Borderless NAP Extension Audit Report

Version 1.0

Conducted by: Alin Mihai Barbatei (ABA)

May 18, 2024

Table of Contents

1 Introduction **4**
- 1.1 About ABA . . . 4
- 1.2 Disclaimer . . . 4
- 1.3 Risk classification . . . 4
  - 1.3.1 Impact . . . 4
  - 1.3.2 Likelihood . . . 4
  - 1.3.3 Actions required by severity level . . . 4
  - 1.3.4 Informational findings . . . 5

2 Executive Summary **6**
- 2.1 Borderless NAP Extension . . . 6
- 2.2 Overview . . . 7
- 2.3 Scope . . . 7
- 2.4 Issues Found . . . 7
- 2.5 Findings & Resolutions . . . 7

3 Findings **10**
- 3.1 Critical Risk . . . 10
  - 3.1.1 Borderless NAP can be drained of LPs due to faulty minimum 200 USDC LP enforcement price . . . 10
- 3.2 High Risk . . . 11
  - 3.2.1 NAP721 deactivations are blocked when interface fee is deduced . . . 11
  - 3.2.2 Faulty NAP treasury reward calculation blocks activations . . . 12
  - 3.2.3 Treasury reward group removal always fails . . . 12
  - 3.2.4 Additional LP costs can be abused to steal the difference from the NAP . . . 13
- 3.3 Medium Risk . . . 13
  - 3.3.1 Treasury rewards for group not deleted on group removal . . . 13
  - 3.3.2 Incorrectly removing IDs from reward groups corrupts internal accounting . . . 14
  - 3.3.3 Anti price abuse premium can be subverted . . . 15
- 3.4 Low Risk . . . 15
  - 3.4.1 Uninitialized distributor will not show failure on process . . . 15
  - 3.4.2 Changing B0rder1essNAP treasury not validated for compatibility . . . 16
  - 3.4.3 Duplicated reward group names are allowed in the treasury . . . 16
  - 3.4.4 NFT IDs can exist in multiple treasury reward groups . . . 17
  - 3.4.5 Minimum activation price of 200 USDC can be abused to get discount on one bulk buy . . . 17
  - 3.4.6 B0rder1essNAP.getNFTPrice shows incorrect NFT price . . . 18
  - 3.4.7 B0rder1essNAP token deactivation does not return correct deactivation amount 18
  - 3.4.8 Safe ERC20 operation variants are not used . . . 19
  - 3.4.9 Collateral token is not enforced to USDC in the Borderless NAP . . . 19
- 3.5 Informational Findings . . . 20
  - 3.5.1 Inadequate event data . . . 20
  - 3.5.2 Missing input validation on sensitive addresses . . . 20

3.5.3 Redundant rule amount percentage 21
3.5.4 NAPs are incompatible with ERC-7631 paired NFTs 21
3.5.5 Lack of events on critical changes 21
3.5.6 Redundant or dead code 22
3.5.7 Typographical Errors 23
3.5.8 CEI pattern not respected for NAP721 activations 23

1 Introduction

1.1 About ABA

Alin Mihai BARBATEI, known as ABA, is an independent security researcher experienced in Solidity smart contract auditing. Having several solo and team smart contract security reviews, he consistently strives to provide top-quality security auditing services. He also serves as a smart contract auditor at Guardian Audits.

1.2 Disclaimer

Audits are a time, resource, and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can show the presence of vulnerabilities **but not their absence**.

1.3 Risk classification

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

1.3.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that's not so critical.

1.3.2 Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

1.3.3 Actions required by severity level

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

1.3.4 Informational findings

Informational findings will also be included. These encompass recommendations to enhance code style, operations alignment with industry best practices, gas optimizations, adherence to documentation and EIP standards, and overall contract design.

Informational findings typically have minimal impact on code functionality or security risk.

Evaluating all vulnerability types, including informational ones, is crucial for a comprehensive security audit to ensure robustness and reliability.

2 Executive Summary

2.1 Borderless NAP Extension

The Native Assurance Protocol (NAP) is a solution designed to enhance NFT valuation using liquidity backing, in other words a liquidity framework.

At its core, NAP allows users to exchange `ERC721` and `ERC1155` tokens for the backed `ERC20` class token, and vice versa. This is realized by integrating with Uniswap V2 for liquidity management and leveraging ERC20 tokens for assurance and collateral.

From a codebase point of view two new contracts where added and integrated with the existing architecture:

- `BorderlessNAPTreasury` used to facilitate premium groups
- `B0rder1essNAP` a NAP721 specifically designed with several activation and deactivation price alterations

From a feature point of view, the audited extension to the existing NAP ecosystem represents the following:

- an anti-price abuse mechanism was introduced in the `B0rder1essNAP`, where an extra fee is added for bulk buys, done in order to discourage mass activations that may be used in price manipulation
- initially a minimum activation cost of 200 USDC in LP equivalent was introduced. After carefully reviewing the implementation, issues that it brought and possible solutions (as mentioned by `C-01`) the borderless correctly decided to change the NFT activation price formula itself to take into consideration the already existing 5505 NFTs in the marketplace. Subsequently, initial activation price will be already set according to team design
- groups for NFTs were created, where the cost of some is increased. Example, a legendary NFT would cost more on activations. Premium group logic implemented in the `BorderlessNAPTreasury` contract
- the old activation/deactivation price calculation model allowed bulk buy price arbitrage and manipulation. To counter this dynamic NFT prices were introduced
- a static UI fee of 1.5% was added on deactivations. This fee is only taken if users use the Borderless UI (as indicated in the name, UI fee). More seasoned users that use the blockchain directly for operations can simply not pay this fee

More on the protocol can be found on the official documentation page. More on the changes can be found here as presented by the team.

2.2 Overview

Project Name	Borderless NAP Extension
Codebase	audit_may
Operating platform	Polygon PoS
Language	Solidity
Initial commit	dd0f12946cabf6d063967d120f5aee00b21fbc0b
Remediation commit	dc30550d7f51704539f78ae8dec1f6dcb41b5409
Audit methodology	Static analysis and manual review

2.3 Scope

Files and folders in scope

- contracts/Distributor.sol
- contracts/B01.sol
- contracts/BorderlessNAPTreasury.sol
- contracts/naps/NAP721.sol
- contracts/implementations/B0rder1essNAP.sol

2.4 Issues Found

Severity	Total Found	Resolved	Partially Resolved	Acknowledged
Critical risk	1	1	0	0
High risk	4	4	0	0
Medium risk	3	2	0	1
Low risk	9	6	1	2
Informational	8	5	1	2

2.5 Findings & Resolutions

ID	Title	Severity	Status
C-01	Borderless NAP can be drained of LPs due to faulty minimum 200 USDC LP enforcement price	Critical	Resolved
H-01	NAP721 deactivations are blocked when interface fee is deduced	High	Resolved
H-02	Faulty NAP treasury reward calculation blocks activations	High	Resolved
H-03	Treasury reward group removal always fails	High	Resolved
H-04	Additional LP costs can be abused to steal the difference from the NAP	High	Resolved
M-01	Treasury rewards for group not deleted on group removal	Medium	Resolved
M-02	Incorrectly removing IDs from reward groups corrupts internal accounting	Medium	Resolved
M-03	Anti price abuse premium can be subverted	Medium	Acknowledged
L-01	Uninitialized distributor will not show failure on process	Low	Acknowledged
L-02	Changing B0rder1essNAP treasury not validated for compatibility	Low	Resolved
L-03	Duplicated reward group names are allowed in the treasury	Low	Resolved
L-04	NFT IDs can exist in multiple treasury reward groups	Low	Acknowledged
L-05	Minimum activation price of 200 USDC can be abused to get discount on one bulk buy	Low	Resolved
L-06	B0rder1essNAP.getNFTPrice shows incorrect NFT price	Low	Resolved
L-07	B0rder1essNAP token deactivation does not return correct deactivation amount	Low	Resolved
L-08	Safe ERC20 operation variants are not used	Low	Partially Resolved
L-09	Collateral token is not enforced to USDC in the Borderless NAP	Low	Resolved
I-01	Inadequate event data	Informational	Partially Resolved
I-02	Missing input validation on sensitive addresses	Informational	Resolved
I-03	Redundant rule amount percentage	Informational	Acknowledged
I-04	NAPs are incompatible with ERC-7631 paired NFTs	Informational	Acknowledged
I-05	Lack of events on critical changes	Informational	Resolved

ID	Title	Severity	Status
I-06	Redundant or dead code	Informational	Resolved
I-07	Typographical Errors	Informational	Resolved
I-08	CEI pattern not respected for NAP721 activations	Informational	Resolved

3 Findings

3.1 Critical Risk

3.1.1 Borderless NAP can be drained of LPs due to faulty minimum 200 USDC LP enforcement price

Severity: *Critical risk (Resolved) [PoC]*

Context: ***B0rder1essNAP.sol***:*210-283*

Description

One of the new feature of the Borderless NAP is the addition of a minimum activation/deactivation price of LP equivalent to 200 USDC.

This price evaluation incorrectly uses the current exact reserves amount from the UniswapV2 pool, leaving the entire mechanism susceptible to market manipulation. Specifically an attack can drain the NAP itself of LP tokens by:

- activating N NFTs at current price paying LP1 amount
- making a flash-loan and swapping large amounts of the assurance token to USDC, in order to reduce the USDC balance of the pool, which is used in the faulty minimum price mechanism
- deactivate the N NFTs at the manipulated price, resulting in a larger amount of LPs being give back; deactivation will be done in assurance tokens
- swap back the entire initial bought USDC amount to assurance token and pay back flash-loan

The above scenario leads to NAP drain of LP, a critical issue for both users and protocol team.

Another scenario is flash-loaning USDC to increase the price equivalent price of the NFT in LP terms that is surpasses 200 USDC equivalent, do one activation, then deactivate the ID and payback the flash-loan at a slight loss. By doing this, an attacker makes the protocol set the `achievedMinimalRequiredLiquidity` flag to true, thus removing the minimum price requirement.

The faulty minimum mechanism combine with organic price fluctuations also create situations where a user that initially paid N equivalent LPs to then on deactivations extracts more LP, leaving the ending users unable to withdraw at all due to lack of LPs within the NAP.

Recommendation

In order to ensure that no price manipulation is done when using UniswapV2, there is a need for a price oracle that will provide valid price information. Depending on the assurance token used, 3rd party price oracles such as Chainlink may not be available, thus consider using a custom on-chain TWAP (Time-Weighted Average Price) oracle. The oracle should use a sliding window, multi-sampling points with at least 5 minutes of history.

As indicated by the official UniswapV2 documentation, an example of a sliding window oracle can be found here. RareSkills also provides a good explanation on how UniswapV2 TWAP oracles function. Some other suggestions for fair asset pricing on Uniswap V2 can be found here.

It is important that the price calculation on activations, calculated correctly using the above information, to not be the same logic used for deactivation cost since price may have fluctuated. Save for each individual activated ID what was the equivalent LP paid on activation and on deactivation provide that value. If you provide more then some users will get their equivalent USDC in LP back while others won't be able to get any amount out.

With the mentioned mechanism, users at least would get what they provided in LP, instead of nothing and all users would be able to exit. As speculation is natural within the cryptocurrency space, users do accept a level or risk when interacting with protocols, however, it must be clearly be stated that what those risk are.

Resolution: Resolved. Fix was implemented in **cfc43b6**, **fbe5ed1**, **b72cb66** and **dc30550**.

ABA: the borderless team removed the minimum 200 USDC LP enforcement price mechanism altogether and changed the cost formula to a simpler one that does not have the vulnerabilities in discussion

3.2 High Risk

3.2.1 NAP721 deactivations are blocked when interface fee is deduced

Severity: *High risk (Resolved)*

Context: ***NAP721.sol***:515-516

Description

When a deactivation is done in the `NAP721` contract, depending on certain conditions, 3 fees are taken:

1. treasury fee: max 1% and always taken
2. interface fee: always 1.5% and taken only if `interfaceFeeEnabled` is set and users has less B01 tokens then the discount threshold
3. deactivation fee: max 10% and always taken

The fees are deducted incorrectly from the user gross amount in the `_processDeactivation`, to be precise, if the interface fee exists, it is not removed from the amount sent to the user. Only the `treasuryFee` and deactivation fee (`feeRecipientFee`) are deduced.

```
    deactivationAmount = _deactivationProfitAmount - treasuryFee - feeRecipientFee;
    tokenOut.safeTransfer(_to, deactivationAmount);
```

When the `transfer` is executed, it will revert because of there not being enough tokens within the contract to execute the swap, thus blocking users deactivations.

Recommendation

Also deduct the `interfaceFee` fee from the amount that will be transferred to the user (`deactivationAmount`).

Resolution: Resolved. The recommended fix was implemented in **8fd9ce1**.

3.2.2 Faulty NAP treasury reward calculation blocks activations

Severity: *High risk (Resolved) [PoC]*

Context: ***BorderlessNAPTreasury.sol***:145-146

Description

When the `B0rder1essNAP` contract attempts to deduct extra price increase from the reward groups for the activations it will call the `BorderlessNAPTreasury.getRewardFor` function.

Due to a faulty implementation, this function will revert if the number of token IDs associated to the reward group is not equal to that of the total existing groups in the treasury.

The issue is with declaring the length of the found groups and rewards per group to that of the treasury existing group names length, not to the `tokenIds` length, itself.

```
    bytes32[] memory foundGroupNames = new bytes32[](groupNames.length);
    uint256[] memory rewardPerGroup = new uint256[](groupNames.length);
```

Because of the above issue, adding token IDs that are in groups to the `foundGroupNames` or `rewardPerGroup` array reverts, blocking activations.

Recommendation

Use the `tokenIds` length when initiating the `foundGroupNames` and `rewardPerGroup` arrays.

Resolution: Resolved. The recommended fix was implemented in **8650835**.

3.2.3 Treasury reward group removal always fails

Severity: *High risk (Resolved) [PoC]*

Context: ***BorderlessNAPTreasury.sol***:73

Description

In the `BorderlessNAPTreasury` contract, there is are options to add and to remove reward groups.

Removing a group is however incorrectly implemented and always reverts. This happens because after the removal of a name group, the logic attempts to send the stored reward LP tokens back to the caller (trusted owner) but fails.

The failure is due to the use of `safeTransferFrom(address(this), ...)` instead of a direct `safeTransfer`. Using `transferFrom` to transfer Uniswap V2 liquidity pair tokens token from the owner itself still checks for approval and in the current case, reverts.

Because removing a group always fails, any unwanted discount or price increases that was intended to be removed will remain, losing the protocol funds.

Recommendation

Change the LP token reimburse call from `safeTransferFrom(address(this), ...)` to a direct `safeTransfer`.

Resolution: Resolved. The recommended fix was implemented in **8650835**.

3.2.4 Additional LP costs can be abused to steal the difference from the NAP

Severity: *High risk (Resolved)*

Context: ***B0rder1essNAP.sol***:233

Description

As per documentation, *certain NFT's can be part of groups (Legendary NFT's, Game fractions, etc.). The goal is to make certain NFT's to cost more, currently we are doing that in a separate contract BorderlessNAPTreasury.sol*

This mechanism in implemented in the `BorderlessNAPTreasury` contract where the owner adds liquidity equivalent to the extra cost for the indicated token IDs. In the `B0rder1essNAP` contract, these will be reflected in the activation prices, where users pay the extra amount and in deactivations, where users get back these extra amount.

Adding liquidity reward to a group can be front run, leading to situations where an attacker can steal the extra price increase, instead of paying it.

Example front-run scenario:

- NDF IDs 1, 2, and 3 are in the NAP contract
- team decides ID 1 is rare and added a cost of activating it via `addLPToGroups`
- attacker sees the `addLPToGroups` transaction in the mempool and front-runs it with an activation of the NFT with ID 1
- after `addLPToGroups` was executed, attacker can call the deactivate NFT function to get the bonus as if he would of activated it, effectively stealing funds from the project.

Recommendation

Create a pause mechanism which will only pause NFT activations and anytime when adding LP to a group is to be done, first pause activations, add LP rewards, then unpause activations.

It is crucial that pausing the activations is in a separate transaction from adding LP rewards, otherwise both could be front-run at the same time.

Resolution: Resolved. The recommended fix was implemented in **cfc43b6**.

3.3 Medium Risk

3.3.1 Treasury rewards for group not deleted on group removal

Severity: *Medium risk (Resolved)*

Context: ***BorderlessNAPTreasury.sol***:60-77

Description

When removing a reward group from the `BorderlessNAPTreasury` contract, the reward funds for that group are sent back to the caller but the internal structure `rewardForGroup` is not cleared.

If the group is ever re-added, then it will wrongly presume that there are funds already allocated to it.

When a deactivation occurs in the `B0rder1essNAP` contract, then the execution either reverts due to lock of funds or if there are enough funds in the contract will send more LPs then intended, ultimately leading to the reward group payout to fail due to not enough funds.

Recommendation

In the `removeGroups` function of the `BorderlessNAPTreasury` contract, after sending the LP rewards for the group to the caller, clear the `rewardForGroup[groupName]` value.

Resolution: Resolved. The recommended fix was implemented in **8650835**.

3.3.2 Incorrectly removing IDs from reward groups corrupts internal accounting

Severity: *Medium risk (Resolved) [PoC]*

Context: ***BorderlessNAPTreasury.sol***:*72*

Description

In the `BorderlessNAPTreasury` contract, partial removals from the reward group are allowed and will incorrectly leave uncleaned data. Because of this, re-adding while using the same group name will reuse leftover token IDs for rewards even though the intended behavior is to only use the newly added IDs and rewards.

Example if from a group of 4 NFT IDs a `removeGroups` call is one with only 2 IDs, the other 2 will still have the group name associated with it. Re-adding the same name will lead to liquidity issues and deactivation reverts.

When a deactivation occurs in the `B0rder1essNAP` contract, then the execution either reverts due to lack of funds or, if there are enough funds in the contract, will send more LPs then intended, ultimately leading to the one reward group payout to fail due to not enough funds.

The same issue appears if the IDs that are removed from a given group name are not the ones actually belonging to the group.

Recommendation

Modify the `removeGroups` function to either block partial removals or correctly support them.

Blocking partial is done by checking if `groupIDs.length` is not equal to `amountOfItemsPerGroup[groupName]` then reverting.

Supporting partial removals is done by deleting the `amountOfItemsPerGroup[groupName]` and `rewardForGroup[groupName]` only if `groupIDs.length` is equal to `amountOfItemsPerGroup[groupName]`.

Also validate that the IDs to be removed do, in fact, belong to the respective group. This can be done by adding a require in the `for-loop` that sets the group ID to false, that checks that in fact the ID does exist. Example implementation:

```
require(groups[groupName][groupIDs[j]], "provided IDs do not belong to the indicated
 group");
```

Resolution: Resolved. Fix was implemented in **f485b71**.

ABA: the team implemented the version in which partial removals are not allowed

3.3.3 Anti price abuse premium can be subverted

Severity: *Medium risk (Acknowledged)*

Context: ***B0rder1essNAP.sol***:166-176

Description

In order to prevent mass buying an activation fee was introduce in borderless NAP. The fee is directly related to the amount of NFT activate at the same time and range from no extra fee (if only 1 NFT is activate) to 2% if more then 1000 NFTs are activated in one transaction.

This mechanism however only affects non-advanced attackers. A determined attacker can simply create an intermediary contract that activates in a loop N NFTs, one at a time, and bypass the fee, paying only the extra gas cost.

Recommendation

Adding a premium on bulk buys can be bypass in the mentioned manner regardless of implementation. Consider if the team would of implement a premium increase by how many NFTs an address activated within a specific time or block period, even in that case an attacker can simply use multiple addresses.

Completely mitigating the issue cannot be done by relying solely on one input data point. A more elaborate mechanism can be implemented in which the premium fluctuates by overall demand, e.g. if multiple activations are done within a specific time/block frame then increase the activation price.

The benefit of such an implementation are however debatable from the overhead it adds and the current implementation will deter the majority of users for attempting bulk buys. Also, implementing a demand dynamic pricing, however, a premium would also be needed to be added to the activation of once single NFT. This is not a desired action since it will deter people normal users from activating NFTs.

Thus, if such a mechanism is wanted, implement a dynamic price system based on demand as see in interval activations (which may deter users form using the protocol) while also adding a maximum allowed activations per single block. Adding a limit per block reduces the efficiency of any flash-loan based price manipulation attack.

Resolution: Acknowledged by the team.

3.4 Low Risk

3.4.1 Uninitialized distributor will not show failure on process

Severity: *Low risk (Acknowledged)*

Context: ***B01.sol***:58-63

Description

In the `B01` contract, the distributor address is initially `address(0)` and not set in the contract initializer.

As the zero address, if any token transfers are done while the distributor is not set then the event `DistributorProcessFailed` will not be sent, since a low level `call` execution on an EOA returns true.

Recommendation

Either document this behavior or also check that `distributor` is not `address(0)` before attempting the `process` call.

Resolution: Acknowledged by the team.

3.4.2 Changing B0rder1essNAP treasury not validated for compatibility

Severity: *Low risk (Resolved)*

Context: ***B0rder1essNAP.sol***:*70-72*

Description

When the address of the `B0rder1essNAP` treasury is changed via the `setNFTTreasury` function in the `B0rder1essNAP` contract, there are no checks if the newly provided NAP has the same LP token as the NAP contract and that the NAP contract, `bNAP` is set as NAP contract as well.

If any of the above mentioned conditions are not meet, further operations will fail.

Recommendation

In the `setNFTTreasury` function validate that the new treasury `lpPair` and `bNAP` addresses match that of the NAP contract, or revert otherwise.

Resolution: Resolved. The recommended fix was implemented in **8650835**.

3.4.3 Duplicated reward group names are allowed in the treasury

Severity: *Low risk (Resolved)*

Context: ***BorderlessNAPTreasury.sol***:*46-47*

Description

When adding a new group name and items to the `BorderlessNAPTreasury` contract via the `addGroups` function, by mistakenly adding the same group twice, it will lead to the `groupNames` array to have double entries.

Both of the entries will not be removed during a `removeGroups`, leading to an inflated array and, depending on further development in regards to already presented issues, possibly other issues.

Recommendation

In the `addGroups` call, before pushing the `groupName` to the `groupNames` array validate that the `amountOfItemsPerGroup[groupName]` is in fact 0, to signalize that the group has not already been added.

Resolution: Resolved. The recommended fix was implemented in **8650835**.

3.4.4 NFT IDs can exist in multiple treasury reward groups

Severity: *Low risk (Acknowledged)*

Context: ***BorderlessNAPTreasury.sol***:*37-53*

Description

When add a new treasury reward group, there is no validation if the IDs for this new group are not in another group.

Because of this, one ID may be in multiple groups benefiting from multiple bonuses at the same time.

Recommendation

If this is not intended behavior, create a mapping in which each newly added ID is set to the group it belongs to. On group removal also clean the mentioned mapping if added.

Resolution: Acknowledged by the team.

borderless: this is indented behavior

3.4.5 Minimum activation price of 200 USDC can be abused to get discount on one bulk buy

Severity: *Low risk (Resolved)*

Context: ***B0rder1essNAP.sol***:*216-232*

Description

When activating an NFT from the `B0rder1essNAP` contract, there is a minimum 200 USDC equivalent NFT cost that is enforced. While within this range, the price for all NFTs in a bulk buy is considered constant as 200 USDC LP equivalent each.

If the price did exceed 200 USDC, the new price calculation formula charges an extra anti-price abuse between 0.1% and 2% increase, depending on the number of activations and the cost of each NFT in the bulk buy increases dynamically.

An attacker may still abuse the current minimum value to actually benefit from a slight discount on a single bulk activation by activating a large amount of NFTs when the activation price approaches 200 USDC.

Example:

- NAP has NFTs of a 10,000 NFT collection and 11000 assurance LPs
- activation cost: 11000/10000 = 1.1 LPs / NFT
- hypothetical cost of one LP = 180 USDC
- the activation price at this point, in equivalent USDC is 198 USDC / NFT
- now an attacker can activate 100 NFTs at the price of $19,800 instead of $19,958 (calculated using the dynamic price variation and anti-price abuse mechanism and considering no LP value increase)

Recommendation

An ideal approach would be out of N NFTs in a bulk activation, to determine how many of them would be bellow the minimal price threshold, how many above and then calculate and the cost for each case.

Such an implementation would add significant overhead to a one instance issue as such, it is somewhat debatable about the benefits of adding it.

A compromise to take into consideration is, when the price is naturally bellow the threshold, to chose the maximum of either considering the 200 minimum USDC value, or that determined by the normal price calculation.

This variation does not completely remove the bulk discount but it reduces it, when realistically it already is not that impactful.

Resolution: Resolved. Fix was implemented in **61c6952**.

ABA: team added the anti price abuse price increase to the default 200 USDC also. This severely reduces any potential discount. Although in theory it still can happen, at this point, it will be dust amount. At a later time, the minimum USDC price was removed completely.

3.4.6 B0rder1essNAP.getNFTPrice shows incorrect NFT price

Severity: *Low risk (Resolved)*

Context: ***B0rder1essNAP.sol**:150*

Description

The `getNFTPrice` function of the `B0rder1essNAP` contract returns the price of a single NFT. It adds a premium to the price of an NFT of 0.1%.

However that premium is not taken in reality for activating 1 NFT. The 0.1% premium is added only when activating more then 1 and less then 11 NFTs at once.

External integrators or users who whish to estimate using the `getNFTPrice` function the value of an NFT will receive incorrect values.

Recommendation

Since the `getNFTPrice` returns the price of only 1 NFT, return the price without the premium of 0.1% added.

Resolution: Resolved. The recommended fix was implemented in **f485b71**.

3.4.7 B0rder1essNAP token deactivation does not return correct deactivation amount

Severity: *Low risk (Resolved)*

Context: ***B0rder1essNAP.sol**:117-135*

Description

When a deactivation is execute, users give up their NFTs in exchange for a part of their initial activation fee. The `deactivateTokens` from any NAP contract, returns thee `The total deactivation profit`.

The `deactivateTokens` function from the `B0rder1essNAP` does not include in the `deactivationProfitAmount` the amount that is gained back from the treasury, which was paid as a premium on the tokens.

The indicated behavior results in an errantly interpretation to the return value by external integrators.

Recommendation

In the `deactivateTokens` function from the `B0rder1essNAP`, also add the amount received from the treasury to the `deactivationAmount`.

Resolution: Resolved. The recommended fix was implemented in **f485b71**.

3.4.8 Safe ERC20 operation variants are not used

Severity: *Low risk (Partially Resolved)*

Context: ***BorderlessNAPTreasury.sol**:132*

Description

The `Distributor` contract inherits the `SafeERC20` but does not use the save variants for all ERC20 transfers. When calling `swapOrTransferToken` the `safeTransfer` variant should be user instead of the normal `transfer` function.

Within the `BorderlessNAPTreasury` contract, in the `transferRewardFor` function, change the call from `transfer` to a directly safe one.

Recommendation

Change all ERC20 transfers to `safe` variations.

Resolution: Partially resolved by the team in **8650835** and **aced55a**.

ABA: resolved only in the Distributor contract

3.4.9 Collateral token is not enforced to USDC in the Borderless NAP

Severity: *Low risk (Resolved)*

Context: ***B0rder1essNAP.sol**:145*

Description

When the `B0rder1essNAP` contract is deployed, the treasury and minimum NFT cost logic all work with the presumption that the collateral token will be USDC. Specifically the minimum price of an NFT activation is pegged to 200 of USDC, or to be more specific 200 units of a 6 decimal token.

If the `B0rder1essNAP` is deployed with any other collateral token then USDC, the initial price may be too low or too high as imposed by the `achievedMinimalRequiredLiquidity` logic.

Recommendation

Either enforce within the `B0rder1essNAP` contract that the `_collateralToken` is the USDC address for the respective deployed chain (Polygon PoS).

Resolution: Resolved. Fix was implemented in **b72cb66**.

ABA: issue was initially acknowledged but after the team removed the minimum USDC price equivalent, it is no longer relevant

3.5 Informational Findings

3.5.1 Inadequate event data

Severity: *Informational (Partially Resolved)*

Context: ***Distributor.sol***:*167*

Description

In the `Distributor` contract and `RulesUpdated` event is emitted when rules are set `setRules`, however this event does not provide any information.

Also in the `freeze` function of the same contract, the `FrozenState` event is emitted with a true status, but the event is only emitted when freezing, it can never be emitted with a false status not can it be emitted multiple times per contract.

Recommendation

Modify the `RulesUpdated` event to contain information about previous rules and new rules or at least about new rules.

Remove the boolean argument of the `FrozenState` event and change it to just a `Frozen` event.

Resolution: Partially resolved by the team in **f485b71**.

b0rder1ess: Partially resolved, only FrozenState changed

3.5.2 Missing input validation on sensitive addresses

Severity: *Informational (Resolved)*

Context: ***Distributor.sol***:*91-103,77*

Description

In the `Distributor` contract, the `setCollateralToken` and `setUniswapRouter` functions are missing the same zero address check that it is done when they are first set in the `initialize` function. Also, in the `initialize` function, the `_distributorAdmin` is not validated to be a non-zero address.

Recommendation

Validate that the addresses are not zero address.

Resolution: Resolved. The recommended fix was implemented in **8650835** and **aced55a**.

3.5.3 Redundant rule amount percentage

Severity: *Informational (Acknowledged)*

Context: ***Distributor.sol***:223

Description

In the `Distributor` contract, when a rule is added, several key information is saved within it. Among it there are an `amount` and a `percentage` field.

When a rule is executed in `_executeRule`, the actual amount it uses is determine as `uint256 amount = (rule.amount * rule.percentage)/ 100_000`.

Since each rule has it's own amount and percentage, it is redundant to keep a `percentage` as only a final amount is used, which is pre-determined by the two values.

There are no external factors on which to add the percentage itself, making it redundant in favor of only keeping an `amount` field.

Recommendation

Remove the `percentage` logic from the distributor rules and use the amount only.

Resolution: Acknowledged by the team.

b0rder1ess: Distributor will probably be rewritten further along, leaving it as so for now

3.5.4 NAPs are incompatible with ERC-7631 paired NFTs

Severity: *Informational (Acknowledged)*

Context: Global

Description

Currently the NAP protocol can accept any ERC721 as the NFT to be backed. A newly emerging token types are ERC-7631: Dual Nature Token Pair type tokens or, DN404 as they are more popularly known. ERC7631 type NFTs (ERC721) are paired with a base ERC20 tokens and by sending these base tokens, the NFTs are minted or burned.

These types of NFTs allow users to control the current circulating supply of the NFTs and as such are not compatible with the NAP technology where the price of activating a NAP takes into account the existing NFT total supply at that moment. A value which with ERC7631 tokens is user controlled.

Recommendation

Note the incompatibility with ERC7631 paired NFT tokens within the project documentation.

Resolution: Acknowledged by the team.

3.5.5 Lack of events on critical changes

Severity: *Informational (Resolved)*

Context: Global

Description

Throughout the protocol there are important changes that do not emit events when done.

- in the `B01` contract, add an event when calling the `setDistributorHookThreshold` function
- in the `B0rder1essNAP` contract, add an event when calling the `setFeeRecipient` and `setNFTTreasury` functions (also add address validation check)
- in the `BorderlessNAPTreasury` contract, add an event when calling the `setB01Nap` and `setLpPair` functions (also add address validation check)

Recommendation

Add the indicated events.

Resolution: Resolved. The recommended fix was implemented in **f485b71**.

3.5.6 Redundant or dead code

Severity: *Informational (Resolved)*

Context: Global

Description

Remove dead or redundant code from the project:

1. in the `B01` contract, the events `TaxUpdated` and `PoolActivationUpdated` are never emitted, removed them
2. within the `Distributor.Rule` structure there is a remnant `swapToken` element. This element is from an outdate version and is currently not used since the supported action types do not include swaps
3. the private function `_safeApproveAndSwap` from the `Distributor` contract is unused. If the function will not be used in further versions, remove it
4. the `Distributor` contract does not inherit the `IDistributor` interface. Either inherit (and use the structures from it, or remove it. Also, the imports in the `IDistributor` interface itself are unused and can be removed
5. also in the `NAP721` contract, in the `getDeactivationProfit` function the `treasuryFeePercentage` is redundantly assigned to a variable `projectFee` when `projectFee` is only used once. For that one usage, directly use the `treasuryFeePercentage`
6. in the `B0rder1essNAP` constructor the `msg.sender` is wrapped in an address cast, this is redundant as `msg.sender` already is an address (payable). Remove the casting
7. in the `B0rder1essNAP`, the `nft` variable, which is already of type `IERC721Enumerable` is redundantly casted to `IERC721Enumerable`. Remove the casting.
8. in the `B0rder1essNAP` and `NAP721` contracts, within the `for-loop` of calculating activations and deactivations, the cost can be reused and directly added to the sum without redoing the `( lpAmount +- sum)/ totalSupply` calculation
9. in the `B0rder1essNAP.calculateActivationCostFor` function, changing the `multiplier` variable default to 0 allows the removal of the `if-else` construction from the `for-loop` while keeping only the `if-true` branch code

Recommendation

Implement the mentioned changes.

Resolution: Resolved. The recommended fix was implemented in **61c6952** and **f485b71**.

3.5.7 Typographical Errors

Severity: *Informational (Resolved)*

Context: Global

Description

Within the codebase there are typographical errors:

- `B0rder1essNAP`: in the `activateTokens` function natspec: `totalActivatIionPrice` → `totalActivationPrice` (and fix variable name also)
- `BorderlessNAPTreasury`: the `reinburstLPtoGroups` function name is incorrect, change it to `reimburseLPtoGroups`. Also rename the variable `totalLpAmout` → `totalLpAmount`

Recommendation

Apply the mentioned changes.

Resolution: Resolved. The recommended fix was implemented in **aced55a**.

3.5.8 CEI pattern not respected for NAP721 activations

Severity: *Informational (Resolved)*

Context: ***NAP721.sol***:*178-185*

Description

Within NAP721 activations, transferring NFTs from the NAP contract to the users uses the `ERC721.safeTransferFrom` function witch if the caller is a contract, executes a specific callback on the calling contract to validate that it is able to handle the NFTs.

This call can give execution flow to attacker from within the execution of the NAP contract that can cause reentrancy issues. The Checks-Effects-Interactions was designed to naturally resolve the issue.

In the case of the `NAP721.activateTokens` function, the pattern is not respected as one of the effect, the setting of the `activationsCountByInterval` variable, is done after the interaction (`safeTransferFrom`).

In this case, because all the relevant NAP entry points have the `nonReentrant` modifier, they are protected but protocol is still advise to respect standard security practices.

Recommendation

Move the `activationsCountByInterval[getCurrentInterval()] += (tokenIds.length).toUint16();` line before the `for`-loop in the `NAP721.activateTokens` function.

Resolution: Resolved. The recommended fix was implemented in **aced55a**.