/code-423n4/2023-03-canto-identity/blob/077372297fc419ea7688ab62cc3fd4e8f4e24e66/canto-bio-protocol/src/Bio.sol#L103-L115>



## Tools Used

Manual Review / Foundry / VSCode



## Recommended Mitigation Steps

It is recommended to properly encode user's bio to make sure it cannot alter returned JSON data.

### [OpenCoreCH \(Canto Identity\) disagreed with severity and commented:](#)

Good point that will be fixed, have some doubts about the severity. The attacks that are mentioned on the frontend in various issues (XSS / external resources) should be handled there in any case because the payload of a NFT should not be considered trusted. And if a user manipulates his own NFT, the impact is limited to him.

### [Oxleatwood \(judge\) commented:](#)

@OpenCoreCH - It doesn't seem like this sort of `tokenURI()` sanitisation can be made within the contract but instead must be handled by front-ends right?

### [OpenCoreCH \(Canto Identity\) commented:](#)

@Oxleatwood - I implemented some sanitisation in the mean time (escaping all characters that need to be escaped for the JSON / SVG) using Solady's [escape functions](#). But that's mainly to ensure that the produced JSON / SVG will always be valid (and the user does not accidentally produce an invalid one by choosing some characters in his bio). I think perfect sanitisation on-chain is not feasible in practice and the frontends should generally treat this payload as untrusted and sanitise it. Otherwise, you could also actively deploy malicious NFTs to e.g. target NFT marketplace users.

### [Oxleatwood \(judge\) decreased severity to Medium and commented:](#)

@OpenCoreCH - Completely agree. I think medium severity would make more sense then.



## [M-05] If the underlying NFT is burned, getPFP may return incorrect results

Submitted by [cccZ](#)

ProfilePicture.getPFP will return information about the underlying NFT and when `addressRegistry.getAddress(cidNFTID) != ERC721(nftContract).ownerOf(nftID)`, it will return 0.

But if the underlying NFT is burned, getPFP may return incorrect information

```
function getPFP(uint256 _pfpID) public view returns (address) {
    if (_ownerOf[_pfpID] == address(0)) revert TokenNotMinted();
    ProfilePictureData storage pictureData = pfp[_pfpID];
    nftContract = pictureData.nftContract;
    nftID = pictureData.nftID;
    uint256 cidNFTID = cidNFT.getPrimaryCIDNFT(subprotocolName);
    IAddressRegistry addressRegistry = cidNFT.addressRegistry();
    if (cidNFTID == 0 || addressRegistry.getAddress(cidNFTID) == address(0)) {
        nftContract = address(0);
        nftID = 0; // Strictly not needed because nftContract is already 0
    }
}
```

Consider the following scenario.

1. Alice mints a ProfilePicture NFT (pfp NFT) with the underlying NFT, and later mints a CidNFT by transferring the pfp NFT to the CidNFT contract.

**Noting that alice does not call AddressRegistry.register to register the CidNFT.**

Since getAddress returns 0, ProfilePicture.getPFP will also return 0.

```
function getAddress(uint256 _cidNFTID) external view returns (address) {
    user = ERC721(cidNFT).ownerOf(_cidNFTID);
    if (_cidNFTID != cidNFTs[user]) {
        // User owns CID NFT, but has not registered it
        user = address(0);
    }
}
```

2. Alice sends the underlying NFT to Bob, who does what Alice did before.
3. Bob sold the underlying NFT to Charlie and Charlie burned it for some reason, now if Alice or Bob calls `ProfilePicture.getPFP()`, it will return the contract address and id of the underlying NFT since `getAddress() == ownerOf() == address(0)`.

When integrating with other projects, there may be bugs because `getPFP` returns an incorrect result.



## Proof of Concept

<https://github.com/code-423n4/2023-03-canto-identity/blob/077372297fc419ea7688ab62cc3fd4e8f4e24e66/canto-pfp-protocol/src/ProfilePicture.sol#L94-L105>

<https://github.com/code-423n4/2023-03-canto-identity/blob/077372297fc419ea7688ab62cc3fd4e8f4e24e66/canto-identity-protocol/src/AddressRegistry.sol#L86-L92>

<https://github.com/code-423n4/2023-03-canto-identity/blob/077372297fc419ea7688ab62cc3fd4e8f4e24e66/canto-identity-protocol/src/AddressRegistry.sol#L86-L92>

<https://github.com/code-423n4/2023-03-canto-identity/blob/077372297fc419ea7688ab62cc3fd4e8f4e24e66/canto-identity-protocol/src/AddressRegistry.sol#L86-L92>



## Recommended Mitigation Steps

Change to

```
function getPFP(uint256 _pfpID) public view returns (address
    if (_ownerOf[_pfpID] == address(0)) revert TokenNotMinted;
    ProfilePictureData storage pictureData = pfp[_pfpID];
    nftContract = pictureData.nftContract;
    nftID = pictureData.nftID;
    uint256 cidNFTID = cidNFT.getPrimaryCIDNFT(subprotocolName);
    IAddressRegistry addressRegistry = cidNFT.addressRegistry();
-   if (cidNFTID == 0 || addressRegistry.getAddress(cidNFTID) == address(0))
+   if (cidNFTID == 0 || ERC721(nftContract).ownerOf(nftID) == address(0))
        nftContract = address(0);
        nftID = 0; // Strictly not needed because nftContract is not zero
    }
}
```

[OpenCoreCH \(Canto Identity\) confirmed and commented:](#)

Very interesting edge case, enjoyed reading it! For an ERC721 compliant NFT this is not the case, because the standard explicitly says:

```
/// @dev NFTs assigned to zero address are considered invalid
/// about them do throw.
```

Therefore, the existing `ownerOf` will throw.

However, in my experience there are some NFTs that do not adhere strictly to ERC721 when it comes to this. Because of that, I'll still implement a fix for this.



## [M-06] Incorrect emoji displaying

Submitted by [glcanvas](#), also found by [fsOc](#) and [adriro](#)

Possible to split bio incorrectly, this leads to incorrect display of generated image. This image will be impossible to modify. This might lead to integration problems.



## Proof of Concept

Let's consider `tokenURI` function of the Bio contract.

Here implemented corner case for some emojis, but current implementation doesn't handle all cases. I suppose handling all cases is redundant, erroneously, complicated and unnecessary.

Let's consider part of `tokenURI` code in python, this part just rewritten on Python part of Solidity code:

```
import binascii

def sol_func_same(bio):
    bioTextBytes = str.encode(bio, "utf-8")
    lengthInBytes = len(bioTextBytes)
    lines = (lengthInBytes - 1) // 40 + 1
    strLines = [None for _ in range(lines)]
    prevByteWasContinuation = False
    insertedLines = 0
    bytesLines = []
    bytesOffset = 0
    for i in range(0, lengthInBytes):
        character = bioTextBytes[i]
        bytesLines.append(character)
```







1. Add view function which is responsible to create svg image. In tokenURI call this function to generate svg result.

Like this:

```
function generateSvg(string memory bioText) public view returns (
    bytes memory bioTextBytes = bytes(bioText);
    uint lengthInBytes = bioTextBytes.length;
    // Insert a new line after 40 characters, taking into ac
    uint lines = (lengthInBytes - 1) / 40 + 1;
...
...
...
    return string(abi.encodePacked("data:application/json;base64,"
    bioTextBytes, "data:image/svg+xml;utf8,"
    generateSvg(bioText)));
}
```

2. Add possibility to manual split by lines. i.e. with bio additionally store offsets in bytes, by which lines must be split.

```
function mint(string calldata _bio, uint256[] calldata byteSplit
...
}

function generateSvg(string memory bioText, uint256[] memory byteSplit
...
bytes memory bioTextBytes = bytes(bioText);
in for loop:
    slice = bioTextBytes[i:i+1]
    write slice to svg
}
```

[OpenCoreCH \(Canto Identity\) confirmed](#)



[M-07] Users can end up buying and paying for a different Tray than the one they were trying to acquire

Submitted by [juancito](#), also found by [luxartvinsec](#), [leopoldjoy](#), [popular00](#), [pipoca](#), [Chom](#), [reassor](#), [iging](#), [adriro](#), [J4de](#), and [Ruhum](#)

Trays and their tiles are the building blocks for fusing a Namespace NFT. The tiles in a tray are generated based on a hash of the previous mint.

Users will need specific tiles to be able to create the name they want, so it is of utter importance for them to get the trays they want with their corresponding tiles. The users should be able to precompute this, [as the documentation states](#):

“The 7 tiles per tray are then generated according to a deterministic algorithm. A user can therefore precompute which trays he will get.”



## Impact

Users can pay the corresponding `$NOTE` tokens to buy a tray with some specific tiles, but end up obtaining a completely different one with tiles they don't expect.

This can happen due to normal usage of the platform, with some user sending a transaction with more gas. Or by a malicious user frontrunning the transaction.

On any case the user ends up with an NFT he didn't want instead of reverting the transaction, and saving their `$NOTE` tokens.



## Proof of Concept

There is no check on the `buy()` function to make sure that the user will actually buy the tray he wanted, thus possibly obtaining another one.

The tiles are calculated based on a hash. That hash is generated each time a new mint is produced, and it is based on the last one.

If the transaction is frontrun, the user will get a new hash, and thus new tiles data for his tray.

File: `canto-namespace-protocol/src/Tray.sol`

```
148:      /// @notice Buy a specifiable amount of trays
149:      /// @param _amount Amount of trays to buy
150:      function buy(uint256 _amount) external {
151:          uint256 startingTrayId = _nextTokenId();
152:          if (prelaunchMinted == type(uint256).max) {
153:              // Still in prelaunch phase
```

```

154:         if (msg.sender != owner) revert OnlyOwnerCanMint();
155:         if (startingTrayId + _amount - 1 > PRE_LAUNCH_MINT_ID) {
156:             } else {
157:                 SafeTransferLib.safeTransferFrom(note, msg.sender, 1);
158:             }
159:         for (uint256 i; i < _amount; ++i) {
160:             TileData[TILES_PER_TRAY] memory trayTiledata;
161:             for (uint256 j; j < TILES_PER_TRAY; ++j) {
162:                 lastHash = keccak256(abi.encode(lastHash));
163:                 trayTiledata[j] = _drawing(uint256(lastHash));
164:             }
165:             tiles[startingTrayId + i] = trayTiledata;
166:         }
167:         _mint(msg.sender, _amount); // We do not use _safeMint
168:     }

```

## [Link to code](#)



## Recommended Mitigation Steps

Check the hash for the last minted Tray NFT, to make sure it is the same the user used to precalculate the next tiles.

On the case the transaction is frontrunned, the expected hash will be different and the transaction will revert, saving the user from spending the \$NOTE tokens.

```

// File: canto-namespace-protocol/src/Tray.sol

/// @notice Buy a specifiable amount of trays
/// @param _amount Amount of trays to buy
- function buy(uint256 _amount) external {
+ function buy(uint256 _amount, bytes32 _lastHash) external {
+     require(_lastHash == lastHash, "Hashes must be the same");
+     uint256 startingTrayId = _nextTokenId();
+     if (prelaunchMinted == type(uint256).max) {
+         // Still in prelaunch phase
+         if (msg.sender != owner) revert OnlyOwnerCanMintPreLaunch();
+         if (startingTrayId + _amount - 1 > PRE_LAUNCH_MINT_ID) {
+             } else {
+                 SafeTransferLib.safeTransferFrom(note, msg.sender, 1);
+             }
+         for (uint256 i; i < _amount; ++i) {
+             TileData[TILES_PER_TRAY] memory trayTiledata;

```

```

        for (uint256 j; j < TILES_PER_TRAY; ++j) {
            lastHash = keccak256(abi.encode(lastHash));
            trayTiledata[j] = _drawing(uint256(lastHash));
        }
        tiles[startingTrayId + i] = trayTiledata;
    }
    _mint(msg.sender, _amount); // We do not use _safeMint c
}

```

## [OpenCoreCH \(Canto Identity\) acknowledged](#)

## [Oxleastwood \(judge\) commented:](#)

While this could be treated as a duplicate of [#121](#), I like how the warden has provided a remediation which would protect users from being front-run and receiving an unexpected ID.



## [M-08] Bio lines will overflow the buffer for repeated “continuation” characters

*Submitted by* [adriro](#)

The Bio `tokenURI` function splits biography text into lines. The algorithm will take into account certain “continuation” characters to prevent splitting the line in the middle of these characters and keep accumulating those in the current line buffer (`bytesLines`):

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-bio-protocol/src/Bio.sol#L60-L95>

```

if ((i > 0 && (i + 1) % 40 == 0) || prevByteWasContinuation || i
    bytes1 nextCharacter;
    if (i != lengthInBytes - 1) {
        nextCharacter = bioTextBytes[i + 1];
    }
    if (nextCharacter & 0xC0 == 0x80) {
        // Unicode continuation byte, top two bits are 10
        prevByteWasContinuation = true;
    } else {

```

```

// Do not split when the prev. or next character is a zero
// Furthermore, do not split when next character is skin tone
if (
    // Note that we do not need to check i < lengthInBytes
    (nextCharacter == 0xE2 && bioTextBytes[i + 2] == 0x84 &&
    (nextCharacter == 0xF0 &&
        bioTextBytes[i + 2] == 0x9F &&
        bioTextBytes[i + 3] == 0x8F &&
        uint8(bioTextBytes[i + 4]) >= 187 &&
        uint8(bioTextBytes[i + 4]) <= 191) ||
    (i >= 2 &&
        bioTextBytes[i - 2] == 0xE2 &&
        bioTextBytes[i - 1] == 0x80 &&
        bioTextBytes[i] == 0x8D)
) {
    prevByteWasContinuation = true;
    continue;
}
assembly {
    mstore(bytesLines, bytesOffset)
}
strLines[insertedLines++] = string(bytesLines);
bytesLines = new bytes(80);
prevByteWasContinuation = false;
bytesOffset = 0;
}
}

```

However, if the unicode string is a sequence of these continuation characters (which is a valid UTF8 string) the line buffer (which is 80 bytes) will eventually overflow and will revert the transaction due to an index of out bounds exception.



## Proof of Concept

In the following test we use a string with 21 [U+1F3FE](#) characters to overflow the line buffer and revert the query to `tokenURI`.

Note: the snippet shows only the relevant code for the test. Full test file can be found [here](#).

```

function test_Bio_tokenURI_LineBufferOverflow() public {
    // This is a skin tone codepoint ("f09f8f8e") repeated 21 ti

```

```
string memory text = unicode"aaaaaaaaaaaaaaaaaaaaaaaa";
bio.mint(string(text));
vm.expectRevert();
bio.tokenURI(1);
}
```



## Recommendation

Change to a new line when the current line buffer is full.

### OpenCoreCH (Canto Identity) confirmed and commented:

Extreme edge case with a string that’s technically valid, but semantically meaningless. But it’s technically true.



## [M-09] Bio NFT incorrectly breaks SVG lines and doesn’t support more than 120 characters effectively

Submitted by [volodya](#), also found by [Haipls](#)

According to the docs

“Any user can mint a Bio NFT by calling Bio.mint and passing his biography. It needs to be shorter than 200 characters.”

Let’s take two strings and pass them to create an NFT. The first one is 200 characters long, and the second one is 120 characters long.

aaaWaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaWaaaWaaaaaaaaaaaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaWaaaW

aaaWaaa  
aaaaaaaaaaaaaWaaaW

This is how they will look like. As you can see they look identical.





Next, let's take this text for which we create nft. I took it from a test and double.

012345678901234567890123456789012345678 🧑🧑🧑

012345678901234567890123456789012345678 🧑🧑🧑

Here is on the left how it looks now vs how it suppose to be. As you can see the line breaking doesn't work. I did enlarge viewBox so you can see the difference.



The problem is in this part of the code, where `(i > 0 && (i + 1) % 40 == 0)` doesn't handle properly because you want to include emojis, so length will be more than 40 (`40 + length(emoji)`)

```
canto-bio-protocol/src/Bio.sol#L56
    for (uint i; i < lengthInBytes; ++i) {
        bytes1 character = bioTextBytes[i];
        bytesLines[bytesOffset] = character;
        bytesOffset++;
        if ((i > 0 && (i + 1) % 40 == 0) || prevByteWasConti
            bytes1 nextCharacter;
```

[canto-bio-protocol/src/Bio.sol#L56](#)

Lastly, the NFT doesn't center-align text, but I believe it should. I took text from a test and on the left is how it currently appears, while on the right is how I think it should be.



Here is the code. dy doesn't apply correctly; it should be 0 for the first line.

```
canto-bio-protocol/src/Bio.sol#L104
    for (uint i; i < lines; ++i) {
        text = string.concat(text, '<tsan x="50%" dy="20">'
    }
```

[canto-bio-protocol/src/Bio.sol#L104](#)

## Recommended Mitigation Steps

Enlarge viewBox so it will support 200 length or restrict to 120 characters.

Here is a complete code with correct line breaking and center text. I'm sorry that I didn't add `diff` to code because there will be too many lines. It does pass tests and fix current issues.

```
function tokenURI(uint256 _id) public view override returns
    if (_ownerOf[_id] == address(0)) revert TokenNotMinted(_
    string memory bioText = bio[_id];
    bytes memory bioTextBytes = bytes(bioText);
    uint lengthInBytes = bioTextBytes.length;
    // Insert a new line after 40 characters, taking into ac
    uint lines = (lengthInBytes - 1) / 40 + 1;
    string[] memory strLines = new string[](lines);
    bool prevByteWasContinuation;
    uint256 insertedLines;
    // Because we do not split on zero-width joiners, line i
    bytes memory bytesLines = new bytes(80);
    uint bytesOffset;
    uint j;
    for (uint i; i < lengthInBytes; ++i) {
        bytesLines[bytesOffset] = bytes1(bioTextBytes[i]);
        bytesOffset++;
        j+=1;
        if ((j>=40) || prevByteWasContinuation || i == lengt
            bytes1 nextCharacter;
            if (i != lengthInBytes - 1) {
                nextCharacter = bioTextBytes[i + 1];
            }
            if (nextCharacter & 0xC0 == 0x80) {
                // Unicode continuation byte, top two bits a
                prevByteWasContinuation = true;
                continue;
            } else {
                // Do not split when the prev. or next chara
                // Furthermore, do not split when next chara
                if (
                    // Note that we do not need to check i <
                    (nextCharacter == 0xE2 && bioTextBytes[i
                    (nextCharacter == 0xF0 &&
                        bioTextBytes[i + 2] == 0x9F &&
                        bioTextBytes[i + 3] == 0x8F &&
                        uint8(bioTextBytes[i + 4]) >= 187 &&
```

```

        uint8(bioTextBytes[i + 4]) <= 191) |
        (i >= 2 &&
        bioTextBytes[i - 2] == 0xE2 &&
        bioTextBytes[i - 1] == 0x80 &&
        bioTextBytes[i] == 0x8D)
    ) {
        prevByteWasContinuation = true;
        continue;
    }
}

assembly {
    mstore(bytesLines, bytesOffset)
}
strLines[insertedLines++] = string(bytesLines);
bytesLines = new bytes(80);
prevByteWasContinuation = false;
bytesOffset = 0;
j=0;
}
}
string
    memory svg = '<svg xmlns="http://www.w3.org/2000/svg'
string memory text = '<text x="50%" y="50%" dominant-bas
text = string.concat(text, '<tspan x="50%" dy="0">', str
for (uint i=1; i < lines; ++i) {
    text = string.concat(text, '<tspan x="50%" dy="20">'
}
string memory json = Base64.encode(
    bytes(
        string.concat(
            '{"name": "Bio #',
            LibString.toString(_id),
            '", "description": "',
            bioText,
            '", "image": "data:image/svg+xml;base64,',
            Base64.encode(bytes(string.concat(svg, text,
            '"})'
        )
    )
);
return string(abi.encodePacked("data:application/json;base
}

```

Yeah the line break generally happened approx. after 40 bytes, which results in somewhat weird results for these huge emojis. Should be fixed when the tokenURI is refactored, see [#87](#).



## Low Risk and Non-Critical Issues

For this contest, 38 reports were submitted by wardens detailing low risk and non-critical issues. The [report highlighted below](#) by Sathish9098 received the top score from the judge.

*The following wardens also submitted reports: [joestakey](#), [tnevler](#), [libratus](#), [atharvasama](#), [Kresh](#), [nadin](#), [alejandrocovrr](#), [Aymen0909](#), [Oxdaydream](#), [lukris02](#), [Udsen](#), [Awesome](#), [OxSmartContract](#), [luxartvinsec](#), [Deathstore](#), [slvDev](#), [cryptonue](#), [OxAgro](#), [Stryder](#), [Matin](#), [descharre](#), [reassor](#), [IceBear](#), [scokaf](#), [JerryOx](#), [Diana](#), [igingy](#), [T1MOH](#), [adriro](#), [shark](#), [jack](#), [codeslide](#), [Bauchibred](#), [Rolezn](#), [nasri136](#), [BRONZEDISC](#), and [Oxnev](#).*



## [L-01] AVOID USING TX.ORIGIN

tx.origin is a global variable in Solidity that returns the address of the account that sent the transaction.

Using the variable could make a contract vulnerable if an authorized account calls a malicious contract. You can impersonate a user using a third party contract.

FILE : 2023-03-canto-identity/canto-pfp-protocol/src/ProfilePicture.sol

```
63:  turnstile.register(tx.origin);
```

### [ProfilePicture.sol#L63](#)

FILE : 2023-03-canto-identity/canto-bio-protocol/src/Bio.sol

```
36:  turnstile.register(tx.origin);
```

### [Bio.sol#L36](#)

FILE : 2023-03-canto-identity/canto-namespace-protocol/src/Namespace.sol



## [L-02] LOSS OF PRECISION DUE TO ROUNDING

FILE : 2023-03-canto-identity/canto-bio-protocol/src/Bio.sol

```
49:  uint lines = (lengthInBytes - 1) / 40 + 1;
```

### [Bio.sol#L49](#)

FILE : 2023-03-canto-identity/canto-namespace-protocol/src/Utils.sol

```
224:  uint256 dx = 400 / (_tiles.length + 1);
```

### [Utils.sol#L224](#)

### [Tray.sol#L259-L263](#)



## [L-03] Consider using OpenZeppelin's SafeCast library to prevent unexpected overflows when casting from various type int/uint values

FILE : 2023-03-canto-identity/canto-bio-protocol/src/Bio.sol

```
77:  uint8(bioTextBytes[i + 4]) >= 187 &&  
78:  uint8(bioTextBytes[i + 4]) <= 191) ||
```

FILE : 2023-03-canto-identity/canto-namespace-protocol/src/Tray.sol

```
163:  trayTiledData[j] = _drawing(uint256(lastHash));
```

### [Tray.sol#L163](#)

```
250 :   tileData.characterIndex = uint16(charRandValue % NUM_CHAF
```

### [src/Tray.sol#L250](#)

```
252:   tileData.characterIndex = uint16(charRandValue % NUM_CHARS_
```

### [Tray.sol#L252](#)

```
    tileData.fontClass = 3 + uint8((res - 80) / 8);  
    } else if (res < 104) {  
        tileData.fontClass = 5 + uint8((res - 96) / 4);  
    } else if (res < 108) {  
        tileData.fontClass = 7 + uint8((res - 104) / 2);
```

### [Tray.sol#L259-L263](#)

```
267:   tileData.characterModifier = uint8(zalgoSeed % 256);
```

### [Tray.sol#L267](#)

FILE : 2023-03-canto-identity/canto-namespace-protocol/src/Utils.sol

```
272:   uint8 lastByteVal = uint8(_startingSequence[3]);
```

```
278:   _startingSequence[2] = bytes1(uint8(_startingSequence[2]) +
```

### [Utils.sol#L278](#)



## Recommended Mitigation Steps

Consider using OpenZeppelin's SafeCast library to prevent unexpected overflows when casting from uint256.



## [L-04] Prevent division by 0

On several locations in the code precautions are not being taken for not dividing by 0, this will revert the code. These functions can be called with 0 value in the input, this value is not checked for being bigger than 0, that means in some scenarios this can potentially trigger a division by zero

FILE : 2023-03-canto-identity/canto-namespace-protocol/src/Utils.sol

```
224: uint256 dx = 400 / (_tiles.length + 1);
```

### [Utils.sol#L224](#)



#### Recommended Mitigation:

Need to check the `(_tiles.length + 1)` value is not 0



## [N-01] Named imports can be used

It's possible to name the imports to improve code readability.

E.g. import “@openzeppelin/contracts/token/ERC20/IERC20.sol”; can be rewritten as import {IERC20} from “import

“@openzeppelin/contracts/token/ERC20/Utils/SafeERC20.sol”

FILE : 2023-03-canto-identity/canto-pfp-protocol/src/ProfilePicture.sol

```
import "../interface/Turnstile.sol";  
import "../interface/ICidNFT.sol";
```

### [ProfilePicture.sol#L5-L6](#)

FILE : 2023-03-canto-identity/canto-bio-protocol/src/Bio.sol

```
import "../interface/Turnstile.sol";
```

### [Bio.sol#L7](#)

FILE : 2023-03-canto-identity/canto-namespace-protocol/src/Namespace.sol

```
import "../Tray.sol";
import "../Utils.sol";
import "../interface/Turnstile.sol";
```

### [Namespace.sol#L7-L9](#)

FILE : 2023-03-canto-identity/canto-namespace-protocol/src/Tray.sol

```
import "../Utils.sol";
import "../interface/Turnstile.sol";
```

### [Tray.sol#L10-L11](#)

FILE : 2023-03-canto-identity/canto-namespace-protocol/src/Utils.sol

```
4: import "../Tray.sol";
```

### [Utils.sol#L4](#)



#### Recommendations

import {IERC20} from “import  
“@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol”



## [N-02] AVOID HARDCODED VALUES

It is not good practice to hardcode values, but if you are dealing with addresses much less, these can change between implementations, networks or projects, so it is convenient to remove these values from the source code

FILE : 2023-03-canto-identity/canto-pfp-protocol/src/ProfilePicture.sol

```
62: Turnstile turnstile = Turnstile(0xEcf044C5B4b867CFda001101c6
```



## [ProfilePicture.sol#L62](#)

```
60:     if (block.chainid == 7700) {
```

## [ProfilePicture.sol#L60](#)

FILE : 2023-03-canto-identity/canto-bio-protocol/src/Bio.sol

```
33: if (block.chainid == 7700) {  
35: Turnstile turnstile = Turnstile(0xEcf044C5B4b867CFda001101c6
```

## [Bio.sol#L33-L35](#)



### [N-03] Include return parameters in NatSpec comments

@return tag is missing.

## [ProfilePicture.sol#L67-L70](#)

## [Bio.sol#L40-L43](#)

## [Namespace.sol#L88-L90](#)

## [Utils.sol#L69-L73](#)

## [Utils.sol#L219-L222](#)

## [Utils.sol#L263-L267](#)



### Recommendations

Add @return tag whenever function returns any value.



### [N-04] ADD PARAMETER TO EVENT-EMIT

Some event-emit description doesn't have parameter. Add parameter for front-end website or client app, they can has that something has happened on the blockchain.

```
80: event PrelaunchEnded();
```

### [Tray.sol#L80](#)



## [N-05] NOT USING THE NAMED RETURN VARIABLES ANYWHERE IN THE FUNCTION IS CONFUSING

Consider changing the variable to be an unnamed one.

### [ProfilePicture.sol#L94](#)

### [Utils.sol#L256](#)

### [Tray.sol#L245](#)

### [Tray.sol#L203](#)

### [Tray.sol#L195](#)



## [N-06] Interchangeable usage of uint and uint256

Context: All Contracts

Consider using only one approach throughout the codebase, e.g. only uint or only uint256

### [Bio.sol#L52-L55](#)



## Recommendations

Only uint or only uint256.



## [N-07] Immutables should be in uppercase

FILE : 2023-03-canto-identity/canto-pfp-protocol/src/ProfilePicture.sol

```
14: ICidNFT private immutable cidNFT;
```

## [ProfilePicture.sol#L14](#)

FILE : 2023-03-canto-identity/canto-namespace-protocol/src/Namespace.sol

```
17:  Tray public immutable tray;
```

## [Namespace.sol#L17](#)

FILE : 2023-03-canto-identity/canto-namespace-protocol/src/Tray.sol

```
37:  uint256 public immutable trayPrice;
```

## [Tray.sol#L37](#)

```
50:  address public immutable namespaceNFT;
```

## [Tray.sol#L50](#)



**[N-08] Move IF/validation statements to the top of the function when validating input parameters**

FILE : 2023-03-canto-identity/canto-pfp-protocol/src/ProfilePicture.sol

```
function mint(address _nftContract, uint256 _nftID) external {
    uint256 tokenId = ++numMinted;
    if (ERC721(_nftContract).ownerOf(_nftID) != msg.sender)
        revert PFPNotOwnedByCaller(msg.sender, _nftContract, _nftID);
    ProfilePictureData storage pictureData = pfp[tokenId];
    pictureData.nftContract = _nftContract;
```

## [ProfilePicture.sol#L79-L88](#)



**[N-09] FOR FUNCTIONS, FOLLOW SOLIDITY STANDARD NAMING CONVENTIONS**

Internal and private functions: the mixedCase format starting with an underscore  
(`_mixedCase` starting with an underscore)

FILE : 2023-03-canto-identity/canto-namespace-protocol/src/Utils.sol

[Utils.sol#L69-L77](#)

[Utils.sol#L222](#)

[Utils.sol#L256](#)



[N-10] Don't use numbers without explanation, it's not a good code practice. It's possible to use constants.

FILE : 2023-03-canto-identity/canto-namespace-protocol/src/Utils.sol

```
258: iteratedState = _currState * 15485863;  
259: iteratedState = (iteratedState * iteratedState * iteratedState)
```

[Utils.sol#L258-L259](#)



[N-11] Use underscores for number literals

FILE : 2023-03-canto-identity/canto-namespace-protocol/src/Utils.sol

```
258: iteratedState = _currState * 15485863;  
259: iteratedState = (iteratedState * iteratedState * iteratedState)
```

[Utils.sol#L258-L259](#)



[N-12] NO SAME VALUE INPUT CONTROL

FILE : 2023-03-canto-identity/canto-namespace-protocol/src/Namespace.sol

```
function changeNoteAddress(address _newNoteAddress) external onlyOwner  
    address currentNoteAddress = address(note);
```

```

        note = ERC20(_newNoteAddress);
        emit NoteAddressUpdate(currentNoteAddress, _newNoteAddress);
    }

    /// @notice Change the revenue address
    /// @param _newRevenueAddress New address to use
    function changeRevenueAddress(address _newRevenueAddress) external {
        address currentRevenueAddress = revenueAddress;
        revenueAddress = _newRevenueAddress;
        emit RevenueAddressUpdated(currentRevenueAddress, _newRevenueAddress);
    }

```

## [Namespace.sol#L196-L208](#)



### [N-13] Need Fuzzing test

FILE : 2023-03-canto-identity/canto-namespace-protocol/src/Utils.sol

```

257: unchecked {

```

## [Utils.sol#L257](#)



### Recommendation

Should use fuzzing test like Echidna.

As Alberto Cuesta Canada said:

“Fuzzing is not easy, the tools are rough, and the math is hard, but it is worth it. Fuzzing gives me a level of confidence in my smart contracts that I didn’t have before. Relying just on unit testing anymore and poking around in a testnet seems reckless now.”

(<https://medium.com/coinmonks/smart-contract-fuzzing-d9b88e0b0a05>)



### [N-14] NatSpec comments should be increased in contracts

Context: All Contracts



## Description

It is recommended that Solidity contracts are fully annotated using NatSpec for all public interfaces (everything in the ABI). It is clearly stated in the Solidity official documentation.

In complex projects such as Defi, the interpretation of all functions and their arguments and returns is important for code readability and auditability.

(<https://docs.soliditylang.org/en/v0.8.15/natspec-format.html>)



## [N-15] 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-16] Add a timelock to critical functions

It is a good practice to give time for users to react and adjust to critical changes. A timelock provides more guarantees and reduces the level of trust required, thus decreasing risk for users. It also indicates that the project is legitimate (less risk of a malicious owner making a sandwich attack on a user). Consider adding a timelock to:

FILE : 2023-03-canto-identity/canto-namespace-protocol/src/Namespace.sol

```
function changeNoteAddress(address _newNoteAddress) external onlyOwner
    address currentNoteAddress = address(note);
    note = ERC20(_newNoteAddress);
    emit NoteAddressUpdate(currentNoteAddress, _newNoteAddress);
}

/// @notice Change the revenue address
/// @param _newRevenueAddress New address to use
function changeRevenueAddress(address _newRevenueAddress) external onlyOwner
    address currentRevenueAddress = revenueAddress;
    revenueAddress = _newRevenueAddress;
    emit RevenueAddressUpdated(currentRevenueAddress, _newRevenueAddress);
}
```

[Namespace.sol#L196-L208](#)

FILE : 2023-03-canto-identity/canto-namespace-protocol/src/Tray.sol

```
function changeNoteAddress(address _newNoteAddress) external onlyOwner
    address currentNoteAddress = address(note);
    note = ERC20(_newNoteAddress);
    emit NoteAddressUpdate(currentNoteAddress, _newNoteAddress);
}

/// @notice Change the revenue address
/// @param _newRevenueAddress New address to use
function changeRevenueAddress(address _newRevenueAddress) external onlyOwner
    address currentRevenueAddress = revenueAddress;
    revenueAddress = _newRevenueAddress;
    emit RevenueAddressUpdated(currentRevenueAddress, _newRevenueAddress);
}

/// @notice End the prelaunch phase and start the public mint
```

```
function endPrelaunchPhase() external onlyOwner {  
    if (prelaunchMinted != type(uint256).max) revert PrelaunchAl  
    prelaunchMinted = _nextTokenId() - 1;  
    emit PrelaunchEnded();  
}
```

[Tray.sol#L210-L229](#)



[N-17] Long lines are not suitable for the ‘Solidity Style Guide’

[ProfilePicture.sol#L56](#)

[Bio.sol#L41](#)

[Bio.sol#L53](#)

[Bio.sol#L69](#)

[Bio.sol#L98](#)

[Namespace.sol#L35](#)

[Namespace.sol#L117](#)

[Namespace.sol#L126](#)

[Namespace.sol#L152](#)

[Tray.sol#L247](#)

[Utils.sol#L19-L49](#)

[Utils.sol#L69-L72](#)

[Utils.sol#L140](#)

[Utils.sol#L195](#)

[Utils.sol#L247](#)



Recommendation

Multiline output parameters and return statements should follow the same style recommended for wrapping long lines found in the Maximum Line Length section.

(<https://docs.soliditylang.org/en/v0.8.17/style-guide.html#introduction>)



[N-18] 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.

FILE : 2023-03-canto-identity/canto-bio-protocol/src/Bio.sol

```
87: assembly {
```

### [Bio.sol#L87](#)

FILE : 2023-03-canto-identity/canto-namespace-protocol/src/Namespace.sol

```
165:         assembly {
```

### [Namespace.sol#L165](#)



## [N-19] Critical changes should use two-step procedure

Lack of two-step procedure for critical operations leaves them error-prone. Consider adding two-step procedure on the critical functions.

Consider adding a two-steps pattern on critical changes to avoid mistakenly transferring ownership of roles or critical functionalities to the wrong address.

FILE : 2023-03-canto-identity/canto-namespace-protocol/src/Namespace.sol

```
function changeNoteAddress(address _newNoteAddress) external onlyOwner {
    address currentNoteAddress = address(note);
    note = ERC20(_newNoteAddress);
    emit NoteAddressUpdate(currentNoteAddress, _newNoteAddress);
}
```

```
/// @notice Change the revenue address
```

```

/// @param _newRevenueAddress New address to use
function changeRevenueAddress(address _newRevenueAddress) external
    address currentRevenueAddress = revenueAddress;
    revenueAddress = _newRevenueAddress;
    emit RevenueAddressUpdated(currentRevenueAddress, _newRevenue
}

```

## [Namespace.sol#L196-L208](#)

FILE : 2023-03-canto-identity/canto-namespace-protocol/src/Tray.sol

```

function changeNoteAddress(address _newNoteAddress) external only
    address currentNoteAddress = address(note);
    note = ERC20(_newNoteAddress);
    emit NoteAddressUpdate(currentNoteAddress, _newNoteAddress);
}

```

```

/// @notice Change the revenue address
/// @param _newRevenueAddress New address to use
function changeRevenueAddress(address _newRevenueAddress) external
    address currentRevenueAddress = revenueAddress;
    revenueAddress = _newRevenueAddress;
    emit RevenueAddressUpdated(currentRevenueAddress, _newRevenue
}

```

```

/// @notice End the prelaunch phase and start the public mint
function endPrelaunchPhase() external onlyOwner {
    if (prelaunchMinted != type(uint256).max) revert PrelaunchAl
    prelaunchMinted = _nextTokenId() - 1;
    emit PrelaunchEnded();
}

```

## [Tray.sol#L210-L229](#)



## [N-20] Missing NATSPEC

FILE : 2023-03-canto-identity/canto-pfp-protocol/src/ProfilePicture.sol

## [ProfilePicture.sol#L40-L45](#)

## [ProfilePicture.sol#L50-L52](#)

## [Bio.sol#L23](#)

[Namespace.sol#L54-L67](#)

[Tray.sol#L78-L90](#)



## [N-21] Imports can be grouped together

Group solmate/utils imports

```
import {ERC721A} from "erc721a/ERC721A.sol";
import {ERC20} from "solmate/tokens/ERC20.sol";
import {SafeTransferLib} from "solmate/utils/SafeTransferLib.sol";
import {Owned} from "solmate/auth/Owned.sol";
import {LibString} from "solmate/utils/LibString.sol";
import {Base64} from "solady/utils/Base64.sol";
import "./Utils.sol";
import "../interface/Turnstile.sol";
```

[Tray.sol#L4-L11](#)



## [S-01] Project Upgrade and Stop Scenario should be

Suggestion

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



## Gas Optimizations

For this contest, 41 reports were submitted by wardens detailing gas optimizations. The [report highlighted below](#) by OxSmartContract received the top score from the judge.

*The following wardens also submitted reports:* [Kresh](#), [anodaram](#), [viking71](#), [Aymen0909](#), [lukris02](#), [nadin](#), [Oxdaydream](#), [Udsen](#), [Shubham](#), [tnevler](#), [EvanW](#), [slvDev](#), [atharvasama](#), [Walter](#), [Deekshith99](#), [turvy\\_fuzz](#), [descharre](#), [Jerry0x](#), [SAAJ](#),

JCN, [Diana](#), [iging](#), [Fanz](#), [MiniGlome](#), [codeslide](#), [felipe](#), [Viktor Cortess](#), [Rolezn](#), [caspersolangii](#), [Oxnev](#), [ReyAdmirado](#), [Rageur](#), [Polaris\\_tow](#), [fatherOfBlocks](#), [SaeedAlipoor01988](#), [ginlee](#), [Sathish9098](#), [arialblack14](#), [K42](#), *and* [Madalad](#).



## Summary

Num ber	Optimization Details	Context
[G-01]	Gas overflow during iteration (DoS)	2
[G-02]	State variables only set in the constructor should be declared immutable	1
[G-03]	Failure to check the zero address in the constructor causes the contract to be deployed again	3
[G-04]	By updating <code>emit</code> , extra emit usage can be reduced	1
[G-05]	Emit can be rearranged	4
[G-06]	Use <code>assembly</code> to write <i>address storage values</i>	2
[G-07]	<i>Functions guaranteed to revert</i> when called by normal users can be marked <code>payable</code>	6
[G-08]	Multiple <code>address</code> /ID mappings can be combined into a single <code>mapping</code> of an <code>address</code> /ID to a <code>struct</code> , where appropriate	2
[G-09]	Use a more recent version of solidity	5
[G-10]	The result of function calls should be cached rather than re-calling the function	1
[G-11]	Avoid contract existence checks by using low level calls	11
[G-12]	<code>bytes</code> constants are more efficient than <code>string</code> constans	1
[G-13]	Change <code>public</code> function visibility to <code>external</code>	3
[G-14]	Usage of uints/ints smaller than 32 bytes (256 bits) incurs overhead	15
[G-15]	Setting the <i>constructor</i> to <code>payable</code>	4

Number	Optimization Details	Context
[G-16]	Use nested if and, avoid multiple check combinations	7
[G-17]	Sort Solidity operations using short-circuit mode	4
[G-18]	Use ERC721A instead ERC721	
[G-19]	Optimize names to save gas	All Contract
[G-20]	Upgrade Solidity's optimizer	3

## 🔗 [G-01] Gas overflow during iteration (DoS)

Each iteration of the cycle requires a gas flow. A moment may come when more gas is required than it is allocated to record one block. In this case, all iterations of the loop will fail.

2 results - 2 files:

```
canto-namespace-protocol\src\Namespace.sol:
```

```

109      /// @param _characterList The tiles to use for the fu
110:      function fuse(CharacterData[] calldata _characterList
+          require(_characterList.length < max _characterLis

111:          uint256 numCharacters = _characterList.length;
112:          if (numCharacters > 13 || numCharacters == 0) rev
113:          uint256 fusingCosts = 2** (13 - numCharacters) * 1
114:          SafeTransferLib.safeTransferFrom(note, msg.sender
115:          uint256 namespaceIDToMint = ++nextNamespaceIDToMi
116:          Tray.TileData[] storage nftToMintCharacters = nft
117:          bytes memory bName = new bytes (numCharacters * 33
118:          uint256 numBytes;
119:          // Extract unique trays for burning them later or
120:          uint256 numUniqueTrays;
121:          uint256[] memory uniqueTrays = new uint256[] (_cha
122:          for (uint256 i; i < numCharacters; ++i) {
123:              bool isLastTrayEntry = true;
```

```

124:         uint256 trayID = _characterList[i].trayID;
125:         uint8 tileOffset = _characterList[i].tileOffs
126:         // Check for duplicate characters in the prov

```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-namespace-protocol/src/Namespace.sol#L110>

canto-namespace-protocol\src\Utils.sol:

```

222:     function generateSVG(Tray.TileData[] memory _tiles, k
+         require(_tiles.length < max _tilesLengt, "max ler
223:         string memory innerData;
224:         uint256 dx = 400 / (_tiles.length + 1);
225:         for (uint256 i; i < _tiles.length; ++i) {
226:             innerData = string.concat(
227:                 innerData,
228:                 '<text dominant-baseline="middle" text-ar
229:                 LibString.toString(dx * (i + 1)),
230:                 '>',
231:                 string(
232:                     characterToUnicodeBytes(_tiles[i].for
233:                 ),
234:                 "</text>"
235:             );
236:             if (_isTray) {
237:                 innerData = string.concat(
238:                     innerData,
239:                     '<rect width="34" height="60" y="70"
240:                     LibString.toString(dx * (i + 1) - 17)
241:                     '" stroke="black" stroke-width="1" fi
242:                 );
243:             }
244:         }
245:         return
246:             string.concat(
247:                 '<svg xmlns="http://www.w3.org/2000/svg"
248:                 innerData,
249:                 "</svg>"
250:             );
251:     }

```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-namespace-protocol/src/Utils.sol#L222>

Here are the data available in the covered contracts. The use of this situation in contracts that are not covered will also provide gas optimization.



## [G-02] State variables only set in the constructor should be declared immutable

Avoids a Gsset (20000 gas) in the constructor and replaces the first access in each transaction (Gcoldload - 2100 gas) and each access thereafter (Gwarmacces - 100 gas) with a PUSH32 (3 gas).

```
canto-pfp-protocol\src\ProfilePicture.sol:
```

```
35:         string public subprotocolName;
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-pfp-protocol/src/ProfilePicture.sol#L35>



## [G-03] Failure to check the zero address in the constructor causes the contract to be deployed again

Zero address control is not performed in the constructor in all 3 contracts within the scope of the audit. Bypassing this check could cause the contract to be deployed by mistakenly entering a zero address. In this case, the contract will need to be redeployed. This means extra gas consumption as contract deployment costs are high.

3 results - 3 files:

```
canto-namespace-protocol\src\Namespace.sol:
```

```
73:         constructor(  
74:             address _tray,  
75:             address _note,  
76:             address _revenueAddress  
77:         ) ERC721("Namespace", "NS") Owned(msg.sender) {
```

```

78:         tray = Tray(_tray);
79:         note = ERC20(_note);
80:         revenueAddress = _revenueAddress;
81:         if (block.chainid == 7700) {
82:             // Register CSR on Canto mainnet
83:             Turnstile turnstile = Turnstile(0xEcf044C5B4b8
84:             turnstile.register(tx.origin);
85:         }
86:     }

```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-namespace-protocol/src/Namespace.sol#L73>

canto-namespace-protocol\src\Tray.sol:

```

97         /// @param _namespaceNFT Address of the Namespace NFT
98:     constructor(
99:         bytes32 _initHash,
100:         uint256 _trayPrice,
101:         address _revenueAddress,
102:         address _note,
103:         address _namespaceNFT
104:     ) ERC721A("Namespace Tray", "NSTRAY") Owned(msg.sender) {
105:         lastHash = _initHash;
106:         trayPrice = _trayPrice;
107:         revenueAddress = _revenueAddress;
108:         note = ERC20(_note);
109:         namespaceNFT = _namespaceNFT;
110:         if (block.chainid == 7700) {
111:             // Register CSR on Canto mainnet
112:             Turnstile turnstile = Turnstile(0xEcf044C5B4b8
113:             turnstile.register(tx.origin);
114:         }
115:     }

```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-namespace-protocol/src/Tray.sol#L98>

canto-pfp-protocol\src\ProfilePicture.sol:

```

56         /// @param _subprotocolName Name with which the subproc

```



```

57:         constructor(address _cidNFT, string memory _subprotoc
58:             cidNFT = ICidNFT(_cidNFT);
59:             subprotocolName = _subprotocolName;
60:             if (block.chainid == 7700) {
61:                 // Register CSR on Canto mainnet
62:                 Turnstile turnstile = Turnstile(0xEcf044C5B4b8
63:                 turnstile.register(tx.origin);
64:             }
65:     }

```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-pfp-protocol/src/ProfilePicture.sol#L57>



## [G-04] By updating `emit`, extra emit usage can be reduced

In the `mint` function in the `Bio.sol` contract, the protocol emits two emits (the `_mint` and `bioAdded` emits). By updating the emit of Solady's `_mint` function, the use of one extra emit can be reduced.

Reference: [https://github.com/ethereum/go-ethereum/blob/master/params/protocol\\_params.go#L78-L82](https://github.com/ethereum/go-ethereum/blob/master/params/protocol_params.go#L78-L82)

```
canto-bio-protocol/lib/solady/lib/solmate/src/tokens/ERC721.sol:
```

```
+ event Transfer(address from, address indexed to, uint256 index)
```

```

+ 157:     function _mint(address to, uint256 id) internal virtu
+ 158:         require(to != address(0), "INVALID_RECIPIENT");
+ 159:
+ 160:         require(_ownerOf[id] == address(0), "ALREADY_MINT
+ 161:
+ 162:         // Counter overflow is incredibly unrealistic.
+ 163:         unchecked {
+ 164:             _balanceOf[to]++;
+ 165:         }
+ 166:
+ 167:         _ownerOf[id] = to;
+ 168:
+ 169:         emit Transfer(address(0), to, tokenId, _bio );
+ 170:     }

```

canto-bio-protocol/src/Bio.sol:

```
23:      event BioAdded(address indexed minter, uint256 indexed r
120      /// @param _bio The text to add
121:      function mint(string calldata _bio) external {
122:          // We check the length in bytes, so will be higher
123:          if (bytes(_bio).length == 0 || bytes(_bio).length > 255)
124:              uint256 tokenId = ++numMinted;
125:              bio[tokenId] = _bio;
- 126:              _mint(msg.sender, tokenId,);
+ 126:              _mint(msg.sender, tokenId, _bio);

- 127:              emit BioAdded(msg.sender, tokenId, _bio);
128:      }
```



## [G-05] Emit can be rearranged

The emit can be rearranged in the `changeNoteAddress()` and `changeRevenueAddress()` functions. With this arrangement, both the deploy time ( $\sim 1.4 \text{ k} * 4 = 5.6 \text{ k}$  gas per function) and every time the functions are triggered ( $\sim 16$  gas) gas savings will be achieved.

4 results - 2 files:

canto-namespace-protocol/src/Tray.sol:

```
209      /// @param _newNoteAddress New address to use
210:      function changeNoteAddress(address _newNoteAddress) external {
211:          address currentNoteAddress = address(note);
212:          note = ERC20(_newNoteAddress);
213:          emit NoteAddressUpdate(currentNoteAddress, _newNoteAddress);
214:      }

217      /// @param _newRevenueAddress New address to use
218:      function changeRevenueAddress(address _newRevenueAddress) external {
219:          address currentRevenueAddress = revenueAddress;
220:          revenueAddress = _newRevenueAddress;
221:          emit RevenueAddressUpdated(currentRevenueAddress, _newRevenueAddress);
222:      }
```

canto-namespace-protocol\src\Namespace.sol:

```
195      /// @param _newNoteAddress New address to use
196:      function changeNoteAddress(address _newNoteAddress) €
197:          address currentNoteAddress = address(note);
198:          note = ERC20(_newNoteAddress);
199:          emit NoteAddressUpdate(currentNoteAddress, _newNoteAddress);
200:      }

203      /// @param _newRevenueAddress New address to use
204:      function changeRevenueAddress(address _newRevenueAddress) €
205:          address currentRevenueAddress = revenueAddress;
206:          revenueAddress = _newRevenueAddress;
207:          emit RevenueAddressUpdated(currentRevenueAddress, _newRevenueAddress);
208:      }
209  }
```

canto-namespace-protocol\src\Namespace.sol:

```
195      /// @param _newNoteAddress New address to use
196:      function changeNoteAddress(address _newNoteAddress) €
- 197:          address currentNoteAddress = address(note);
- 198:          note = ERC20(_newNoteAddress);
- 199:          emit NoteAddressUpdate(currentNoteAddress, _newNoteAddress);
+ 199:          emit NoteAddressUpdate(note, _newNoteAddress);
+ 198:          note = ERC20(_newNoteAddress);
200:      }
```



## [G-06] Use assembly to write *address storage values*

2 results - 4 files:

canto-namespace-protocol\src\Namespace.sol:

```
73:      constructor(
80:          revenueAddress = _revenueAddress;

204:      function changeRevenueAddress(address _newRevenueAddress) €
```

```
206:         revenueAddress = _newRevenueAddress;
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-namespace-protocol/src/Namespace.sol#L80>

```
canto-namespace-protocol\src\Tray.sol:
```

```
98:         constructor(  
107:             revenueAddress = _revenueAddress;  
  
218:         function changeRevenueAddress(address _newRevenueAddr  
220:             revenueAddress = _newRevenueAddress;
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-namespace-protocol/src/Tray.sol#L107>



## Recommendation Code

```
canto-namespace-protocol\src\Tray.sol:
```

```
98:         constructor(  
- 107:             revenueAddress = _revenueAddress;  
+             assembly {  
+                 sstore(vault. revenueAddress, _revenueAdc  
+             }  
+             }
```



**[G-07] Functions guaranteed to revert when called by normal users can be marked payable**

If a function modifier such as `onlyOwner` is used, the function will revert if a normal user tries to pay the function. Marking the function as `payable` will lower the gas cost for legitimate callers because the compiler will not include checks for whether a payment was provided.

6 results - 2 files:

```
canto-namespace-protocol\src\Namespace.sol:
```

```
196:         function changeNoteAddress(address _newNoteAddress) €
```

```
204:         function changeRevenueAddress(address _newRevenueAddr
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-namespace-protocol/src/Namespace.sol#L196>

```
canto-namespace-protocol\src\Tray.sol:
```

```
210:         function changeNoteAddress(address _newNoteAddress) €
```

```
218:         function changeRevenueAddress(address _newRevenueAddr
```

```
225:         function endPrelaunchPhase() external onlyOwner {
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-namespace-protocol/src/Tray.sol#L210>



**[G-08] Multiple address /ID mappings can be combined into a single mapping of an address /ID to a struct , where appropriate**

Saves a storage slot for the mapping. Depending on the circumstances and sizes of types, can avoid a Gsset (20000 gas) per mapping combined. Reads and subsequent writes can also be cheaper when a function requires both values and they both fit in the same storage slot. Finally, if both fields are accessed in the same function, can save ~42 gas per access due to not having to recalculate the key's keccak256 hash (Gkeccak256 - 30 gas) and that calculation's associated stack operations.

2 results - 1 file:

```
canto-namespace-protocol\src\Namespace.sol:
```

```
43:         mapping(string => uint256) public nameToToken;
```

```
46:         mapping(uint256 => string) public tokenToName;
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-namespace-protocol/src/Namespace.sol#L43-L46>



## [G-09] Use a more recent version of solidity

- Use a solidity version of at least 0.8.0 to get overflow protection without SafeMath
- Use a solidity version of at least 0.8.2 to get simple compiler automatic inlining
- Use a solidity version of at least 0.8.3 to get better struct packing and cheaper multiple storage reads
- Use a solidity version of at least 0.8.4 to get custom errors, which are cheaper at deployment than revert()/require() strings
- Use a solidity version of at least 0.8.10 to have external calls skip contract existence checks if the external call has a return value
- In 0.8.15 the conditions necessary for inlining are relaxed. Benchmarks show that the change significantly decreases the bytecode size (which impacts the deployment cost) while the effect on the runtime gas usage is smaller.
- In 0.8.17 prevent the incorrect removal of storage writes before calls to Yul functions that conditionally terminate the external EVM call; Simplify the starting offset of zero-length operations to zero. More efficient overflow checks for multiplication.

5 results - 5 files:

```
canto-bio-protocol/src/Bio.sol:
```

```
2: pragma solidity >=0.8.0;
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-bio-protocol/src/Bio.sol#L2>

```
canto-namespace-protocol/src/Namespace.sol:
```

```
2: pragma solidity >=0.8.0;
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-namespace-protocol/src/Namespace.sol#L2>

```
canto-namespace-protocol/src/Tray.sol:
```

```
2: pragma solidity >=0.8.0;
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-namespace-protocol/src/Tray.sol#L2>

```
canto-namespace-protocol/src/Utils.sol:
```

```
2: pragma solidity >=0.8.0;
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-namespace-protocol/src/Utils.sol#L2>

```
canto-pfp-protocol/src/ProfilePicture.sol:
```

```
2: pragma solidity >=0.8.0;
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-pfp-protocol/src/ProfilePicture.sol#L2>



**[G-10] The result of function calls should be cached rather than re-calling the function**

```
canto-namespace-protocol\src\Tray.sol:
```

```
245:         function _drawing(uint256 _seed) private pure returns
```

```
247:             uint256 charRandValue = Utils.iteratePRNG(_seed);
```

```

265:                                // Set seed for Zalgo to ensure same
- 266:                                uint256 zalgoSeed = Utils.iteratePRNG
+ 266:                                uint256 zalgoSeed = charRandValue;
267:                                tileData.characterModifier = uint8(zal
268:                                }

```



## [G-11] Avoid contract existence checks by using low level calls

Prior to 0.8.10 the compiler inserted extra code, including `EXTCODESIZE` (100 gas), to check for contract existence for external function calls. In more recent solidity versions, the compiler will not insert these checks if the external call has a return value. Similar behavior can be achieved in earlier versions by using low-level calls, since low level calls never check for contract existence.

12 results - 4 files:

```
canto-bio-protocol\src\Bio.sol:
```

```

// @audit register()
36:     turnstile.register(tx.origin);

```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-bio-protocol/src/Bio.sol#L36>

```
canto-namespace-protocol\src\Namespace.sol:
```

```

// @audit register()
84:     turnstile.register(tx.origin);

// @audit getTile()
133:     Tray.TileData memory tileData = tray.getTile(trayID, ti

// @audit ownerOf()
156:     address trayOwner = tray.ownerOf(trayID);

// @audit getApproved()
159:     tray.getApproved(trayID) != msg.sender &&

// @audit isApprovedForAll()

```



```
160:    !tray.isApprovedForAll(trayOwner, msg.sender)
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-namespace-protocol/src/Namespace.sol#L84>

```
canto-namespace-protocol\src\Tray.sol:
```

```
// @audit register()  
113:    turnstile.register(tx.origin);
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-namespace-protocol/src/Tray.sol#L113>

```
canto-pfp-protocol\src\ProfilePicture.sol:
```

```
// @audit register()  
63:    turnstile.register(tx.origin);  
  
// @audit tokenURI()  
73:    return ERC721(nftContract).tokenURI(nftID);  
  
// @audit ownerOf()  
81:    if (ERC721(_nftContract).ownerOf(_nftID) != msg.sender)  
  
// @audit addressRegistry()  
100:    IAddressRegistry addressRegistry = cidNFT.addressRegist  
  
// @audit getAddress(), ownerOf()  
101:    if (cidNFTID == 0 || addressRegistry.getAddress(cidNFTI
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-pfp-protocol/src/ProfilePicture.sol#L63>



**[G-12]** bytes constants are more efficient than string constans

If the data can fit in 32 bytes, the bytes32 data type can be used instead of bytes or strings, as it is less robust in terms of robustness.

```
canto-pfp-protocol\src\ProfilePicture.sol:
```

```
35:     string public subprotocolName;
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-pfp-protocol/src/ProfilePicture.sol#L35>



## [G-13] Change public function visibility to external

Since the following public functions are not called from within the contract, their visibility can be made external for gas optimization.

3 results - 3 files:

```
canto-bio-protocol\src\Bio.sol:
```

```
43:     function tokenURI(uint256 _id) public view override re
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-bio-protocol/src/Bio.sol#L43>

```
canto-namespace-protocol\src\Namespace.sol:
```

```
90:     function tokenURI(uint256 _id) public view override re
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-namespace-protocol/src/Namespace.sol#L90>

```
canto-namespace-protocol\src\Tray.sol:
```

```
119:    function tokenURI(uint256 _id) public view override r
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-namespace-protocol/src/Tray.sol#L119>



[G-14] Usage of uints/ints smaller than 32 bytes (256 bits) incurs overhead

When using elements that are smaller than 32 bytes, your contracts gas usage may be higher. This is because the EVM operates on 32 bytes at a time. Therefore, if the element is smaller than that, the EVM must use more operations in order to reduce the size of the element from 32 bytes to the desired size.

[https://docs.soliditylang.org/en/v0.8.11/internals/layout\\_in\\_storage.html](https://docs.soliditylang.org/en/v0.8.11/internals/layout_in_storage.html)

Use a larger size then downcast where needed.

15 result - 2 files:

```
canto-namespace-protocol\src\Tray.sol:

// @audit uint8 fontClass
259:         tileData.fontClass = 3 + uint8((res - 80) / 8);
261:         tileData.fontClass = 5 + uint8((res - 96) / 4);
263:         tileData.fontClass = 7 + uint8((res - 104) / 2);


// @audit uint8 characterModifier
267:         tileData.characterModifier = uint8(zalgoSeed % 256);
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-namespace-protocol/src/Tray.sol#L259>

```
canto-namespace-protocol\src\Utils.sol:

// @audit uint16 _characterIndex
85:         byteOffset = _characterIndex * 3;
86:         supportsSkinToneModifier = _characterIndex >= EMOJIS_
89:         byteOffset = EMOJIS_BYTE_OFFSET_FOUR_BYTES + (_charac
93:         byteOffset = EMOJIS_BYTE_OFFSET_SIX_BYTES + (_charact
96:         byteOffset = EMOJIS_BYTE_OFFSET_SEVEN_BYTES + (_chara
99:         byteOffset = EMOJIS_BYTE_OFFSET_EIGHT_BYTES + (_chara
102:        byteOffset = EMOJIS_BYTE_OFFSET_FOURTEEN_BYTES + (_ch

// @audit uint8 asciiStartingIndex
135:        return abi.encodePacked(bytes1(asciiStartingIndex + u
```

```
// @audit uint8 asciiStartingIndex, _characterIndex
145:     bytes memory character = abi.encodePacked(bytes1(ascii
273:     uint8 lastByteSum = lastByteVal + _characterIndex;
277:     lastByteVal = 128 + (lastByteSum % 192);
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-namespace-protocol/src/Utils.sol#L85>



## [G-15] Setting the *constructor* to payable

You can cut out 10 opcodes in the creation-time EVM bytecode if you declare a constructor payable. Making the constructor payable eliminates the need for an initial check of `msg.value == 0` and saves 13 gas on deployment with no security risks.

4 results - 4 files:

```
canto-bio-protocol\src\Bio.sol:
```

```
32:     constructor() ERC721("Biography", "Bio") {
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-bio-protocol/src/Bio.sol#L32>

```
canto-namespace-protocol\src\Namespace.sol:
```

```
73:     constructor(
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-namespace-protocol/src/Namespace.sol#L73>

```
canto-namespace-protocol\src\Tray.sol:
```

```
98:     constructor(
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-namespace-protocol/src/Tray.sol#L98>

```
canto-pfp-protocol\src\ProfilePicture.sol:
```

```
57:         constructor(address _cidNFT, string memory _subprotoc
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-pfp-protocol/src/ProfilePicture.sol#L57>



## Recommendation

Set the constructor to payable



## [G-16] Use nested if and, avoid multiple check combinations

Using nested is cheaper than using && multiple check combinations. There are more advantages, such as easier to read code and better coverage reports.

7 results - 3 files:

```
canto-bio-protocol\src\Bio.sol:
```

```
60:         if ((i > 0 && (i + 1) % 40 == 0) || prevByteWasContinu
```

```
65:         if (nextCharacter & 0xC0 == 0x80) {
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-bio-protocol/src/Bio.sol#L60>

```
canto-namespace-protocol\src\Namespace.sol:
```

```
136:         if (tileData.fontClass != 0 && _characterList[i].skinT
```

```
157:         if (
```

```
158:             trayOwner != msg.sender &&
```

```
159:             tray.getApproved(trayID) != msg.sender &&
```

```

160:         !tray.isApprovedForAll(trayOwner, msg.sender)
161:     ) revert CallerNotAllowedToFuse();

186:     if (nftOwner != msg.sender && getApproved[_id] != msg.

```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-namespace-protocol/src/Namespace.sol#L136>

canto-namespace-protocol\src\Tray.sol:

```

175:     if (
176:         namespaceNFT != msg.sender &&
177:         trayOwner != msg.sender &&
178:         getApproved(_id) != msg.sender &&
179:         !isApprovedForAll(trayOwner, msg.sender)
180:     ) revert CallerNotAllowedToBurn();

184:     if (numPrelaunchMinted != type(uint256).max && _id <=

```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-namespace-protocol/src/Tray.sol#L175-L180>



## [G-17] Sort Solidity operations using short-circuit mode

Short-circuiting is a solidity contract development model that uses OR/AND logic to sequence different cost operations. It puts low gas cost operations in the front and high gas cost operations in the back, so that if the front is low If the cost operation is feasible, you can skip (short-circuit) the subsequent high-cost Ethereum virtual machine operation.

```

//f(x) is a low gas cost operation
//g(y) is a high gas cost operation

//Sort operations with different gas costs as follows
f(x) || g(y)
f(x) && g(y)

```

4 results - 3 files:

```
canto-bio-protocol\src\Bio.sol:
```

```
60:     if ((i > 0 && (i + 1) % 40 == 0) || prevByteWasContinu
```

```
123:     if (bytes(_bio).length == 0 || bytes(_bio).length > 200
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-bio-protocol/src/Bio.sol#L60>

```
canto-namespace-protocol\src\Namespace.sol:
```

```
112:     if (numCharacters > 13 || numCharacters == 0) revert Ir
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-namespace-protocol/src/Namespace.sol#L112>

```
canto-pfp-protocol\src\ProfilePicture.sol:
```

```
101:     if (cidNFTID == 0 || addressRegistry.getAddress(cidNFTI
```

<https://github.com/code-423n4/2023-03-canto-identity/blob/main/canto-pfp-protocol/src/ProfilePicture.sol#L101>



## [G-18] Use ERC721A instead ERC721

ERC721A standard, ERC721A is an improvement standard for ERC721 tokens. It was proposed by the Azuki team and used for developing their NFT collection.

Compared with ERC721, ERC721A is a more gas-efficient standard to mint a lot of of NFTs simultaneously. It allows developers to mint multiple NFTs at the same gas price. This has been a great improvement due to Ethereum's sky-rocketing gas fee.

Reference: <https://nextrope.com/erc721-vs-erc721a-2/>



## [G-19] Optimize names to save gas

Contracts most called functions could simply save gas by function ordering via `Method ID`. Calling a function at runtime will be cheaper if the function is positioned earlier in the order (has a relatively lower Method ID) because `22 gas` are added to the cost of a function for every position that came before it. The caller can save on gas if you prioritize most called functions.

Context: All Contracts



## Recommendation

Find a lower `method ID` name for the most called functions for example `Call()` vs. `Call1()` is cheaper by `22 gas`.

For example, the function IDs in the `Namespace.sol` contract will be the most used; A lower method ID may be given.



## Proof of Concept

<https://medium.com/joyso/solidity-how-does-function-name-affect-gas-consumption-in-smart-contract-47d270d8ac92>

`Namespace.sol` function names can be named and sorted according to `METHOD ID`

Sighash		Function Signature
=====		
c87b56dd	=>	tokenURI(uint256)
11f839e2	=>	fuse(CharacterData[])
42966c68	=>	burn(uint256)
21496de6	=>	changeNoteAddress(address)
92f8b9a3	=>	changeRevenueAddress(address)



## [G-20] Upgrade Solidity's optimizer

Make sure Solidity's optimizer is enabled. It reduces gas costs. If you want to gas optimize for contract deployment (costs less to deploy a contract) then set the Solidity optimizer at a low number. If you want to optimize for run-time gas costs (when functions are called on a contract) then set the optimizer to a high number.

3 results - 3 files:



```
canto-bio-protocol\foundry.toml:
```

```
2: optimizer = true  
3: optimizer_runs = 1_000
```

```
canto-namespace-protocol\foundry.toml:
```

```
2: optimizer = true  
3: optimizer_runs = 1_000
```

```
canto-pfp-protocol\foundry.toml:
```

```
2: optimizer = true  
3: optimizer_runs = 1_000
```



## 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