Nested Finance contest
Findings & Analysis Report

2021-01-12

Table of contents

Overview

About C4

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

A C4 code 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 code contest outlined in this document, C4 conducted an analysis of Nested Finance smart contract system written in Solidity. The code contest took place between November 11—November 17 2021.

Wardens

29 Wardens contributed reports to the Nested Finance contest:

1. GreyArt (hickuphh3 and itsmeSTYJ)
2. jayjonah8
3. WatchPug (jtp and ming)
4. palina
5. cmichel
6. pauliax

7. hyh
8. pants
9. TomFrench
10. fatima_naz
11. hack3r-0m
12. defsec
13. 0xngndev
14. yeOlde
15. gzeon
16. GiveMeTestEther
17. PierrickGT
18. xYrYuYx
19. loop
20. Meta0xNull
21. 0x0x0x
22. pmerkleplant
23. harleythedog
24. elprofesor
25. MaCree
26. gpersoon
27. nathaniel

This contest was judged by Alberto Cuesta Cañada.

Final report assembled by moneylegobatman and CloudEllie.

Summary

The C4 analysis yielded an aggregated total of 31 unique vulnerabilities and 101 total findings. All of the issues presented here are linked back to their original finding.

Of these vulnerabilities, 1 received a risk rating in the category of HIGH severity, 8 received a risk rating in the category of MEDIUM severity, and 22 received a risk rating in the category of LOW severity.

C4 analysis also identified 20 non-critical recommendations and 50 gas optimizations.

Scope

The code under review can be found within the C4 Nested Finance contest repository, and is composed of 29 smart contracts written in the Solidity programming language and includes 1960 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into three primary risk categories: high, medium, and low.

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

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website.

High Risk Findings (1)

[H-01] Copy your own portfolio to keep earning royalties

Submitted by jayjonah8

Impact

In `NestedFactory.sol` going through the `create()` function which leads to the `sendFeesWithRoyalties()` `=>` `addShares()` function, Im not seeing any checks preventing someone from copying their own portfolio and receiving royalty shares for it and simply repeating the process over and over again.

Proof of Concept

- `FeeSplitter.sol` L152
- `FeeSplitter.sol` L220
- `NestedFactory.sol` L103
- `NestedAsset.sol` L69
- `NestedFactory.sol` L103
- `NestedFactory.sol` L491

Tools Used

Manual code review

Recommended Mitigation Steps

A require statement should be added not allowing users to copy their own portfolios.

maximebrugel (Nested) disagreed with severity:

> Indeed, a user can copy his own portfolio to reduce the fees, however a require statement won't fix this issue…

> This problem cannot be corrected but only mitigated, since the user can use two different wallets. Currently the front-end doesn't allow to duplicate a portfolio with the same address.

> I don't consider this a "High Risk" since the assets are not really stolen. Maybe "Med Risk" ? This is by design an issue and we tolerate that users can do this (with multiple wallets).

alcueca (judge) commented:

> I’m reading that the vulnerability actually lowers fees to zero for a dedicated attacker, since creating a arbitrarily large number of wallets and bypassing the frontend is easy. In theory leaking protocol value would be a severity 2, but since this is effectively disabling a core feature of the protocol (fees), the severity 3 is sustained.

Medium Risk Findings (8)

[M-01] `setReserve()` can be front-run

Submitted by palina

Impact

The `reserve` address variable in NestedFactory.sol remains equal to 0 before the `setReserve()` function is called by an owner. This may lead to incorrect transfers of tokens or invalid comparison with e.g., the asset reserve (nestedRecords.getAssetReserve(_nftId) == address(reserve)), should they occur before the value for `reserve` was set. In addition, the immutabiliy of the `reserve` variable requires extra caution when setting the value.

Proof of Concept

`setReserve()`: [`NestedFactory.sol` L89]

Tools Used

Manual Analysis

Recommended Mitigation Steps

Consider initializing the value for the `reserve` variable in the constructor.

[maximebrugel (Nested) commented]:

> The main issue is duplicated : #60

> The following comment can be considered as a duplicate of #83 if the extra caution is checking the zero address.

> In addition, the immutabiliy of the reserve variable requires extra caution when setting the value.

alcueca (judge) commented:

> The fact that the call to `setReserve` can be front-run is not being taken into account by the sponsor. I'm marking this one as not a duplicate.

[M-02] FeeSplitter: No sanity check to prevent shareholder from being added twice.

Submitted by GreyArt, also found by hack3r-0m

Impact

It is possible for duplicate shareholders to be added. These shareholders will get more than intended when `_sendFee()` is called.

Recommended Mitigation Steps

Ensure that the `_accounts` array is sorted in `setShareholders()`.

```
    for (uint256 i = 0; i < _accounts.length; i++) {
            if (i > 0) {
                    require(_accounts[i - 1] < _accounts[i], "FeeSpl
            }
            _addShareholder(_accounts[i], _weights[i]);
    }
```

adrien-supizet (Nested) commented:

> Duplicate #231

adrien-supizet (Nested) commented:

> Indeed there is a fix to do here, we'll prevent adding the same shareholders instead as suggested in #231

alcueca (judge) commented:

> Taking this issue as the principal, and raising #231 to medium severity.

[M-03] NestedFactory: Ensure zero msg.value if transferring from user and `inputToken` is not ETH

Submitted by GreyArt

Impact

A user that mistakenly calls either `create()` or `addToken()` with WETH (or another ERC20) as the input token, but includes native ETH with the function call will have his native ETH permanently locked in the contract.

Recommended Mitigation Steps

It is best to ensure that `msg.value = 0` in `_transferInputTokens()` for the scenario mentioned above.

```
    } else if (address(_inputToken) == ETH) {
            ...
    } else {
            require(msg.value == 0, "NestedFactory::_transferInputTc
      _inputToken.safeTransferFrom(_msgSender(), address(this), _inp
    }
```

[adrien-supizet (Nested) confirmed]

[M-04] FeeSplitter: Unbounded number of shareholders can cause DOS

Submitted by GreyArt

Impact

There is no limit to the number of shareholders. It is therefore possible to set a large number of shareholders such that `_sendFees()` will run out of gas when adding shares to each shareholder. This will cause denial of service to all NestedFactory functions, especially the ones that will remove funds like `withdraw()` and `destroy()`.

Recommended Mitigation Steps

It would be best to set a sanity maximum number of shareholders that can be added.

adrien-supizet (Nested) acknowledged

[M-05] isResolverCached() will always return false after removing operator

Submitted by GreyArt, also found by WatchPug

Impact

While there is no loss of funds, removing an operator will cause the cache functionality to be permanently broken. If there was a function that had a modifier which requires the cache to be synced before the function can be called, it would not be callable as well.

The underlying issue is how the `bytes32` operator is removed from the array when `removeOperator()` is called. Its value gets deleted (set to 0x0), but isn't taken out of the array.

Proof of Concept

For ease of reading, we use abbreviated strings like `0xA`, `0xB` for `bytes32` and `address` types.

1. Import 3 operators by calling `OperatorResolver.importOperators([0xA, 0xB, 0xC], [0x1, 0x2, 0x3)`.
2. Call `NestedFactory.addOperator()` 3 times to push these 3 operators into the `operators` state variable.
3. Call `NestedFactory.rebuildCache()` to build the cache.
4. Let's say the second operator `0xB` is to be removed. Taking reference from the `removeOperator.ts` script, `OperatorResolver.importOperators([0xA,

`0xB, 0xC], [0x1, 0x2, 0x3)` is called first. This works because OperatorResolver uses a mapping(bytes32 ⇒ address) to represent the operators. Hence, by setting `0xB`'s destination address to be the null address, it is like as if he was never an operator.
5. Call `NestedFactory.rebuildCache()` to rebuild the cache. `resolverAddressesRequired()` will return `[0xA, 0xB, 0xC]`. `0xB` will be removed from `addressCache` because `resolver.getAddress(0xB)` returns 0x000 since it has been deleted from the OperatorResolver.
6. Call `NestedFactory.removeOperator(0xB)`. The `operators` array now looks like this: `[0xA, 0x0, 0xC]`.
7. When you try to call `NestedFactory.isResolverCached`, it will always return false because of the null `bytes32` value, where `addressCache[0x0]` will always return the null address.

Recommended Mitigation Steps

Instead of doing an element deletion, it should be replaced with the last element, then have the last element popped in `removeOperator()`.

```
    function removeOperator(bytes32 operator) external override only
            for (uint256 i = 0; i < operators.length; i++) {
                    if (operators[i] == operator) {
                            operators[i] = operators[operators.lengt
                            operators.pop();
                            break;
                    }
            }
    }
```

[maximebrugel (Nested) commented]:

> Duplicated : #58

[alcueca (judge) commented]:

> Taking this issue apart as a non-duplicate, for finding the most severe consequence of the incorrect implementation.

[M-06] `NestedFactory.sol#_submitInOrders()` Wrong implementation cause users to be overcharged

Submitted by WatchPug

When executing orders, the actual `amountSpent + feesAmount` can be lower than `_inputTokenAmount`, the unspent amount should be returned to the user.

However, in the current implementation, the unspent amount will be taken as part of the fee. `NestedFactory.sol` L285-L309

```
    function _submitInOrders(
        uint256 _nftId,
        IERC20 _inputToken,
        uint256 _inputTokenAmount,
        Order[] calldata _orders,
        bool _reserved,
        bool _fromReserve
    ) private returns (uint256 feesAmount, IERC20 tokenSold) {
        _inputToken = _transferInputTokens(_nftId, _inputToken, _inp
        uint256 amountSpent;
        for (uint256 i = 0; i < _orders.length; i++) {
            amountSpent += _submitOrder(address(_inputToken), _order
        }
        feesAmount = _calculateFees(_msgSender(), amountSpent);
        assert(amountSpent <= _inputTokenAmount - feesAmount); // ov

        // If input is from the reserve, update the records
        if (_fromReserve) {
            _decreaseHoldingAmount(_nftId, address(_inputToken), _ir
        }

        _handleUnderSpending(_inputTokenAmount - feesAmount, amountS

        tokenSold = _inputToken;
    }
```

Recommendation

Change to:

```
    function _submitInOrders(
        uint256 _nftId,
        IERC20 _inputToken,
        uint256 _inputTokenAmount,
        Order[] calldata _orders,
        bool _reserved,
        bool _fromReserve
    ) private returns (uint256 feesAmount, IERC20 tokenSold) {
        _inputToken = _transferInputTokens(_nftId, _inputToken, _inp
        uint256 amountSpent;
        for (uint256 i = 0; i < _orders.length; i++) {
            amountSpent += _submitOrder(address(_inputToken), _order
        }
        feesAmount = _calculateFees(_msgSender(), amountSpent);
        assert(amountSpent <= _inputTokenAmount - feesAmount); // ov

        // If input is from the reserve, update the records
        if (_fromReserve) {
            _decreaseHoldingAmount(_nftId, address(_inputToken), amo
        }

        ExchangeHelpers.setMaxAllowance(_token, address(feeSplitter)
        feeSplitter.sendFees(_token, feesAmount);

        if (_inputTokenAmount > amountSpent + feesAmount) {
            _inputToken.transfer(_fromReserve ? address(reserve) : _
        }

        tokenSold = _inputToken;
    }
```

[adrien-supizet (Nested) disputed and then confirmed](#):

Rationale

> We don’t consider this an issue as this was done on purpose. We wanted to treat the positive slippage as regular fees. Most times, the dust of positive slippage will cost more to the user if they are transferred rather than passed along fees.

> We made it possible for us to retrieve overcharged amounts in case of mistakes to give them back to users.

New behavior

> But for the sake of transparency, and in the spirit of DeFi, we have reviewed the business model of the protocol and decided to transfer back any amount that was unspent and which exceeds the 1% fixed fee.

Resolution

> Selecting “disputed” for now but I’ll let a judge review if this should be included in the report, and if the severity was correct, as we were able to give back tokens to users if they made a mistake calling the protocol.

alcueca (judge) commented:

> If stated in the README or comments, the issue would be invalid. The sponsor can choose whichever behaviour suits their business model. When not stated in the README, any asset loss to users or protocol is a valid issue. User losses are expected to be a severity 3, but in this case, and given that those losses are inferior to the gas, the issue is downgraded to severity 2.

> In the future, please state in the code any asset losses that are accepted by the protocol.

[M-07] Ensure on-chain that cache is synced

Submitted by GreyArt, also found by WatchPug

Impact

Currently, many core operations (like `NestedFactory.create()` , `NestedFactory.swapTokenForTokens()`) are dependent on the assumption that the cache is synced before these functions are executed however this may not necessarily be the case.

Proof of Concept

1. `OperatorResolver.importOperators()` is called to remove an operator.
2. A user calls `NestedFactory.create()` that uses the operator that was being removed / updated.
3. `NestedFactory.rebuildCache()` is called to rebuild cache.

This flow is not aware that the cache is not in synced.

Recommended Mitigation Steps

Add a modifier to require that the cache is synced to all functions that interact with the operators.

[M-08] Passing multiple ETH deposits in orders array will use the same `msg.value` many times

Submitted by hyh, also found by jayjonah8

Impact

Contract holdings can be emptied as malicious user will do deposit/withdraw to extract value. This is possible because after `transferInputTokens` system uses contract balance for user's operations, assuming that equivalent value was transferred.

Proof of Concept

`msg.value` persist over calls, so passing `'Order[] calldata _orders'` holding multiple ETH deposits will use the same msg.value in each of them, resulting in multiple deposits, that sums up to much bigger accounted value than actually deposited value, up to contract's ETH holdings.

create / `addTokens -> submitInOrders -> transferInputTokens`

- `NestedFactory.sol` L103
- `NestedFactory.sol` L119

`sellTokensToWallet -> submitOutOrders -> transferInputTokens`

- `NestedFactory.sol` L172

`sellTokensToNft -> submitOutOrders -> transferInputTokens`

- `NestedFactory.sol` L152 `transferInputTokens` uses msg.value:

`NestedFactory.sol` L462

Recommended Mitigation Steps

Controlling ETH to be only once in orders will not help, as `NestedFactory` inherits from `Multicall`, which `multicall(bytes\[] calldata data)` function allows same reusage of msg.value, which will persist over calls.

So, it is recommended to treat ETH exclusively, not allowing ETH operations to be batched at all.

adrien-supizet (Nested) disagreed with severity:

> Multicall is not currently used, and the funds exposed would be the NestedFactory's which should hold no funds.

> To avoid future bugs, we're going to remove the multicall library, but we don't think this is a high severity issue.

alcueca (judge) commented:

> Downgrading severity to 2 because the NestedFactory is not expected to hold funds, and therefore there is no risk of a loss. You can't deposit the same Ether twice in the WETH contract.

> Also keeping this as the main over #13.

Low Risk Findings (22)

- [L-01] 1:1 linkage between factory and reserve prevents desired upgradability path. *Submitted by TomFrench*
- [L-02] `claimFees` may end up locking user funds *Submitted by 0xngndev*
- [L-03] Use SafeERC20 instead of IERC20 in contracts/mocks/DummyRouter.sol *Submitted by fatima*naz_
- [L-04] FeeSplitter: `totalWeights` can be set to 0 by `onlyOwner` *Submitted by GiveMeTestEther, also found by pauliax*

- [L-05] `ExchangeHelpers` : in `setMaxAllowance` , `safeApprove` shouldn't be used *Submitted by PierrickGT, also found by harleythedog, pants, gzeon, and WatchPug*
- [L-06] Unchecked return value in `triggerForToken()` *Submitted by palina, also found by pauliax*
- [L-07] `NestedFactory.unlockTokens` fails to use safe transfer *Submitted by elprofesor, also found by loop, palina, WatchPug, cmichel, and pauliax*
- [L-08] Add zero-address checkers *Submitted by xYrYuYx, also found by PierrickGT, loop, palina, and pauliax*
- [L-09] DummyRouter.sol .transfer isn't safe *Submitted by pants*
- [L-10] `transferOwnership` should be two step process *Submitted by defsec*
- [L-11] Missing input validation on array lengths *Submitted by yeOlde*
- [L-12] FeeSplitter: `ETH_ADDR` isn't supported *Submitted by GreyArt*
- [L-13] Consider making `_calculateFees` inline to save gas *Submitted by WatchPug, also found by PierrickGT, loop, palina, yeOlde, hyh, and hack3r-0m*
- [L-14] Missing parameter validation *Submitted by cmichel, also found by GreyArt*
- [L-15] Cannot change `tokenUri` *Submitted by cmichel*
- [L-16] Can add duplicate operators *Submitted by cmichel, also found by GreyArt and gzeon*
- [L-17] Function using `msg.value` called in loop *Submitted by cmichel*
- [L-18] `_handleUnderSpending` reverts if condition is false *Submitted by cmichel*
- [L-19] `NestedFactory.addTokens` and withdraw functions require NFT reserve check *Submitted by hyh*
- [L-20] Can't revoke factory in NestedRecrods *Submitted by pauliax*
- [L-21] `NestedFactory.removeOperator` code doesn't correspond to it's logic *Submitted by hyh, also found by loop, MaCree, pmerkleplant, palina, elprofesor, fatimanaz,* fatima*naz, yeOlde, xYrYuYx, gzeon, gpersoon, WatchPug, and pauliax*
- [L-22] Use of `assert()` instead of `require()` *Submitted by WatchPug, also found by fatimanaz_*

Non-Critical Findings (20)

- [N-01] `ZeroExOperator` *Submitted by TomFrench*
- [N-02] Weak guarantees on ŻeroExOperator using correct create2 salt to recompute storage address *Submitted by TomFrench*
- [N-03] Multiple Solidity pragma *Submitted by fatimanaz_*
- [N-04] NestedBuybacker sends NST to NestedReserve with no proper way to retrieve it. *Submitted by TomFrench*
- [N-05] Typo *Submitted by 0xngndev, also found by GreyArt*
- [N-06] Indexing parameters of your events *Submitted by 0xngndev*
- [N-07] Missing events for critical privileged functions *Submitted by GiveMeTestEther, also found by 0x0x0x, elprofesor, pants, and WatchPug*
- [N-08] Comment for PaymentReceived event should state “received” instead of “released” *Submitted by GiveMeTestEther*
- [N-09] Different coding style for same pattern: x += y and sometimes x = x + y *Submitted by GiveMeTestEther*
- [N-10] Remove empty file OwnableOperator.so *Submitted by GiveMeTestEther, also found by WatchPug*
- [N-11] OperatorHelpers.sol: function decodeDataAndRequire state mutability can be restricted to pure *Submitted by GiveMeTestEther, also found by palina and WatchPug*
- [N-12] Wrong Error Message in _transferInputTokens() *Submitted by Meta0xNull*
- [N-13] Missing events on changes *Submitted by palina*
- [N-14] Unused Named Return *Submitted by yeOlde, also found by pants*
- [N-15] NestedFactory._decreaseHoldingAmount needs explicit amount control for spending reserve *Submitted by hyh*
- [N-16] No used library added *Submitted by xYrYuYx*
- [N-17] Misleading error message *Submitted by WatchPug*
- [N-18] NestedAsset.setFactory should be named addFactory *Submitted by hyh*
- [N-19] INestedToken interface *Submitted by pauliax*

- [N-20] OperatorResolver.areAddressesImported doesn't check lengths of argument arrays *Submitted by hyh*

Gas Optimizations (50)

- [G-01] use msg.sender rather than _msgSender() in FeeSplitter.receive *Submitted by TomFrench*
- [G-02] NestedFactory: _transferToReserveAndStore can be simplified to save on gas *Submitted by PierrickGT*
- [G-03] Save gas by caching array length used in for loops *Submitted by 0x0x0x, also found by pants, xYrYuYx, gzeon, WatchPug, defsec, and pauliax*
- [G-04] For `uint` replace `> 0` with `!= 0` *Submitted by 0x0x0x, also found by defsec*
- [G-05] `updateShareholder` in `FeeSplitter.sol` can be implemented more efficiently *Submitted by 0x0x0x*
- [G-06] Reduce require messages length to save contract size *Submitted by 0xngndev, also found by GiveMeTestEther, gzeon, WatchPug, pauliax, and yeOlde*
- [G-07] unchecked { ++i } is more gas efficient than i++ for loops *Submitted by GiveMeTestEther, also found and pants*
- [G-08] FlatOperator can be inlined into NestedFactory to save gas *Submitted by TomFrench*
- [G-09] `NestedReserve.transferFromFactory` function increases deployment gas costs unnecessarily *Submitted by TomFrench*
- [G-10] More gas efficient calculation of weights *Submitted by GiveMeTestEther, also found by WatchPug*
- [G-11] Mix of external and public function visibility with the same access modifier *Submitted by GiveMeTestEther, also found by defsec*
- [G-12] Move from a pull to a push pattern for sending fees to the FeeSplitter *Submitted by TomFrench, also found by pauliax*
- [G-13] Store hash of `type(ZeroExStorage).creationCode` rather than recalculating it on each call *Submitted by TomFrench*
- [G-14] Adding an if check to avoid unnecessary call *Submitted by 0xngndev*

- [G-15] Subtraction from `totalWeights` can be done unchecked to save gas *Submitted by GiveMeTestEther, also found by WatchPug, defsec, and pauliax*
- [G-16] NestedFactory: in deleteAsset and freeToken, tokens should only be declared once *Submitted by PierrickGT*
- [G-17] function mintWithMetadata() Unused *Submitted by MetaOxNull*
- [G-18] _sendFees() Repeat SLOAD shareholders In Loop *Submitted by MetaOxNull*
- [G-19] `removeFactory` has `==true` comparison in require statement *Submitted by loop, also found by palina*
- [G-20] Remove unnecessary `balanceOf` call in `NestedBuybacker::triggerForToken` *Submitted by pmerkleplant, also found by palina, nathaniel, WatchPug, and cmichel*
- [G-21] Refactor `FeeSplitter::getAmountDue` to save one variable slot *Submitted by pmerkleplant*
- [G-22] Public functions can be declared external *Submitted by palina, also found by xYrYuYx*
- [G-23] Gas-consuming way to add shareholders *Submitted by palina*
- [G-24] WETHMock withdraw function unnecessary safe math *Submitted by pants*
- [G-25] reordering struct fields *Submitted by pants*
- [G-26] double reading of state variable inside a loop *Submitted by pants*
- [G-27] Use existing memory version of state variables *Submitted by yeOlde, also found by pauliax*
- [G-28] Use `calldata` keyword instead of `memory` keyword in function arguments *Submitted by xYrYuYx*
- [G-29] Add index param to remove in function argument to reduce gas. *Submitted by xYrYuYx*
- [G-30] OperatorResolver: importOperators() function redeclares local variable multiple times *Submitted by GreyArt*
- [G-31] NestedRecords: Unnecessary variable in the Holding struct *Submitted by GreyArt*

- [G-32] MixinOperatorResolver: variables are declared multiple times in rebuildCache() *Submitted by GreyArt*
- [G-33] NestedReserve: Redundant valid token address checks *Submitted by GreyArt*
- [G-34] NestedRecords: createRecord() can be made internal *Submitted by GreyArt*
- [G-35] NestedRecords: createRecord()'s isActive check is redundant *Submitted by GreyArt*
- [G-36] NestedRecords: createRecord() can have modifier check removed *Submitted by GreyArt*
- [G-37] NestedFactory: _fromReserve param in _submitOutOrders() is redundant *Submitted by GreyArt*
- [G-38] Gas Optimization: Pack struct in FeeSplitter.sol *Submitted by gzeon*
- [G-39] Gas Optimization: Set allowance only when needed *Submitted by gzeon*
- [G-40] Avoid unnecessary storage writes can save gas *Submitted by WatchPug*
- [G-41] `NestedFactory#removeOperator()` Avoid empty items can save gas *Submitted by WatchPug*
- [G-42] Adding unchecked directive can save gas *Submitted by WatchPug, also found by xYrYuYx*
- [G-43] Cache and read storage variables from the stack can save gas *Submitted by WatchPug*
- [G-44] Unnecessary Use of _msgSender() *Submitted by defsec*
- [G-45] Small refactor for functions to save some gas *Submitted by 0xngndev*
- [G-46] Check condition before calling NestedFactory._handleUnderSpending *Submitted by hyh*
- [G-47] index + 1 can be simplified *Submitted by pauliax, also found by GreyArt*
- [G-48] _burnNST *Submitted by pauliax*
- [G-49] mintWithMetadata onlyFactory *Submitted by pauliax*
- [G-50] Unused local variables *Submitted by yeOlde, also found by PierrickGT, pmerkleplant, WatchPug, and hack3r-0m*

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