PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Alpha Shares

26 August 2022

paladinsec.co

info@paladinsec.co

Table of Contents

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains full rights over all intellectual property (including expertise and new attack or exploit vectors) discovered during the audit process. Paladin is therefore allowed and expected to re-use this knowledge in subsequent audits and to inform existing projects that may have similar vulnerabilities. Paladin may, at its discretion, claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for Alpha Shares on the Avalanche network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	Alpha Shares
URL	https://www.alphashares.io/
Platform	Avalanche
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
AlphaNFT	0x375c29c56B1b92aCB2eb7a1504b1315a4B81d3b4	MATCH
AlphaNodesNFTMarketPlace	Not deployed; using Marketplace V2	UNDEPLOYED
BaseAttributeManager	Abstract	MATCH
ShareNFTPowerLevelManager	0xf5a47219BCE85fb269869A6FAf6d7ECCB9144e4D	MATCH
ShareNFTVanityAttributeManager	0x3D28287D631156EFd2FE4D97CEC4c40F70C81216	MATCH
AlphaShareCreator	0xf5a47219BCE85fb269869A6FAf6d7ECCB9144e4D	MATCH
ShareStaking	0x837C334421aAF2B00Dc54D4e19ACfb94B17d670e	MATCH
ShareStakeDividendsModel	0x4Aa60d02065e638559eE0106dc6866559a9b347d	MATCH
AlphaSingleStake	0x356ea99F33E679a446903670F5E9f57bDAb92Fe0	MATCH
Auth	Dependency	MATCH
AuthContract	Dependency	MATCH
ArtistRoyaltyDistributor	0x6553462dbCFe3676053A5c75e7EAEBd242743428	MATCH
OwnerRoyaltyDistributor	0x38cB510e9E3E7CfE0576fee3B09Ad11Ac56b9593	MATCH
TreasuryRoyaltyDistributor	0x635698A16875d6c4327734b9a27cD3cbbd545986	MATCH
TransactionFeeDistributor	0xBC179E904ab5C0c069F1219fD9255a2Fd08399D2	MATCH
Marketplace V2	0x15145b0227B74750c7885ebD8d58F8B0CA832EB0	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	11	10	-	1
Medium	8	8	-	-
Low	20	17	-	3
Informational	57	46	5	6
Total	**96**	**81**	**5**	**10**

Classification of Issues

Severity	Description
High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
Medium	Bugs or issues that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 Global Issues

ID	Severity	Summary	Status
01	HIGH	Governance risks: The governance has crucial control over all aspects of the codebase	ACKNOWLEDGED
02	INFO	Inconsistent usage of `nonReentrant`	ACKNOWLEDGED
03	INFO	Missing licensing	PARTIAL

1.3.2 AlphaNFT

ID	Severity	Summary	Status
04	HIGH	`createToken` does not adhere to checks-effects-interactions pattern, which could allow a malicious party to wipe out all parameter changes (like `isForSale`) right after they mint an NFT	✓ RESOLVED
05	MEDIUM	Missing safeguards on royalty fee and alpha royalty fee allows for governance to set the NFT marketplace tax to 100%	✓ RESOLVED
06	LOW	Changing of URL should be locked after everything is minted and revealed	ACKNOWLEDGED
07	LOW	`setIsForSale` does not call `setLastUpdated`	✓ RESOLVED
08	LOW	Missing safeguards on `setAddresses`	✓ RESOLVED
09	INFO	Unused functionality	PARTIAL
10	INFO	`total` can be made `immutable`	✓ RESOLVED
11	INFO	Lack of events for various methods	PARTIAL
12	INFO	Gas optimizations	✓ RESOLVED

1.3.3 AlphaNodesNFTMarketPlace

ID	Severity	Summary	Status
13	HIGH	The checks-effects-interactions pattern not adhere to in various functions which allows for reentrancy vulnerabilities	✓ RESOLVED
14	HIGH	`collectFees` return value is too low, causing the contract to have insufficient funds to pay out the seller in certain cases	✓ RESOLVED
15	HIGH	`alphaNodeNFTNoLongerForSale` does not reset the last bid and the last bid is furthermore never refunded when the creator does not accept it	✓ RESOLVED
16	HIGH	The auction extension logic is fundamentally flawed causing the auction to be extended on any bid	✓ RESOLVED
17	HIGH	The contract does not allow genesis token 1 to be sold if the marketplace is genesis tokens only	✓ RESOLVED
18	MEDIUM	`setIsForSale` is never set to `true` when a sale starts	✓ RESOLVED
19	MEDIUM	`offerAlphaNodeNFTForSale` and `offerAlphaNodeNFTForSaleToAddress` can be called even when a sale is already active to overwrite an ongoing sale	✓ RESOLVED
20	LOW	Lack of validation to ensure the user does not overpay in direct purchases	✓ RESOLVED
21	LOW	Several important variables are private	✓ RESOLVED
22	INFO	Lack of events for various functions	✓ RESOLVED
23	INFO	Unused functionality	✓ RESOLVED
24	INFO	Various functions can be made external	✓ RESOLVED
25	INFO	Typographical errors	✓ RESOLVED
26	INFO	`WRAPPED_NATIVE`, `MAX_NFT_SUPPLY` and `nftContract` can be made immutable	✓ RESOLVED

1.3.4 BaseAttributeManager

ID	Severity	Summary	Status
27	MEDIUM	`DEFAULT_ADMIN_ROLE` not assigned	✓ RESOLVED
28	LOW	`_nftContract` and `_erc20Token` are `private`	✓ RESOLVED
29	INFO	Lack of events	✓ RESOLVED
30	INFO	Unused imports and functionality	✓ RESOLVED

1.3.5 ShareNFTPowerLevelManager

ID	Severity	Summary	Status
31	LOW	Missing safeguards on `registerNewToken`	✓ RESOLVED
32	INFO	Gas optimization	✓ RESOLVED

1.3.6 ShareNFTVanityAttributeManager

ID	Severity	Summary	Status
33	LOW	An EnumerableMap would be an adequate structure for tracking the various attributes	✓ RESOLVED
34	INFO	SafeERC20 already imported	✓ RESOLVED
35	INFO	Typographical error	✓ RESOLVED
36	INFO	Unused functionality	✓ RESOLVED
37	INFO	Lack of events for various functions	✓ RESOLVED
38	INFO	Gas optimizations	✓ RESOLVED
39	INFO	DEAD and _creationCostUpperLimit can be made constant	✓ RESOLVED

1.3.7 AlphaShareCreator

ID	Severity	Summary	Status
40	LOW	The checks-effects-interactions pattern is not adhered to in `purchaseSharesNative`	✓ RESOLVED
41	LOW	`_isNativePurchaseEnabled`, `bondExists` and `whitelistOnly` are private	✓ RESOLVED
42	INFO	Missing safeguards	✓ RESOLVED
43	INFO	Unused functionality	✓ RESOLVED
44	INFO	Gas optimization	PARTIAL
45	INFO	Typographical errors	✓ RESOLVED
46	INFO	`_powerLevelManagerContract` can be made immutable	✓ RESOLVED
47	INFO	Lack of events for various functions	✓ RESOLVED

1.3.8 ShareStaking

ID	Severity	Summary	Status
48	HIGH	Governance parameter setting is inverted causing certain parameters to never be set in the privileged setter functions	✓ RESOLVED
49	HIGH	Users will not receive the full rewards	✓ RESOLVED
50	MEDIUM	`DEFAULT_ADMIN_ROLE` not assigned	✓ RESOLVED
51	MEDIUM	`rewardsDelay` requirement is fundamentally incorrect on `unclaim`, causing it to always pass	✓ RESOLVED
52	LOW	User overpays for NFTs that do not have amounts to claim	✓ RESOLVED
53	LOW	Adjusting of reward rate affects users rewards retroactively	ACKNOWLEDGED
54	LOW	`_initialBaseNFTCost` is private	✓ RESOLVED
55	LOW	`claimAll()` will revert if `_rewardsDelay` time has not passed	✓ RESOLVED
56	INFO	`claimRewardsBySingleTokenId` does not charge a `claimFee`	✓ RESOLVED
57	INFO	Balance check for `rewardToken` excludes equal case	✓ RESOLVED
58	INFO	Several functions can be made external	✓ RESOLVED
59	INFO	Unused functionality	✓ RESOLVED
60	INFO	Typographical errors	✓ RESOLVED
61	INFO	Lack of validation	✓ RESOLVED
62	INFO	Lack of events for various methods	✓ RESOLVED
63	INFO	Gas optimizations	PARTIAL

1.3.9 ShareStakeDividendsModel

ID	Severity	Summary	Status
64	MEDIUM	`DEFAULT_ADMIN_ROLE` is not assigned	✓ RESOLVED
65	LOW	Missing safeguards on `setFeeReciever`	✓ RESOLVED
66	INFO	Unused and unnecessary functionality	PARTIAL
67	INFO	Lack of events for various functions	✓ RESOLVED
68	INFO	Typographical errors	✓ RESOLVED

1.3.10 AlphaSingleStake

ID	Severity	Summary	Status
69	HIGH	Users will potentially lose their pending rewards when updateRewards is called and the final rewards will be locked in the contract indefinitely	✓ RESOLVED
70	MEDIUM	Lack of validation during `updateRewards`	✓ RESOLVED
71	LOW	`deposit` could be vulnerable to reentrancy and does not support tokens with a fee on transfer	✓ RESOLVED
72	INFO	Typographical errors	✓ RESOLVED

1.3.11 Auth and AuthContract

ID	Severity	Summary	Status
73	LOW	The previous owner still keeps the authorization status after `transferOwnership`	✓ RESOLVED
74	LOW	`owner` is internal within the Auth contract	✓ RESOLVED
75	INFO	`authorize`, `unauthorize` and `transferOwnership` can be made external	✓ RESOLVED
76	INFO	Unnecessary modifier: `payable`	✓ RESOLVED
77	INFO	There is no easy way for users to iterate over the list of authorized wallets	✓ RESOLVED
78	INFO	Typographical errors	✓ RESOLVED
79	INFO	Lack of indexing for event parameters	✓ RESOLVED
80	INFO	Lack of events for `authorize` and `unauthorize`	✓ RESOLVED

1.3.12 ArtistRoyaltyDistributor, OwnerRoyaltyDistributor and TreasuryRoyaltyDistributor

No issues found.

1.3.13 TransactionFeeDistributor

ID	Severity	Summary	Status
81	HIGH	`DEFAULT_ADMIN_ROLE` is not assigned	✓ RESOLVED
82	LOW	`deposit` could be used to execute a sandwich attack	✓ RESOLVED
83	INFO	Lack of validation	✓ RESOLVED
84	INFO	`rewardDurationInBlocks` can be made constant	✓ RESOLVED
85	INFO	Fee default variables are immediately overwritten	✓ RESOLVED
86	INFO	Unused variable and pausable logic	✓ RESOLVED
87	INFO	Typographical errors	✓ RESOLVED
88	INFO	Unnecessary modifier: payable	✓ RESOLVED
89	INFO	Lack of events for various functions	✓ RESOLVED
90	INFO	Gas optimizations	✓ RESOLVED

1.3.14 Marketplace V2

ID	Severity	Summary	Status
91	LOW	The contract does not support tokens with a fee on transfer	ACKNOWLEDGED
92	INFO	`processSale` might revert if `feeRecipient` is an EOA or an incompatible contract	ACKNOWLEDGED
93	INFO	Governance privileges: pause and setGracePeriod	ACKNOWLEDGED
94	INFO	Genesis auction can be longer than desired	ACKNOWLEDGED
95	INFO	`collectFees` could revert if `baseFee` and `royaltyFee` exceed 100%	ACKNOWLEDGED
96	INFO	Seller can always revoke his approval	ACKNOWLEDGED

2 Findings

2.1 Global Issues

The issues listed within this section are global issues that apply to the entire codebase.

2.1.1 Issues & Recommendations

Issue #01	**Governance risks: The governance has crucial control over all aspects of the codebase**
Severity	HIGH SEVERITY
Description	The Alpha Shares codebase has several governance risks: almost all critical variables throughout the contracts can be changed by the governance. If the private keys of an admin account are ever stolen or the team turns malicious, this would likely result in the loss of all value in the codebase.
Recommendation	Consider eventually carefully designing a governance setup with for example a multisig and/or timelock. In the short term, the client could consider undergoing a KYC program as this significantly reduces the likelihood of malpractice (all though it does not affect the risk of key-theft).
Resolution	ACKNOWLEDGED

Issue #02	**Inconsistent usage of `nonReentrant`**
Severity	INFORMATIONAL
Description	Presently the client adds the `nonReentrant` modifier to some functions but not to others. This seems quite inconsistent to us as they are almost always exclusively added on the least crucial functions within this codebase: the privileged ones.
Recommendation	Consider adding `nonReentrant` to all external/public functions.
Resolution	ACKNOWLEDGED

Issue #03	Missing licensing
Severity	INFORMATIONAL
Description	Throughout the codebase, some contracts are missing the SPDX-License-Identifier definition or have the Identifier as Unlicensed. Even though the contracts are public on the blockchain, it is good to have correct licensing on them. A correct license can be selected with the help of https://choosealicense.com/.
Recommendation	Consider adding the proper license identifier.
Resolution	PARTIALLY RESOLVED

2.2 AlphaNFT

The ALPHANFT contract is an ERC721 NFT token, "Alpha Shares", which can be minted exclusively by the linked minter contract, which is the AlphaShareCreator described in this report.

There are four types of Alpha Shares with the following supply (from uncommon to common):

1. Apex: 10 NFTs
2. Ronin: 47 NFTs
3. Alpha: 100 NFTs
4. Base: 9843 NFTs

Each of these NFTs can be minted through the `AlphaShareCreator` contract using one of the create methods in this contract.

Each Alpha Shares NFT has various attributes. The most important attributes are its type (eg. Apex or Ronin), its name (initially empty and must be set through a management contract) and the total amount paid.

Authorized governance addresses can pause Alpha Shares minting via the `setPaused` function. Transfers can be limited to marketplace purchase/sales by authorized governance addresses via the `setAllowNonAlphaMarketplaceTransfers` function.

Before and after any token transfer, various interactions are made with the `ShareStakeDividendsModel` and `ShareStaking` contracts to update the staking NFTs and rewards based on the NFT power level.

The contract finally supports the ERC-2981 standard, which indicates to NFT exchanges that the contract creator, in this case one of the transaction fee splitters, should receive a share of each sale cost. These shares are distributed using the

`TransactionFeeSplitter` to various sections of the system. More about this distribution can be read in the `TransactionFeeSplitter` section of this report. In case the sales were made on an NFT marketplace other than the Alpha Shares NFT marketplace, an additional fee is levied.

2.2.1 Privileges

The following functions can be called by the various privileged roles of the contract:

- `setAddresses [ super admin ]`
- `createApexNFT [ minter ]`
- `createRoninNFT [ minter ]`
- `createAlphaNFT [ minter ]`
- `createBaseNFT [ minter ]`
- `setBaseURI [ super admin ]`
- `setIsForSale [ market place ]`
- `changeMinter [ super admin ]`
- `changeMarketplaceAddress [ super admin ]`
- `updateName [ authorized addresses ]`
- `updateTotalPaid [authorized addresses ]`
- `setSuperAdmin [ owner ]`
- `logContractCreated [ owner ]`
- `setPaused [ authorized ]`
- `setRoyaltyFees [ authorized addresses ]`
- `setShareStakeDividendModelAddress [ authorized addresses ]`
- `setPowerLevelManager [ authorized addresses ]`
- `setShareStakingAddress [ authorized addresses ]`
- `setAllowNonAlphaMarketplaceTransfers [ authorized addresses ]`
- `authorize`
- `unauthorize`
- `transferOwnership`
- `renounceOwnership`

2.2.2 Issues & Recommendations

Issue #04	**createToken does not adhere to checks-effects-interactions pattern, which could allow a malicious party to wipe out all parameter changes (like isForSale) right after they mint an NFT**
Severity	HIGH SEVERITY

Description	The createToken function is used to mint new tokens for an address. This method uses the safeMint method which triggers a hook on the receiving address to check if the receiver is a valid receiver of ERC-721 tokens.
	
	The createToken does not adhere to the checks-effects-interactions pattern, a well-known pattern in Solidity that lowers the chances for re-entrancy attacks, as the contract state is updated after the external call (safeMint in this case).
	
	Lines 250-258
	`alphaNodes[newItemId] = AlphaNFTAttributes({`
	`name: "",`
	`nftType: _nftType,`
	`totalPaid: 0,`
	`created: block.timestamp,`
	`updated: block.timestamp,`
	`lastOwnershipTransfer: block.timestamp,`
	`isForSale: false`
	`});`
	
	In this case when the mint is made, the exploiter could potentially enlist the token for sale, for example. Afterward, the isForSale boolean would be forced to false again while it was never sold. This is just an example of why the current order is undesirable.

Recommendation	Consider moving the _safeMint call to the end of the function as the last action before the return.
Resolution	RESOLVED The recommendation has been implemented.

Issue #05	**Missing safeguards on royalty fee and alpha royalty fee allows for governance to set the NFT marketplace tax to 100%**
Severity	MEDIUM SEVERITY
Description	The AlphaNFT contract implements the ERC-2981 standard. This standard allows contracts, such as NFTs that support ERC-721 and ERC-1155 interfaces, to signal a royalty amount to be paid to the NFT creator or rights holder every time the NFT is sold or re-sold. This is intended for NFT marketplaces that want to support the ongoing funding of artists and other NFT creators. These royalty amounts are used within the Alpha Marketplace as well. The privileged `setRoyalties` and `setAlphaTransactionRoyalties` functions set the royalty fee and alpha royalty fee information for every NFT that is part of the AlphaNFT contract. Currently, there are no safeguards that prevent the owner from setting these fees to 100% (or higher). Even though this does not affect the contract itself, it can be a problem when the NFTs are traded on different marketplaces that support this standard. A fee that can be set up to 100% poses a problem because if users decide to trade these NFTs, they might wind up giving the complete sale value to the governance of the NFT. There is no reason why any user would want this. Paladin strongly believes that explicitly capping these fees would boost user confidence. This validation is finally also lacking on the `setDenominator` function.
Recommendation	Consider merging `setRoyalties`, `setAlphaTransactionRoyalties` and `setDenominator` into a single function. The sum of the two fees should not exceed a percentage of the denominator.
Resolution	RESOLVED The three functions have been merged into one and the total fee is capped to 1/8th.

Issue #06	**Changing of URL should be locked after everything is minted and revealed**
Severity	LOW SEVERITY
Description	As NFTs are non-fungible tokens, after everything is minted and revealed, the changing of `baseURL` should be locked to preserve the non-fungible state of the token.
Recommendation	Consider introducing a locking mechanism on changing `baseUrl` that can be triggered once everything has been revealed and minted.
Resolution	ACKNOWLEDGED

Issue #07	**`setIsForSale` does not call `setLastUpdated`**
Severity	LOW SEVERITY
Description	Throughout the contract, whenever an existing `alphaNodes` is updated, the updated field from `AlphaNFTAttributes` struct is updated to mark the last time when this particular node was updated. Currently, this update is missing from the `setIsForSale` function.
Recommendation	Consider calling the `setLastUpdated` function within `setIsForSale`.
Resolution	RESOLVED

Issue #08	Missing safeguards on `setAddresses`
Severity	LOW SEVERITY
Description	The `setAddresses` function is used to initialize critical addresses used inside the contract. Currently, there is no check if any of these addresses is `address(0)` which can cause the protocol to malfunction or lose important funds.
Recommendation	Consider adding `address(0)` checks.
Resolution	RESOLVED

Issue #09	Unused functionality
Severity	INFORMATIONAL
Description	Throughout the contract, there are several functionalities that are unused or redundant: - `_wrappedNative` - `SafeERC20 import` - `_tokenURIs` - `event tokeURISet` - `event attributeDeleted` - `event tokenOwnershipChanged` - `function logContractCreated` - `modifier tokenIDInRange` Different functionality defined in a contract but not used within said contract could confuse third-party auditors. They also increase the contract length and bytecode size unnecessarily.
Recommendation	Consider removing the functionalities to keep the contract short and simple.
Resolution	PARTIALLY RESOLVED Most of this functionality has been removed.

Issue #10	`total` can be made `immutable`
Severity	INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the immutable keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the variable explicitly immutable. This variable can even be made `constant` by moving `total` directly to the top of the contract.
Resolution	RESOLVED

Issue #11	Lack of events for various methods
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications. The following functions that should emit an event: - `setDenominator` - `setAllowNonAlphaMarketplaceTransfers` - `setShareStakingAddress` - `setShareStakeDividendModelAddress` - `setPowerLevelManager` - `setAlphaTransactionRoyalties` - `setRoyalties` - `setPaused` - `setSuperAdmin`
Recommendation	Add events for the above functions.
Resolution	PARTIALLY RESOLVED Some of these functions now emit events.

Issue #12	Gas optimizations
Severity	INFORMATIONAL

Description	Throughout the contract we discovered some gas optimizations that can be applied:
	
	Line 46
	`bool public _allowNonAlphaMarketplaceTransfers = false;`
	
	The assignment to false can be removed as default assignment is false.
	
	Lines 95-102
	`struct AlphaNFTAttributes { string name; uint8 nftType; uint256 totalPaid; uint256 created; uint256 updated; uint256 lastOwnershipTransfer; bool isForSale; }`
	
	If `isForSale` is moved underneath `nftType`, less gas will be used when retrieving `AlphaNFTAttributes` as `nftType` and `isForSale` will be packed variables.

Line 332

```
emit boolAttributeChanged(
            tokenId,
            "isForSale",
            previousMarketStatus,
            alphaNodes[tokenId].isForSale
);
```

Inside the emitted event, `alphaNodes[tokenId].isForSale` can be replaced by the `isForSale` variable.

Line 344

```
require(newAddress == address(newAddress), "Invalid
address");
```

The `require` check on line 344 is redundant

Recommendation	Consider implementing the above recommendations to optimize gas usage.
Resolution	RESOLVED

2.3 AlphaNodesNFTMarketPlace

`AlphaNodesNFTMarketPlace` is an NFT marketplace that allows for the sale, bidding and purchasing of Alpha Shares NFT tokens.

The key functions of the contract are as follows:

- `transferAlphaNodeNFTWithoutPayment` allows the NFT owner to transfer the NFT to another user when it's not for sale. By default, normal NFT transfers are not allowed, so this function can be used to still pass on the NFT.
- `alphaNodeNFTNoLongerForSale` allows the user to cancel the sale of their NFT. This is only allowed once the auction duration has expired.
- `offerAlphaNodeNFTForSale` allows the user to offer their Alpha Shares NFT for sale. The user can choose the duration of their sale and whether or not it should follow the auction format. Additionally, the user can specify a price (or minimum price in case of an auction) and how much value each bid must outbid the previous bid by.
- `offerAlphaNodeNFTForSaleToAddress` is almost identical to the previous function, but only allows for a direct-sale (no auction) and locks in the sale to a specific purchaser. This means that no one else except for a predetermined account can purchase the NFT.
- `buyalphaNodeNFT` allows for anyone to purchase an NFT that is not being auctioned (it has a direct purchase price) and transfers the NFT to the bidding user.
- `enterBidForNFT` allows anyone to bid on an NFT that is being auctioned.
- `acceptBidForalphaNodeNFT` allows the auctioning party to accept a bid by a user and transfers the NFT to the bidding user.

2.3.1 Privileges

The following functions can be called by the owner of the contract:

- `setTransactionFeeDistributorAddress`
- `setGenisisFeeDistributorAddress`
- `setAntiSnipTimer`
- `setGenesisRequirements`
- `setPaused`
- `setIsGenesisAuction`
- `setMaxAuctionTimeInHours`
- `setShareStakingAddress`
- `setFees`
- `transferOwnership`
- `renounceOwnership`

2.3.2 Issues & Recommendations

Issue #13	**The checks-effects-interactions pattern not adhere to in various functions which allows for reentrancy vulnerabilities**
Severity	HIGH SEVERITY

Description	Throughout the contract, we noticed various functions that do not adhere to the checks-effects-interactions pattern, which can lead to re-entrancy attacks.
	
	Line 448
	`IALPHANFT(nftContract).safeTransferFrom(seller, bid.bidder, tokenId);`
	
	Consider moving `transfer` to be the last action in the `acceptBidForalphaNodeNFT` function.
	
	Line 339
	`WRAPPED_NATIVE.safeTransfer(_msgSender(), bid.value);`
	
	Consider moving this `safeTransfer` call on line 440.
	
	Line 327
	`IALPHANFT(nftContract).safeTransferFrom(seller, _msgSender(), tokenId);`
	
	Line 341
	`WRAPPED_NATIVE.safeTransfer(_msgSender(), bid.value);`
	
	Line 169
	`WRAPPED_NATIVE.safeTransfer(to, bid.value);`
	
	Line 546
	`WRAPPED_NATIVE.safeTransfer(royaltyFeeAddress, royaltyFee);`

	Line 400
	`WRAPPED_NATIVE.safeTransfer(existingBid.bidder, existingBid.value);`
	
	Line 408
	`WRAPPED_NATIVE.safeTransferFrom(_msgSender(), address(this), amount);`
	
	The above sections (especially the `safeTransferFrom` calls on the NFTs) could allow for reentrancy vectors. It should be noted that the transfer on line 546 will be more difficult to write in checks-effects-interactions as it is within a function used internally.

Recommendation	Consider adhering to the checks-effects-interractions pattern as much as possible, furthermore, consider adding a nonReentrant modifier on all public/external methods. Due to the complex nature of the external calls (eg. in `alphaNodeNFTNoLongerForSale` and `collectFees`), we understand that the client might desire to simply add `nonReentrant` modifiers to all functions.
Resolution	RESOLVED The client moved to a new NFT marketplace contract.

Issue #14	**`collectFees` return value is too low, causing the contract to have insufficient funds to pay out the seller in certain cases**
Severity	HIGH SEVERITY

Description	Line 547
	`_marketPlaceTransactionFee -= royaltyFee;`
	
	The `collectFees` function is supposed to return how much the seller can still withdraw from the contract in funds.
	
	`return (paymentReceived - _marketPlaceTransactionFee);`
	
	However, as the `marketplaceTransactionFee` is reduced on line 547, this return value can be too high in certain cases. This causes the seller to be eligible to withdraw too many funds from the contract. The contract might not be able to pay out all sellers because of this.

Recommendation	Consider fixing the logic flow to not incorporate the `royaltyFee` subtraction in the return value.
Resolution	RESOLVED The client moved to a new NFT marketplace contract.

Issue #15	**`alphaNodeNFTNoLongerForSale` does not reset the last bid and the last bid is furthermore never refunded when the creator does not accept it**
Severity	HIGH SEVERITY
Description	The `alphaNodeNFTNoLongerForSale` function does not reset the last bid details causing the bidder's funds to be locked in the contract. The contract also refunds any bid when it is outbid. However, if the bid is the highest bid and it never gets outbid, the bidder never receives a refund, even if the auction ends and the owner does not accept the highest bid.
Recommendation	Consider resetting the latest bid when `alphaNodeNFTNoLongerForSale` is called and refund the bidder. Consider allowing the highest bid to force finalization of the auction after it has ended.
Resolution	RESOLVED The client moved to a new NFT marketplace contract.

Issue #16	The auction extension logic is fundamentally flawed causing the auction to be extended on any bid
Severity	HIGH SEVERITY
Location	Line 404 `if ( (block.timestamp - startTime) < (auctionLength + antiSnipeTimer) ) {`
Description	As the logic above is fundamentally incorrectly written, the auction is not only extended when a bid is made in the last 5 minutes, but is always extended when a bid is made. The offer is also of type `memory`, which makes the attempt to save the extension futile. It needs to be `storage` for the save to work.
Recommendation	Consider fixing the above logic. `if (block.timestamp + antiSnipeTimer > startTime + auctionLength)`
Resolution	RESOLVED The client moved to a new NFT marketplace contract.

Issue #17	The contract does not allow genesis token 1 to be sold if the marketplace is genesis tokens only
Severity	HIGH SEVERITY
Location	Line 639 `require(nftContract.getNFTTypeByTokenId(tokenId) != 1, "Not genesis");`
Description	The contract has a flag to allow for only genesis tokens to be sold. However, due to an error in the requirement above, the token type "1" cannot be sold as well, even though it is genesis.
Recommendation	Consider removing the above requirement.
Resolution	RESOLVED The client moved to a new NFT marketplace contract.

Issue #18	`setIsForSale` is never set to true when a sale starts
Severity	MEDIUM SEVERITY
Description	The `setIsForSale` function is never called when a sale starts. It is only set to `false` whenever a sale ends.
Recommendation	Consider also calling `setIsForSale` with the `true` parameter when sales start in the `offerAlphaNodeNFTForSale` and `offerAlphaNodeNFTForSaleToAddress` functions.
Resolution	RESOLVED The client moved to a new NFT marketplace contract.

Issue #19	**`offerAlphaNodeNFTForSale` and `offerAlphaNodeNFTForSaleToAddress` can be called even when a sale is already active to overwrite an ongoing sale**
Severity	MEDIUM SEVERITY
Description	The `offerAlphaNodeNFTForSale` and `offerAlphaNodeNFTForSaleToAddress` functions can be called even when a sale is already active to overwrite an ongoing sale. This could cause an ongoing sale to suddenly have different parameters from what users expect.
Recommendation	Consider checking that no sale is presently active for the NFT. The logic of these two functions could also be combined into a single internal function in line with the "Don't Repeat Yourself" principle.
Resolution	RESOLVED The client moved to a new NFT marketplace contract.

Issue #20	**Lack of validation to ensure the user does not overpay in direct purchases**
Severity	LOW SEVERITY
Location	Line 312 `require(amount >= offer.minValue, "Offer not high enough.");`
Description	The `buyalphaNodeNFT` function allows the user to overpay for an NFT. This requirement should be `==`.
Recommendation	Consider making the requirement `==`.
Resolution	RESOLVED The client moved to a new NFT marketplace contract.

Issue #21	Several important variables are private
Severity	LOW SEVERITY
Description	Important variables that third-parties might want to inspect should be marked as public so that these third-parties can easily inspect them through the explorer, web3 and derivative contracts. The following functions are currently private: - `paused` - `isGenesisAuction` - `genesisERC20TokenAddressRequired` - `amountERC20TokenRequired`
Recommendation	Consider marking the above variables as public.
Resolution	RESOLVED The client moved to a new NFT marketplace contract.

Issue #22	Lack of events for various functions
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications. The following functions lack events: - `setTransactionFeeDistributorAddress` - `setGenisisFeeDistributorAddress` - `setAntiSnipTimer` - `setGenesisRequirements` - `setPaused` - `setIsGenesisAuction` - `setMaxAuctionTimeInHours` - `setShareStakingAddress` - `setFees` - `userWithdrawal`
Recommendation	Add events for the above functions.
Resolution	RESOLVED The client moved to a new NFT marketplace contract.

Issue #23	Unused functionality
Severity	INFORMATIONAL

Description	Throughout the contract, we've identified multiple functionalities that are unused or redundant:
	
	Lines 3,4,6
	`import "@openzeppelin/contracts/token/ERC721/ERC721.sol";`
	`import "@openzeppelin/contracts/token/ERC721/IERC721.sol";`
	`import "@openzeppelin/contracts/utils/Context.sol";`
	
	The above imports are unnecessary.
	
	`_safeTransferNative` is not used and `payable` is unnecessary across the contract.
	
	Line 161
	`IALPHANFT(nftContract).setIsForSale(false, tokenId);`
	
	This is already called within `alphaNodeNFTNoLongerForSale`.
	
	Line 308, 386, 628
	`require(tokenId <= MAX_NFT_SUPPLY, "No alphaNodeNFT");`
	
	This `require` is unnecessary as `nftContract.ownerOf` already checks if the `tokenId` exists.
	
	Unused events:
	
	- `alphaNodeNFTOfferedUpdated`
	- `alphaNodeNFTBidWithdrawn`
	
	Throughout the contract, the `nftContract` variable is unnecessarily casted to `IALPHANFT`. This variable is already of `IALPHANFT` type.
	
	`IShareStaking` interface and `SHARE_STAKING_ADDRESS` related functionality are unused.

Recommendation	Consider fixing the above issues to keep the contract short and simple.
Resolution	RESOLVED The client moved to a new NFT marketplace contract.

Issue #24	**Various functions can be made external**
Severity	INFORMATIONAL
Description	Throughout the contract, we've identified various functions that can be made external: - `transferAlphaNodeNFTWithoutPayment` - `offerAlphaNodeNFTForSale` - `offerAlphaNodeNFTForSaleToAddress` - `buyalphaNodeNFT` - `userWithdrawal` - `setFees` Functions that are not used within the contract but only externally can be marked as such with the external keyword. Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases.
Recommendation	Consider marking the functions mentioned above as external.
Resolution	RESOLVED The client moved to a new NFT marketplace contract.

Issue #25	Typographical errors
Severity	INFORMATIONAL

Description	The contract contains several typographic mistakes which we have consolidated below in a single issue in an effort to keep the report size reasonable.
	
	Line 227
	`require(auctionLength < maxAuctionTimeInHours, "Auction is too long");`
	
	This should be a <= statement.
	
	Line 264
	`require(auctionLength < maxAuctionTimeInHours, "Too long");`
	
	This should be a <= statement.
	
	Line 288
	`buyalphaNodeNFT`
	
	The name of the function should be buyAlphaNodeNFT.
	
	Line 304
	`// auction length in hours`
	
	The auction length is in fact in seconds here.
	
	Line 421
	`acceptBidForalphaNodeNFT`
	
	The name of the function should be `acceptBidForAlphaNodeNFT`.
	
	Line 490
	`function setGenisisFeeDistributorAddress`
	
	The spelling should be genesis.

Recommendation	Consider fixing the typographical errors.

Resolution	RESOLVED The client moved to a new NFT marketplace contract.

Issue #26	**`WRAPPED_NATIVE`, `MAX_NFT_SUPPLY` and `nftContract` can be made `immutable`**
Severity	INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the immutable keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the variables explicitly immutable.
Resolution	RESOLVED The client moved to a new NFT marketplace contract.

2.4 BaseAttributeManager

`BaseAttributeManager` is the base contract inherited by the `AlphaNFTPowerLevelManager` and `AlphaVanityAttributeManager` contracts. It defines some common functionality and parameters for both of these contracts.

2.4.1 Privileges

The following functions can be called by the owner of the contract:

- `setERC20Contract`
- `setFeeReceiver`
- `setNFTContract`
- `setMinter`
- `grantRole`
- `revokeRole`
- `renounceRole`
- `grantAdminRole`

2.4.2 Issues & Recommendations

Issue #27	`DEFAULT_ADMIN_ROLE` not assigned
Severity	MEDIUM SEVERITY
Description	The `BaseAttributeManager` contract uses the `AccessControl` library from OpenZeppelin, a known pattern that gives Role-Based Access capabilities to a contract. The `AccessControl` contract has defined a `DEFAULT_ADMIN_ROLE` role which has the power to grant/revoke roles to/from different recipients. By default, this role is not assigned to any address, and usually it should be assigned in the constructor to `msg.sender` or to a particular address. The downside of not setting a `DEFAULT_ADMIN_ROLE` is that there is no way for the governance to revoke any roles for privileged accounts, except with the accounts' consent.
Recommendation	Consider assigning the `DEFAULT_ADMIN_ROLE` to the `msg.sender` in the constructor or to a certain address received as parameter.
Resolution	RESOLVED

Issue #28	`_nftContract` and `_erc20Token` are `private`
Severity	LOW SEVERITY
Description	Important variables that third-parties might want to inspect should be marked as public so that these third-parties can easily inspect them through the explorer, web3 and derivative contracts.
Recommendation	Consider marking the variables as public.
Resolution	RESOLVED

Issue #29	Lack of events
Severity	INFORMATIONAL
Description	Throughout the contracts we've identified various functions that should emit an event: - `setERC20Contract` - `setFeeReceiver` - `setNFTContract` - `setMinter` Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the functions.
Resolution	RESOLVED

Issue #30	Unused imports and functionality
Severity	INFORMATIONAL

Description	Lines 5-11
	`import "@openzeppelin/contracts/token/ERC721/IERC721.sol";`
	`import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";`
	`import "@openzeppelin/contracts/token/ERC721/ERC721.sol";`
	`import "@openzeppelin/contracts/utils/Counters.sol";`
	`import "../interfaces/IALPHANFT.sol";`
	
	Line 29
	`function _init() internal virtual {}`
	
	Files that are imported in a contract but not used within said contract could confuse third-party auditors. They also increase the contract length unnecessarily.

Recommendation	Consider removing the imports and functionality to keep the contract short and simple.
Resolution	RESOLVED

2.5 ShareNFTPowerLevelManager

`ShareNFTPowerLevelManager` is a registry contract that keeps track of the power level of each NFT in circulation. The governance can set a unique power level for every NFT type: Base, Alpha, Ronin and Apex. Whenever an NFT is minted of that type, it is assigned the current power level for that type indefinitely.

Once an NFT receives a specific power level, that power level never changes, even if the governance adjusts the power level for the NFT type. The new power level will adjust only newly minted NFTs.

2.5.1 Privileges

The following functions can be called by the following priviledged roles and owner of the contract:

- `registerNewToken [ minter addresses ]`
- `setStartingPower`
- `setERC20Contract`
- `setFeeReceiver`
- `setNFTContract`
- `setMinter`
- `grantRole`
- `revokeRole`
- `renounceRole`
- `grantAdminRole`

2.5.2 Issues & Recommendations

Issue #31	Missing safeguards on `registerNewToken`
Severity	LOW SEVERITY
Description	The `registerNewToken` function registers a token for a specific NFT type (`BASE`, `ALPHA`, `RONIN`, `APEX`). Currently, there are no checks that verify if the `tokenId` is actually part of that `nftType`.
Recommendation	Consider adding a `require nftContract.getNFTTypeByTokenId(tokenId)` at the beginning of the function.
Resolution	RESOLVED

Issue #32	Gas optimization
Severity	INFORMATIONAL
Description	The `getPowerLevelByNFTType` function returns the power level for specific `tokenType`. This function does a check for every token type and as a final step, it returns the `startingPower` associated with `tokenType` or `0` if `tokenType` is greater than three. This is a bit inefficient in terms of gas as for example, for `tokenType == 0` , the function still goes through all the other checks before returning the `startingPower`.
Recommendation	Consider returning the `startingPower` directly in the `if` conditions and `0` at the final in case `tokenType > 3`.
Resolution	RESOLVED

2.6 ShareNFTVanityAttributeManager

`ShareNFTVanityAttributeManager` contract is a utility registry contract that allows users to register string and number attributes with regard to their Alpha Shares NFT. Such attributes could be a name or a description.

A fee is levied on attribute changes which is burned.

2.6.1 Privileges

The following functions can be called by the owner of the contract:

- `registerNewToken`
- `setAttributeCreationCost`
- `setAttributeUpdateCost`
- `setERC20Contract`
- `setFeeReceiver`
- `setNFTContract`
- `setMinter`
- `grantRole`
- `revokeRole`
- `renounceRole`
- `grantAdminRole`

2.6.2 Issues & Recommendations

Issue #33	**An EnumerableMap would be an adequate structure for tracking the various attributes**
Severity	LOW SEVERITY
Description	The contract keeps track of key-value attributes in an array (a list). It would however make a lot more sense if these were kept track of in a mapping.
Recommendation	Consider using an enumerable mapping instead.
Resolution	RESOLVED

Issue #34	**`SafeERC20` already imported**
Severity	INFORMATIONAL
Location	`Line 5` `import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";`
Description	The `SafeERC20` import is already imported in the `BaseAttributeManager`.
Recommendation	Consider removing the import to keep the contract short and simple.
Resolution	RESOLVED

Issue #35	Typographical error
Severity	INFORMATIONAL
Location	Line 17 `_creationCostUpperLimit`
Description	This variable should be named `_costUpperLimit` as it is used for both create and update operations.
Recommendation	Consider fixing the typographical error.
Resolution	RESOLVED

Issue #36	Unused functionality
Severity	INFORMATIONAL
Description	Throughout the contract, we have identified multiple functionalities that are unused or redundant: - `struct GameAttributeStringType` - `struct GameAttributeNumericType` - `gameStringAttributes variable` - `gameNumericAttributes variable` Functionality which is defined in a contract but remain unused could confuse third-party auditors. They also increase the contract length unnecessarily.
Recommendation	Consider removing the above functionality to keep the contract short and simple.
Resolution	RESOLVED

Issue #37	Lack of events for various functions
Severity	INFORMATIONAL
Description	Throughout the contract we've identified various functions that need to emit an event: - `registerNewToken` - `setAttributeCreationCost` - `setAttributeUpdateCost` Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions.
Resolution	RESOLVED

Issue #38	Gas optimizations
Severity	INFORMATIONAL
Description	Throughout the contract, we have identified various gas optimizations: - All the `internal` and `external` functions that take a string as a parameter should define that string as `calldata` and not `memory` to save gas. - The modifier `_requireTokenOwner` on the following functions `addStringAttribute`, `addNumericAttribute` and `updateStringAttribute` can be removed as this modifier is already called in the internal methods. We however recommend keeping the modifiers on the external function and removing them from the internal functions. - Returning arrays could eventually run out of gas for `getVanityStringAttributesByTokenId` and `getVanityNumericAttributesByTokenId`. Consider using a paginated method and introducing a get length function.
Recommendation	Consider implementing the above suggestions to optimize gas usage.
Resolution	RESOLVED

Issue #39	`DEAD` and `_creationCostUpperLimit` can be made `constant`
Severity	INFORMATIONAL
Description	Variables that are never modified can be indicated as such with the constant keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the variables explicitly constant.
Resolution	RESOLVED

2.7 AlphaShareCreator

`AlphaShareCreator` is a contract where users can mint and purchase Alpha Shares NFTs. It allows for both whitelisted purchases and non-whitelisted purchases. The latter will only be allowed once governance sets the contract to "not whitelist only".

NFTs can be purchased with either the native Alpha token and AVAX, purely with AVAX or with 'bond tokens'.

Genesis NFTs (Alpha, Ronin, Apex) can be exclusively minted by authorized accounts to users.

The NFT power is immediately added to the user's power level and the NFT is furthermore immediately staked after an NFT is minted.

The governance can finally set a limit of the number of NFTs a single address can purchase per transaction and whether tokens can be purchased with AVAX or not.

2.7.1 Privileges

The following functions can be called by the owner of the contract:

- `withdraw`
- `mintGenesis`
- `registerBondERC20`
- `removeBond`
- `setGlobalMediumOfExchangeAddress`
- `setBaseShareCostERC20`
- `setBaseShareCostNative`
- `setNativeShareCost`
- `setAlphaNftContractAddress`
- `setAlphaShareNativePaymentReceiver`
- `setAlphaShareERC20PaymentReceiver`
- `setNFTVanityAttributeContractAddress`
- `setMaxSharesPerTransaction`
- `setPaused`
- `setIsNativeEnabled`
- `setSharesRemaining`
- `addWhitelistAddress`
- `setWhiteListOnly`
- `setGenesisMinting`
- `setShareStakeDividendModelAddress`
- `setShareStakingContractAddress`
- `authorize`
- `unauthorize`
- `transferOwnership`
- `renounceOwnership`

2.7.2 Issues & Recommendations

Issue #40	**The checks-effects-interactions pattern is not adhered to in `purchaseSharesNative`**
Severity	LOW SEVERITY
Location	Line 299 `_nftSharesRemaining = _nftSharesRemaining - number;`
Description	The `purchaseSharesNative` function is used to purchase BASE NFTs using AVAX. This function does not respect the checks-effects-interactions pattern, a well known pattern in Solidity that lowers the chances for re-entrancy attacks, by updating contract state after the external call.
Recommendation	Consider moving the above line before the `_safeTransferNative` call on line 293.
Resolution	RESOLVED The recommendation has been implemented.

Issue #41	**`_isNativePurchaseEnabled`, `bondExists` and `whitelistOnly` are private**
Severity	LOW SEVERITY
Description	Important variables that third-parties might want to inspect should be marked as public so that these third-parties can easily inspect them through the explorer, web3 and derivative contracts.
Recommendation	Consider marking the variables as public.
Resolution	RESOLVED

Issue #42	Missing safeguards
Severity	INFORMATIONAL
Description	Throughout the contract, we have identified different functions that lack safeguards: 1) `setAlphaShareNativePaymentReceiver` and `setAlphaShareERC20PaymentReceiver` does not check against address zero (`address(0)`) which can cause the protocol to malfunction or lose important funds. 2) The constructor does not check against address zero (`address(0)`) which can cause the protocol to malfunction or lose important funds. 3) Line 281 `msg.value >= totalCostNative` The `purchaseSharesNative` function allows the user to overpay for an NFT. This requirement should be `==`. This is also inconsistent with the other functions as the others do use an equality check.
Recommendation	Consider adding the above safeguards.
Resolution	RESOLVED

Issue #43	Unused functionality
Severity	INFORMATIONAL
Description	Throughout the contract, we have identified multiple functionalities that are unused or redundant: - `_alphaShareERC20BondPaymentReceiver` and `_maxNFTShares` variables - `rewardsPoolAddressChanged` event - `IERC721.sol` and `ERC721.sol` import - `payable` on Line 453 - The `receive()` function on line 137 seems unnecessary. Functionality defined in a contract but not used within said contract could confuse third-party auditors. They also increase the contract length and bytecode size unnecessarily.
Recommendation	Consider removing the above functionality to keep the contract short and simple.
Resolution	RESOLVED

Issue #44	Gas optimization
Severity	INFORMATIONAL

Description	Throughout the contract, we have identified various sections that can be optimized for gas usage:
	
	1)
	Line 51, 53, 55
	`bool public GENESIS_MINT = false;`
	`bool whitelistOnly = false;`
	`bool _isNativePurchaseEnabled = false;`
	
	These are assigned to false by default, therefore explicitly assigning it is unnecessary and consumes gas.
	
	2) `addWhitelistAddress` receives an array of address as buyers—this parameter should be `calldata` to save gas.
	
	3) `minterIsWhitelisted` modifier retrieves `whiteListedBuyer[buyer]` from storage three times, consider caching this into memory to save gas.
	
	4)
	Line 330
	`uint8 powerType = IALPHANFT(_alphaNftContract).getNFTTypeByTokenId( tokenId );`
	
	The external call to `_alphaNftContract` is unnecessary as `powerType` will always be equal with `nftType` which is sent as a parameter. This is not a gas optimization but will make the code simpler.

5)
Lines 196, 240, 285

```
require(
        _nftSharesRemaining >= _nftSharesRemaining -
number,
        "Can not purchase more than available."
        );
```

The `nftSharesRemaining >= _nftSharesRemaining - number` is unnecessary and incorrect and consumes unnecessary gas. Consider modifying it to `nftSharesRemaining >= number`.

6) Usage of types smaller than 256 bytes (e.g. uint8) does not save gas outside of structs. It, in fact causes more gas usage due to extra conversion operations. Consider moving to uint256 consistently.

7) `mintGenesis` if statements could use "`else if`" clauses to save gas.

Recommendation	Consider implementing the above suggestions to optimize gas usage.
Resolution	PARTIALLY RESOLVED The `_nftSharesRemaining >= _nftSharesRemaining - number` check is still present within `purchaseSharesAtBondDiscount`.

Issue #45	Typographical errors
Severity	INFORMATIONAL

Description	The contract contains a number of typographic mistakes which we have consolidated below in a single issue in an effort to keep the report size reasonable.
	
	Lines 203-206
	`require( msg.value == totalCostNative, "Insufficient gas token payment amount" );`
	
	The revert error should be *Insufficient payment amount* as defined also in `purchaseSharesFullPriceWhiteList`.
	
	Line 122
	`_alphaShareNativePaymentReceiver = payable(msg.sender);`
	
	Although `_alphaShareNativePaymentReceiver` is properly initialized, `_alphaShareERC20PaymentReceiver` is initially set to zero. We recommend initializing it as well in the constructor.

Recommendation	Consider fixing the typographical errors.
Resolution	RESOLVED

Issue #46	`_powerLevelManagerContract` can be made immutable
Severity	INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the immutable keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the variable explicitly immutable.
Resolution	RESOLVED

Issue #47	Lack of events for various functions
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications. The following functions lack events: - `withdraw` - `purchaseSharesFullPriceWhiteList` - `purchaseSharesFullPrice` - `purchaseSharesAtBondDiscount` - `purchaseSharesNative` - `mintGenesis` - `registerBondERC20` - `removeBond` - `setGlobalMediumOfExchangeAddress` - `setBaseShareCostERC20` - `setBaseShareCostNative` - `setNativeShareCost` - `setAlphaNftContractAddress` - `setAlphaShareNativePaymentReceiver` - `setAlphaShareERC20PaymentReceiver` - `setNFTVanityAttributeContractAddress` - `setMaxSharesPerTransaction` - `setPaused` - `setIsNativeEnabled` - `setSharesRemaining` - `addWhitelistAddress` - `setWhiteListOnly` - `setGenesisMinting` - `setShareStakeDividendModelAddress` - `setShareStakingContractAddress`
Recommendation	Add events for the functions.
Resolution	RESOLVED

2.8 ShareStaking

`ShareStaking` is a simple NFT-Staking contract where users can deposit different NFTs. For each NFT, the user will get a different amount of rewards, based on `_baseNFTCost * tokenStakes[tokenId].tokenPower`. For each claim, users have to pay a 0.01 AVAX `claimFee` which can be set up to 0.1 AVAX. Governance needs to supply the `rewardToken` to the contract via the `deposit` function, only then users can receive rewards for their staked NFTs (withdrawals could become impossible if there are insufficient tokens).

Only admins can stake the NFTs for the users. These admins are other contracts like the NFT itself (which automatically stakes the NFT on receipt). Withdrawals can only be done by the NFT contract. The user can either call `claimAll` or `claimByTokenId` to claim his rewards. Rewards are only claimable after the `rewardsDelay` is passed which is 1 minute by default. For both functions, the user needs enough gas tokens to cover the `claimFee`. It is important to mention that when the NFT contracts unstake an NFT for the user, they claim the rewards without a `claimFee`. Governance can change the `rewardToken`, NFT contract, `baseNFTCost`, `denominator`, `nativeClaimFee`, `feeReceiver`, `rewardsDelay` and the `powerLevelManagerContract`.

2.8.1 Privileges

The following functions can be called by the owner and other privileged roles of the contract:

- `grantAdminRole [ admins ]`
- `resetRewardsToInitialCost`
- `triggerRewardsDecay`
- `launch`
- `withdrawRewardsToken [ admins ]`
- `withdrawNative [ admins ]`
- `unstakeAndAutoClaimForOldOwner [ nft contract ]`
- `stakeNFT [ admins ]`
- `setRewardToken [ admins ]`
- `setNFTContract [ admins ]`
- `setBaseNFTCost [ admins ]`
- `setDenominator [ admins ]`
- `setNativeClaimFee [ admins ]`
- `setFeeReciever [ admins ]`
- `setRewardsDelay [ admins ]`
- `setPowerLevelContractAddress [ admins ]`
- `setPaused [ admins ]`
- `grantRole`
- `revokeRole`
- `renounceRole`

2.8.2 Issues & Recommendations

Issue #48	**Governance parameter setting is inverted causing certain parameters to never be set in the privileged setter functions**
Severity	HIGH SEVERITY

Location	Lines 588-613
	
	`function setFeeReciever(address payable feeReceiver) external requireAdmin nonReentrant {`
	`  feeReceiver = _feeReceiver;`
	`}`
	
	`function setRewardsDelay(uint256 rewardsDelay) external requireAdmin nonReentrant {`
	`  rewardsDelay = _rewardsDelay * 1 minutes;`
	`}`

Description	In both `setFeeReciever` and `setRewardsDelay`, the parameters are used without a `_`. This causes these functions to do the exact opposite of what they are supposed to: they now assign the stored value to the temporary provided value (which is never used).
Recommendation	Consider inverting these two statements.
Resolution	RESOLVED The variables have been inverted.

Issue #49	Users will not receive the full rewards
Severity	HIGH SEVERITY
Description	Within the function `getClaimableByTokenId`, the passed time for the claiming amount is calculated as such: `(time - timerFrom)` However, `timerFrom` includes the `_rewardsDelay`: `uint256 timerFrom = tokenStakes[tokenId].lastClaimTime + _rewardsDelay` Users will therefore only receive the rewards from `timerFrom` until time which does not include the `_rewardsDelay` time.
Recommendation	Consider subtracting the `_rewardsDelay` time from `timerFrom` during the final calculation: `uint256 currentClaimable = (tokenStakes[tokenId].tokenPower * _baseNFTCost * (time - (timerFrom - _rewardsDelay))) / _secondsInDay / _denominator;` This can also be resolved on the note that the waiving of rewards over the `rewardsDelay` period is desired.
Resolution	RESOLVED The client has mitigated this by adding delay again as was recommended.

Issue #50	`DEFAULT_ADMIN_ROLE` not assigned
Severity	MEDIUM SEVERITY
Description	`ShareStaking` contract uses the `AccessControl` library from OpenZeppelin, a known pattern that gives Role-Based Access capabilities to a contract. The `AccessControl` contract has defined a `DEFAULT_ADMIN_ROLE` role which has the power to grant/revoke roles to/from different recipients. By default, this role is not assigned to any address, and usually it should be assigned in the constructor to msg.sender or to a particular address. This causes the governance to be unable to revoke any roles from other contracts and wallets.
Recommendation	Consider assigning the `DEFAULT_ADMIN_ROLE` to the `msg.sender` in the constructor or to a certain address received as parameter or move to Ownable implementation as currently the contract has just one role defined.
Resolution	RESOLVED The client has moved to the AuthContract.

Issue #51	**rewardsDelay requirement is fundamentally incorrect on unclaim, causing it to always pass**
Severity	MEDIUM SEVERITY
Description	Line 311-314 `require( tokenStakes[tokenStakesByAddress[_msgSender()][i].tokenId].lastClaimTime < block.timestamp + _rewardsDelay, "No can do, pal." )` The requirement within `claimAll` always passes due to it adding `_rewardsDelay` on the wrong side.
Recommendation	The requirement within `claimAll` should put `_rewardsDelay` on the left hand side.
Resolution	RESOLVED The client moved this into the `getClaimableByTokenId` function.

Issue #52	User overpays for NFTs that do not have amounts to claim
Severity	LOW SEVERITY

Description	The `claimAll` function iterates through all the tokens a user staked. For each token, it does a check to see if any amount claimable exists and adds that to the amountClaimable variable.
	
	After this for-loop, if the _nativeClaimFee is greater than 0, meaning that the user has to pay a fee in AVAX for each nft to claim, a check is done to see if the user has sent enough AVAX for claiming.
	
	Line 329-332
	`require(`
	`msg.value == totalClaimFee,`
	`"Insufficient native for claim"`
	`);`
	
	The issue is that this implementation also accounts for the NFTs that have 0 claimable amount, as `totalClaimFee` is calculated based on how many tokens the user has staked, and not subtracting the ones with claimable amount 0.

Recommendation	Consider counting only the tokens where claimable amount is greater than 0.
Resolution	RESOLVED

Issue #53	Adjusting of reward rate affects users rewards retroactively
Severity	LOW SEVERITY
Description	If the reward rate is adjusted with either `setBaseNFTCost` or `setDenominator`, users will receive the new rewards rate over all unclaimed rewards. This could be especially bad if the reward rate is significantly increased.
Recommendation	Consider using Masterchef logic or not adjusting the reward rate in large magnitudes.
Resolution	ACKNOWLEDGED

Issue #54	`_initialBaseNFTCost` is private
Severity	LOW SEVERITY
Description	Important variables that third-parties might want to inspect should be marked as public so that these third-parties can easily inspect them through the explorer, web3 and derivative contracts.
Recommendation	Consider marking the variables as public.
Resolution	RESOLVED

Issue #55	`claimAll()` will revert if `_rewardsDelay` time has not passed
Severity	LOW SEVERITY
Description	Within the `claimAll` function, a loop is executed which checks if all staked NFTs have passed the minimum staking time to claim the rewards. `require( tokenStakes[tokenStakesByAddress[_msgSender()][i].tokenId].lastClaimTime < block.timestamp + _rewardsDelay, "No can do, pal." )` The whole function will fail if one staked NFT has not passed the time. Additionally, it is not necessary to check that during the loop since it is already checked in `getClaimableByTokenId` and will simply return 0.
Recommendation	Consider removing the redundant check.
Resolution	RESOLVED

Issue #56	`claimRewardsBySingleTokenId` does not charge a `claimFee`
Severity	INFORMATIONAL
Description	Since claimRewardsBySingleTokenId is called by unstakeAndAutoClaimForOldOwner, which is called by the NFT contract, it does not take any claimFee. Depending on the contract flow, this might be abused by users to claim their rewards without paying a fee.
Recommendation	Consider whether this is desired. If not, consider charging a fee as well or forgoing rewards on unstake.
Resolution	RESOLVED The client has indicated this is desired.

Issue #57	Balance check for `rewardToken` excludes equal case
Severity	INFORMATIONAL
Description	Within the functions `claimAll` and `claimRewardsBySingleTokenId` it checks if the contract balance is enough to pay out the rewards. However, it does not include the case balance is equal to the amount which should be paid out: `require( IERC20(_rewardToken).balanceOf(address(this)) > amountClaimable, "Insufficient balance." );`
Recommendation	Consider changing > to >= .
Resolution	RESOLVED

Issue #58	Several functions can be made external
Severity	INFORMATIONAL
Description	Functions that are not used within the contract but only externally can be marked as such with the external keyword. Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases. - `claimAll` - `getTotalClaimedByAddress` - `getTokenIdsStakedByAddress` - `getStakerInformationByAddress`
Recommendation	Consider marking the functions as external.
Resolution	RESOLVED

Issue #59	Unused functionality
Severity	INFORMATIONAL
Description	Functionality which is defined in a contract but remains unused could confuse third-party auditors. They also increase the contract length unnecessarily. Throughout the contract, we have identified multiple functionalities that are unused or redundant: 1) Line 110 `function deposit(uint256 amount) external payable nonReentrant` The `deposit` function does not need to be declared as `payable` because there is no need for governance to deposit gas tokens. Gas tokens are only used by the user to pay the `claimFee`. Additionally, this function should emit a `Deposit` event. 2) `requireTokenOwner(uint256 tokenId)` modifier is unused. 3) `StakerCompoundedReward` event is not used. 4) `paused` seems to be unused as there is no functionality that actually stops working if paused, and functionality only starts working once paused.
Recommendation	Consider removing the functionalities to keep the contract short and simple.
Resolution	RESOLVED

Issue #60	Typographical errors
Severity	INFORMATIONAL

Description	The contract contains a number of typographic mistakes which we have consolidated in a single issue in an effort to keep the report size reasonable.
	
	Line 78
	`event Launch(bool launced);`
	
	launced should be *launched*.
	
	Line 83
	`address rewardToken,`
	
	`rewardToken` can be cast to `IERC20`.
	
	Line 588
	`function setFeeReciever(address payable feeReceiver)`
	
	`setFeeReciever` should be named `setFeeReceiver`.
	
	Several sections of the documentation are also outdated, especially towards the end of the contract. We expect the developer copy-pasted certain parts of documentation without adjusting them.
	
	Finally, both `msg.sender` and `_msgSender()` are used throughout the contract. All though not an explicit issue, it is inconsistent and the client should stick to one version.

Recommendation	Consider fixing the typographical errors.
Resolution	RESOLVED

Issue #61	Lack of validation
Severity	INFORMATIONAL

Description	The contract contains a number of validation issues which we've enumerated below in a single issue in an effort to keep the report size reasonable.
	
	Line 491
	`function setRewardToken(address rewardToken) external requireAdmin requirePaused nonReentrant {`
	`    _rewardToken = rewardToken;`
	`}`
	
	`_rewardToken` should have a non-zero check.
	
	Line 510
	`function setNFTContract(address nftContract) external requireAdmin requirePaused nonReentrant {`
	`    _nftContract = nftContract;`
	`}`
	
	`_nftContract` should have a non-zero check.
	
	Line 549
	`function setDenominator(uint256 denominator) external requireAdmin requirePaused nonReentrant {`
	`    _denominator = denominator;`
	`}`
	
	`_denominator` should be within a reasonable range.
	
	Line 588
	`function setRewardsDelay(uint256 rewardsDelay) external requireAdmin nonReentrant {`
	`    rewardsDelay = _rewardsDelay * 1 minutes;`
	`}`
	
	`_feeReceiver` should have a non-zero check.

Line 607

```
function setRewardsDelay(uint256 rewardsDelay) external
requireAdmin nonReentrant {
    rewardsDelay = _rewardsDelay * 1 minutes;
}
```

`_rewardsDelay` should be within a reasonable range.

Line 660

```
function setPowerLevelContractAddress(address
powerLevelManagerContract) external  requireAdmin
nonReentrant  {
...
    _powerLevelManagerContract = powerLevelManagerContract;
...
}
```

`_powerLevelManagerContract` should have a non-zero check.

Recommendation Consider validating all above functions and variables.

We recommend combining the `setBaseNFTCost` and `setDenominator` functions into a single function as currently, the requirement on line 535 is not applied in `setDenominator`. In general, it is cleaner to put the setter for the nominator and denominator of a fraction in the same function (or making the denominator constant).

Resolution RESOLVED

Issue #62	Lack of events for various methods
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications. We identified the following functions that need to emit an event: - `deposit` - `withdrawRewardsToken` - `withdrawNative` - `setRewardToken` - `setNFTContract` - `setBaseNFTCost` - `setDenominator` - `setNativeClaimFee` - `setFeeReciever` - `setRewardsDelay` - `setPowerLevelContractAddress` - `setPaused`
Recommendation	Consider marking the functions mentioned above as external.
Resolution	RESOLVED

Issue #63	Gas optimizations
Severity	INFORMATIONAL
Description	Throughout the contract, we have identified several sections that can be optimized for gas usage: - `_secondsInDay` can be `constant`. - `_rewardsDelay` and `_denominator` initial values are immediately overridden in the constructor. - `lengths` can be cached into `memory` in for loops. - The assignment of `lastClaimTime` could be removed in `claimRewardsBySingleTokenId` as this is deleted afterwards. - Certain functions like `getTokenIdsStakedByAddress` could run out of gas and get rate-limited by the RPC that handles them because the response becomes too long.
Recommendation	Consider implementing the above recommendations.
Resolution	PARTIALLY RESOLVED Some of these optimizations have been resolved.

2.9 ShareStakeDividendsModel

`ShareStakeDividendsModel` is a variant on the dividend distributor. It represents another mechanism for users to receive rewards for holding the Alpha Shares NFTs. A part of the Alpha Shares purchase rewards are sent here.

A fee in the gas token (AVAX) needs to be paid to claim dividends.

When an Alpha Share NFT is transferred, it is automatically staked into the Dividends Model.

2.9.1 Privileges

The following functions can be called by the owner and other privileged roles of the contract:

- `emergencyWithdrawRewardToken [ admins ]`
- `emergencyWithdrawNative [ admins ]`
- `addShares [ nft contract ]`
- `removeShares [ admins ]`
- `setPaused [ admins ]`
- `setFeeReciever [ admins ]`
- `setNativeClaimFee [ admins ]`
- `grantRole`
- `revokeRole`
- `renounceRole`

2.9.2 Issues & Recommendations

Issue #64	`DEFAULT_ADMIN_ROLE` is not assigned
Severity	MEDIUM SEVERITY
Description	`ShareStakeDividendsModel` contract uses the `AccessControl` library from OpenZeppelin, a known pattern that gives Role Based Access capabilities to a contract. The `AccessControl` contract has defined a `DEFAULT_ADMIN_ROLE` role which has the power to grant/revoke roles to/from different recipients. By default, this role is not assigned to any address, and usually it should be assigned in the constructor to `msg.sender` or to a particular address. The downside of not setting a `DEFAULT_ADMIN_ROLE` is that there is no way for the governance to revoke any roles for privileged accounts, except with the accounts' consent.
Recommendation	Consider assigning the `DEFAULT_ADMIN_ROLE` to the `msg.sender` in the constructor or to a certain address received as parameter or move to Ownable implementation as currently the contract has just one role defined.
Resolution	RESOLVED The client has moved to the AuthContract.

Issue #65	Missing safeguards on `setFeeReciever`
Severity	LOW SEVERITY
Description	The `setFeeReciever` function is used to initialize who should receive the fee when someone is claiming the dividends. Currently there is no check if the new fee receiver address is `address(0)` which can cause the protocol to lose the fees.
Recommendation	Consider adding an `address(0)` check.
Resolution	RESOLVED

Issue #66	Unused and unnecessary functionality
Severity	INFORMATIONAL

Description	Throughout the contract, we have identified multiple functionalities that are unused or redundant:
	
	1) `minPeriod` and `minDistribution` variables are unused.
	
	2)
	Line 169-171
	`if (!stakerHasDeposit[tokenOwner]) return;`
	
	`require(stakerHasDeposit[tokenOwner], "No Shares");`
	
	The `if` statement and the `require` check against the same variable `stakerHasDeposit[tokenOwner]` . Consider removing one of the checks.
	
	3)
	Line 77
	`function deposit(uint256 amount) external payable {`
	
	`payable` modifier is unnecessary.
	
	4) The `receive()` function is unnecessary.

Recommendation	Consider implementing the recommendation fixes.
Resolution	PARTIALLY RESOLVED The `receive` function is still present.

Issue #67	Lack of events for various functions
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications. The following functions lack events: - `setRewardToken` - `setFeeReciever` - `setNativeClaimFee` - `claimDividend` - `emergencyWithdrawNative` - `emergencyWithdrawRewardToken` - `setPaused`
Recommendation	Add events for the above functions.
Resolution	RESOLVED

Issue #68	Typographical errors
Severity	INFORMATIONAL

Description	The contract contains a number of typographical errors which we have consolidated below in a single issue in an effort to keep the report size reasonable.
	
	Line 24
	`address public REWARD_TOKEN`
	
	`REWARD_TOKEN` can be made of type `IERC20`.
	
	Line 57
	`constructor(address rewardToken)`
	
	Line 284
	`function setRewardToken(address rewardToken)`
	
	`rewardToken` can be made of type `IERC20`.
	
	Line 301
	`function setFeeReciever(address payable feeReceiver)`
	
	`setFeeReciever` should be `setFeeReceiver`.
	
	The contract mixes usage of both `msg.sender` and `_msgSender` which is not an issue but is clearly inconsistent and forgoes the value of using the `_msgSender` function.
	
	Line 295
	`*  @param feeReceiver - big int of the native claim fee`
	
	This description is outdated.

Recommendation	Consider fixing the typographical errors.
Resolution	RESOLVED

2.10 AlphaSingleStake

`AlphaSingleStake` is a standard staking contract heavily inspired by the well-known Synthetix StakingRewards (https://github.com/Synthetixio/synthetix/blob/develop/contracts/StakingRewards.sol) staking contract. It allows users to deposit some `stakingToken` and earn `rewardsToken` over time.

Compared to Synthetix, the `deposit` and `withdraw` functions were changed and they use a share calculation during the deposits and withdrawals which acts similar to the user balance increments and decrements. Users can deposit the Alpha token and receive a reward token based on their share.

Governance can supply the reward token to the contract and determine the reward distribution period. The `rewardsPerBlock` variable is determined by the amount of the reward token which is supplied by governance divided through the `rewardDurationInBlocks`.

Governance power can extend the reward period freely without any restrictions.

2.10.1 Privileges

The following functions can be called by the owner and other privileged roles of the contract:

- `updateRewards [ admins ]`
- `emergencyWithdrawRewardToken`
- `authorize`
- `unauthorize`
- `transferOwnership`
- `renounceOwnership`

2.10.2 Issues & Recommendations

Issue #69	**Users will potentially lose their pending rewards when `updateRewards` is called and the final rewards will be locked in the contract indefinitely**
Severity	HIGH SEVERITY
Description	Currently, during `updateRewards`, an authorized party sets the `lastUpdateBlock` to `block.number`. This will result in a loss of rewards for the unclaimed period if the `lastUpdateBlock` is still before the end block. These rewards will also not be incremented into the next reward cycle and are permanently stuck in the contract. This issue is marked as High because the `updateRewards` function is called inside the non-privileged `deposit` function in the `TransactionFeeDistributor`. The non-privileged trait of this function will leave an open door for an attacker to call the `updateRewards` and cause a loss of rewards.
Recommendation	Consider updating the rewards similar to how Synthetix does this by calling `_updateReward(address(0))` and adding a zero-exception to `_updateReward` to ban the zero address from rewards.
Resolution	RESOLVED

Issue #70	Lack of validation during `updateRewards`
Severity	MEDIUM SEVERITY
Description	Currently, there is no validation when setting the parameters for the `updateRewards` function. If the reward duration was set mistakenly way too short, all future rewards might be distributed instantly. `currentRewardPerBlock = (reward + ((periodEndBlock - block.number) * currentRewardPerBlock)) / rewardDurationInBlocks;`
Recommendation	Consider setting reasonable safeguards for this function.
Resolution	RESOLVED

Issue #71	`deposit` could be vulnerable to reentrancy and does not support tokens with a fee on transfer
Severity	LOW SEVERITY
Description	The `deposit` function is not written in checks-effects-interactions as the inward transfer is executed too early. This is not best practice as it potentially opens up the contract to reentrancy vulnerabilities. This issue has been rated as low as the contract will likely not support those tokens and the functions are safeguarded with nonReentrant as well.
Recommendation	Consider moving the inwards transfer downwards and consider whether tokens with a fee on transfer need to be supported. If so, Paladin will guide the client through how to add the logic to support them.
Resolution	RESOLVED The client has indicated they do not plan to support tokens with a fee on transfer and `deposit` has been safeguarded against reentrancy.

Issue #72	Typographical errors
Severity	INFORMATIONAL

Description	Variables that are of type `IERC20` can be marked as such, if the contract imports the specific library.
	
	Line 79
	`rewardToken = IERC20(_rewardToken)`
	
	Line 80
	`alphaToken = IERC20(_alphaToken)`
	
	These variables can be declared directly as `IERC20` in the constructor.
	
	Line 305
	`// Retrieve total amount staked and calculated current amount (in LOOKS)`
	
	Consider removing *`in LOOKS`*.
	
	Lines 313-324
	`if (claimRewardToken) { alphaToken.safeTransfer // Fetch pending rewards pendingRewards = userInfo[msg.sender].rewards; if (pendingRewards > 0) { userInfo[msg.sender].rewards = 0; rewardToken.safeTransfer(msg.sender, pendingRewards); } }`
	
	The `withdrawAll` check does not check that the user has shares to withdraw. This is inconsistent with `withdraw`. The client should consider adding a `require` statement to be consistent.

	Finally, the whole share accounting logic is redundant because the share ratio will always remain 1:1. It can therefore be removed (and to be honest a simple StakingRewards copy would have sufficed for this contract).
Recommendation	Consider fixing the above errors.
Resolution	RESOLVED This issue has been resolved, however, we have to indicate that the way the client moved away from share accounting logic is not very pretty. A bunch of logic can still be simplified.

2.11 Auth and AuthContract

Auth

`Auth` is a contract module that provides an access control mechanism. After deployment, the deployer becomes the owner of the contract. They are also given authority privileges. The owner can later pass on their ownership using the `transferOwnership` function.

The codebase uses the Auth module through inheritance and functions can be made exclusively callable by the owner by applying the `onlyOwner` modifier to these functions.

Another main functionality of this contract is that the owner can grant authorization to desired addresses with the authorize function. Specific functions can be made accessible only by the authorized addresses when used with the isAuthorized modifier. The owner can revoke the authorizations with the unauthorize function.

AuthContract

`AuthContract` is another contract module that provides the same access control mechanism as the `Auth` contract. Unlike `Auth` which has its own `owner` variable, `onlyOwner` modifier and `transferOwnership` function, the `AuthContract` inherits the Ownable contract of OpenZeppelin. Other than that, these two contracts basically provide the same functionality. They both allow implementations within the codebase to limit access to certain parts of the contract to only the owner or authorized addresses.

Unless there is a particular reason that the client needs both the `Auth` and `AuthContract` contracts separately, we recommend that the client removes either one.

2.11.1 Privileges

The following functions can be called by the owner and other privileged roles of Auth:

- `authorize`
- `unauthorize`
- `transferOwnership`

The following functions can be called by the owner and other privileged roles of AuthContract:

- `authorize`
- `unauthorize`
- `transferOwnership`
- `renounceOwnership`

2.11.2 Issues & Recommendations

Issue #73	The previous owner still keeps the authorization status after `transferOwnership`
Severity	LOW SEVERITY
Description	Within the `transferOwnership` function, the old owner is changed to a new owner, however, the authorizations role is not being removed.
Recommendation	Consider if this is intentional or not and if not, revoke the authorization role from the old owner.
Resolution	RESOLVED This has been fixed within the AuthContract. The client has indicated Auth will not be used.

Issue #74	`owner` is internal within the Auth contract
Severity	LOW SEVERITY
Description	Important variables that third-parties might want to inspect should be marked as public so that these third-parties can easily inspect them through the explorer, web3 and derivative contracts.
Recommendation	Consider marking the variable as public.
Resolution	RESOLVED The client has indicated Auth will not be used.

Issue #75	**`authorize`, `unauthorize` and `transferOwnership` can be made external**
Severity	INFORMATIONAL
Description	Functions that are not used within the contract but only externally can be marked as such with the external keyword. Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases.
Recommendation	Consider marking the functions as external. Note that `transferOwnership` cannot be made `external` within `AuthContract`.
Resolution	RESOLVED `transferOwnership` has not been made external because it is an Ownable function.

Issue #76	**Unnecessary modifier: payable**
Severity	INFORMATIONAL
Description	Line 62 `function transferOwnership(address payable adr) public onlyOwner {` The `payable` modifier is not necessary as the contract never sends tokens to this address. It could confuse third-party auditors and increase the contract length. Note that this issue does not apply to the `AuthContract`.
Recommendation	Consider removing the modifier mentioned above to keep the contract short and straightforward.
Resolution	RESOLVED The client has indicated Auth will not be used.

Issue #77	There is no easy way for users to iterate over the list of authorized wallets
Severity	INFORMATIONAL
Description	The contract allows the owner to grant "authorization" to multiple wallets. The user can check if any specific wallet has such authorization but there is no easy way to get the list of authorized wallets. If the user wants to confirm that all authorized wallets are for example multisigs, this becomes very difficult to validate.
Recommendation	Consider using EnumerableSet instead of a mapping and adding the appropriate `public view` functions to get the length and elements at specific indices.
Resolution	RESOLVED The client moved to an `EnumerableSet`.

Issue #78	Typographical errors
Severity	INFORMATIONAL

Description	The contract contains a number of typographical errors which we have consolidated below in a single issue in an effort to keep the report size reasonable.
	
	`Auth::39`
	`* Remove address' authorization. Owner only`
	
	`Auth::53`
	`* Return address' authorization status`
	
	`AuthContract::14`
	`* Return address' authorization status`
	
	`AuthContract::28`
	`* Remove address' authorization. Owner only`
	
	The way to indicate possession by an address is `address's`.

Recommendation	Consider fixing the typographical errors.
Resolution	RESOLVED

Issue #79	Lack of indexing for event parameters
Severity	INFORMATIONAL
Description	Essential identifying parameters within events should be marked as indexed. This allows for off-chain components to filter events only including these values.
Recommendation	Add indices to the key variables within the events you might want to filter on.
Resolution	RESOLVED

Issue #80	Lack of events for `authorize` and `unauthorize`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions. Remember to add the `OwnershipTransferred` event to the `constructor` and the `AuthorisationSet` event to `authorize`, `transferOwnership` and the `constructor`.
Resolution	RESOLVED

2.12 ArtistRoyaltyDistributor, OwnerRoyaltyDistributor and TreasuryRoyaltyDistributor

`ArtistRoyaltyDistributor`, `OwnerRoyaltyDistributor` and `TreasuryRoyaltyDistributor` are basically identical contracts as they only differ by their names. The three contracts inherit the `PaymentSplitter` contract of OpenZeppelin to lock away a token and distribute the token to various recipients according to predefined weights.

The contract stores the stakeholder addresses and their share during the deployment. The stakeholder information is not editable afterward so it is important to make sure the provided information is correct.

Once the contract receives a token, the contract locks up the token until stakeholders call the `release` function. Each stakeholder has to claim the pending payment individually otherwise, the tokens will remain unclaimed in the contract.

2.12.1 Issues & Recommendations

No issues found.

2.13 TransactionFeeDistributor

`TransactionFeeDistributor` enables distribution of the `WRAPPED_NATIVE` token to 5 different addresses, namely `NFT_STAKING_POOL`, `PLATFORM_PAYMENT_SPLITTER`, `ARTIST_ROYALTY_SPLITTER`, `OWNER_PAYMENT_SPLITTER`, and `ALPHA_STAKING_POOL`. These address variables are initially empty and the owner has to manually set them using the `setAddresses` function, otherwise most of the functions in the contract will likely fail.

The deposit function transfers the `WRAPPED_NATIVE` token to the contract from the `msg.sender`. The transferred tokens are then distributed to the five addresses mentioned above. The distribution proportion is initially set to specific numbers during deployment but the owner can later change it with the `setFeeSplits` function.

Unlike other distribution addresses that receive the `WRAPPED_NATIVE` token, `NFT_STAKING_POOL` receives `NFT_STAKING_REWARD_TOKEN`. To be more specific, the `distributeNFTStakingShare` function swaps the `WAVAX` into `NFT_STAKING_REWARD_TOKEN`, and then the `NFT_STAKING_POOL` receives the resulting amount.

The contract has significant governance privilege as the owner has full freedom to change all recipient addresses, the fee distribution percentages and even the tokens.

2.13.1 Privileges

The following functions can be called by the owner and other privileged roles of the contract:

- `setAddresses`
- `emergencyWithdrawNative`
- `emergencyWithdrawNFTRewardToken`
- `emergencyWithdrawWrappedNative`
- `setNFTStakingRewardToken`
- `setRouterAddress`
- `setAlphaStakingPoolAddress`
- `setShareStakeDividendModelAddress`
- `setPlatformPaymentSplitterAddress`
- `setArtistPaymentSplitterAddress`
- `setOwnerPaymentSplitterAddress`
- `setPaused`
- `setMinPurchaseAmount`
- `setFeeSplits`
- `grantRole`
- `revokeRole`
- `renounceRole`

2.13.2 Issues & Recommendations

Issue #81	`DEFAULT_ADMIN_ROLE` is not assigned
Severity	HIGH SEVERITY
Description	`TransactionFeeDistributor` contract uses the `AccessControl` library from OpenZeppelin, a known pattern that gives Role Based Access capabilities to a contract. The `AccessControl` contract has defined a `DEFAULT_ADMIN_ROLE` role which has the power to grant/revoke roles to/from different recipients. By default, this role is not assigned to any address, and usually it should be assigned in the constructor to msg.sender or to a particular address. The `TransactionFeeDistributor` is missing the assignment of the role to any address which will result in a scenario where the only role defined by the contract, `ADMIN_ROLE`, is locked to the deployer.
Recommendation	Consider assigning the `DEFAULT_ADMIN_ROLE` to the `msg.sender` in the constructor or to a certain address received as parameter or consider moving to Ownable if only `ADMIN_ROLE` is used within the contract.
Resolution	RESOLVED The client has moved to `AuthContract` from `AccessControl`.

Issue #82	**`deposit` could be used to execute a sandwich attack**
Severity	LOW SEVERITY
Description	If the `MIN_DEX_PURCHASE_AMOUNT` is ever set to a very high number, a malicious attacker could spy the WAVAX amount in the contract and can execute a deposit with a tiny amount which finally triggers `distributeNFTStakingShare(balance)`. This results in a swap from WAVAX to `NFT_STAKING_REWARD_TOKEN` which increases the value of `NFT_STAKING_REWARD_TOKEN`. An attacker can then dump his `NFT_STAKING_REWARD_TOKEN` for a profit. More often these tokens were purchased right before the attack in what is called a "sandwich attack". On the other hand, if `MIN_DEX_PURCHASE_AMOUNT` is ever too low, the whole function would revert due to the require statement in the deposit function in `AlphaSingleStakingDividendsModel`: `require( amount >= PRECISION_FACTOR, "Deposit: Amount must be >= 1 ALPHA" );`
Recommendation	Consider adding reasonable safeguards to `setMinPurchaseAmount`.
Resolution	RESOLVED The minimum purchase amount is now kept reasonably low.

Issue #83	Lack of validation
Severity	INFORMATIONAL
Description	The contract contains functions with parameters that are not properly validated. Having unvalidated parameters could allow the governance or users to provide variable values which are unexpected and incorrect. This could cause side-effects or worse exploits in other parts of the codebase. Consider validating the following function parameters: The client could consider to validate that `NFT_STAKING_REWARD_TOKEN` matches `REWARD_TOKEN` of `NFT_STAKING_POOL` or even automatically set it to that token. Within the constructor, all shares should be validated to sum to `_denominator`. Within the `deposit` function, only the artist's payment is checked to be greater than zero. We recommend adding the same `if` statement for all transfers. The `distributeNFTStakingShare` does not support that `NFT_STAKING_REWARD_TOKEN` is equal to WAVAX. This would cause the path to go from WAVAX into WAVAX and revert. This should either be validated or the swap should not occur in this case. The `setAddresses` function needs to validate that the addresses are not set to the zero address (`address(0)`), otherwise multiple functionality will misbehave in this scenario as most ERC-20 tokens do not allow transfers to the zero address.
Recommendation	Consider validating the function parameters mentioned above.
Resolution	RESOLVED Some of these variables will however be checked after deployment.

Issue #84	`rewardDurationInBlocks` can be made constant
Severity	INFORMATIONAL
Description	Variables that are never modified can be indicated as such with the constant keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the variable explicitly constant.
Resolution	RESOLVED

Issue #85	Fee default variables are immediately overwritten
Severity	INFORMATIONAL

Description	Lines 42-46
	`uint256 public _nftStakingPoolShare = 277;`
	`uint256 public _alphaStakingPoolShare = 112;`
	`uint256 public _platformShare = 167;`
	`uint256 public _ownerShare = 444;`
	`uint256 public _artistShare = 0;`
	
	The contract specifies various initial parameter values for the pool shares. However, these are immediately overwritten within the constructor and therefore bear no meaning.

Recommendation	Consider either removing the constructor initialization of these variables, or the parameter values.
Resolution	RESOLVED

Issue #86	Unused variable and pausable logic
Severity	INFORMATIONAL
Description	Line 50 `IERC20 public WAVAX;` Variables defined in a contract but not used within the contract could confuse third-party auditors. They also increase the contract length unnecessarily. The pausable logic is also basically unused as deposits can still occur when the contract is paused.
Recommendation	Consider removing the unused variable and logic to keep the contract short and simple.
Resolution	RESOLVED

Issue #87	Typographical errors
Severity	INFORMATIONAL

Description	The contract contains a number of typographical errors which we have consolidated below in a single issue in an effort to keep the report size reasonable.
	
	Line 66
	`address dexRouter`
	
	Line 226
	`address _router`
	
	These addresses can immediately be cast as `IDEXRouter` to avoid casting it later on.
	
	Line 322
	`* @dev Mofifiers`
	
	This should say *modifiers*.

Recommendation	Consider fixing the typographical errors.
Resolution	RESOLVED

Issue #88	Unnecessary modifier: payable
Severity	INFORMATIONAL
Description	The `deposit` function specifies a `payable` modifier but no raw `AVAX` is supposed to be sent to this function. Many addresses are also unnecessarily marked as `payable`.
Recommendation	Consider whether this is desired for future interface compatibility. If not, consider removing the `payable` modifier.
Resolution	RESOLVED

Issue #89	Lack of events for various functions
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications. The following functions lack events: - `deposit` - `setAddresses` - `emergencyWithdrawNative` - `emergencyWithdrawNFTRewardToken` - `emergencyWithdrawWrappedNative` - `setNFTStakingRewardToken` - `setRouterAddress` - `setAlphaStakingPoolAddress` - `setShareStakeDividendModelAddress` - `setPlatformPaymentSplitterAddress` - `setArtistPaymentSplitterAddress` - `setOwnerPaymentSplitterAddress` - `setPause` - `setMinPurchaseAmount` - `setFeeSplits`
Recommendation	Add events for the above functions.
Resolution	RESOLVED Most of the functions now have events.

Issue #90	Gas optimizations
Severity	INFORMATIONAL

Description	The contract contains multiple sections of code that could be further optimized for gas efficiency. We have consolidated these in a single issue in an effort to keep the report brief and readable.
	
	Line 194
	`IERC20(router.WAVAX()).approve(address(router), amount);`
	
	Line 197
	`path[0] = router.WAVAX();`
	
	The client can reuse the `WRAPPED_NATIVE` address here to reduce the gas cost of the `distributeNFTStakingShare` function. In general, many variables, including `WRAPPED_NATIVE` can be cached into memory to reduce gas costs further but the client might want to strike a balance between code length and gas optimization so we understand if they refrain from doing this.

Recommendation	Consider implementing the gas optimizations mentioned above.
Resolution	RESOLVED

2.14 Marketplace V2

The NFT Marketplace is a simple marketplace where users can sell and auction their NFTs. Users have the ability to create an English auction, where the highest bidder will win, the seller can either accept or deny the bid within the `gracePeriod` which is currently one day but changeable without any limits. Users can bid on all open auctions, with a minimum difference to the last bid of `minIncrementPerBid` which is unique for each auction. Additionally, sellers can list their NFT for a flat price to the open world, as well as only for one specific wallet.

If a bid to an auction was not accepted by the seller within the `sale.endTime + gracePeriod`, the sale can be canceled by the seller or by the highest bidder, in which case the last bidder will get refunded and the sale will be cancelled.

A special function was implemented with the genesis case: during the genesis timeline, which can be set by the admin in `scheduleGenesis`, every bidder needs to deposit a collateral of the `genesisCollateralToken` to the contract to be able to bid. The amount which needs to be deposited can be set by the admin via `setGenesisCollateralRequired` without any limits. During the genesis time, users can only auction `alphaNFTs` which can be set by the admin in `setAlphaNFTs` and requires `IAlphaNFT(address(_nft)).getNFTTypeByTokenId(_id) > 0`. During that genesis time, only NFTs from type 1 or higher can be auctioned.

If a bidder was outbidded, the old bidder will get back his collateral (in case of the genesis period) and his bid refunded. The user can then withdraw his bid amount via `withdrawEarnings()`, and his collateral will be refunded automatically via `refundGenesis()`.

If a bid was accepted by the seller or a direct buy was made, the sale will be cleared, processed and fees are calculated in `collectFees()`. The fee is 2.5% for both `alphaNFTs` and standard NFTs. Both fees can be changed with `setFee()`. The fee receiver for the AlphaNFTs is the `_genesisFeeDistributor` and for the

standard NFTs the `_baseFeeDistributor`. Both can be changed during `setNftTypeFeeDistributor`. If the NFT supports the standard interface, the `royaltyInfo` is being fetched and `royaltyFee` and `royaltyFeeReceiver` are cached in the accounting struct. After that is done, the amount after fees is added to the sellers pending payments and the fees will be distributed, the genesis collateral from the buyer will be refunded, if he has no other open bids and the NFT will be transferred from the seller to the buyer.

The contract uses the `AccessControl` library to determine one or more "MANAGER". The admin and the manager both are able to ban NFT collections via `toggleNFTCollection` — this was implemented as a safeguard in case of malicious NFT projects. Once a NFT collection gets banned, there is no possibility that this collection can be traded on the marketplace without unbanning it. It is worth to note that every NFT collection is marked `false` as default (banned), which means that they need to be approved first before they can be traded.

Furthermore, the NFT Marketplace implemented an `offer` function: via `addOffer()`, a user can suggest an offer for any approved NFT on the marketplace with the restriction that this can only happen if an NFT does not have an active auction. The owner can accept the offer via `acceptOffer` which transfers the NFT from the seller to the new owner and the funds after fees to the seller. Users are also able to add offers to expired auctions.

Two additional governance privileges are the `sweepNFT` and `sweepERC20` functions, which allows the admin to withdraw banned NFTs as well as `ERC20` tokens excluding the `paymentToken` and the `genesisCollateral`.

The owner has the privilege to pause the contract which will block the following functions:

- `bid`
- `acceptBid`
- `buy`
- `cancelSale`

- `startAuction`
- `startDirectSell`
- `startDirectSellTo`

2.14.1 Privileges

The following functions can be called by the owner of the contract:

- `setNftTypeDistributor`
- `setBaseFeeDistributor`
- `setFee`
- `setGracePeriod`
- `setAlphaNFTs`
- `setGenesisCollateralRequired`
- `scheduleGenesis`
- `pause`
- `toggleNFTCollection (also MANAGER)`
- `sweepNFT`
- `sweepERC20`

2.14.2 Issues & Recommendations

Issue #91	The contract does not support tokens with a fee on transfer
Severity	LOW SEVERITY
Description	Due to contract logic, it is necessary to always inspect which tokens to use as collateralToken and as paymentToken, in case of transfer-tax tokens, this will lead to issues.
Recommendation	Consider either adding logic to support tokens with a fee on transfer or avoid adding them.
Resolution	ACKNOWLEDGED The client will never add tokens with a fee on transfer as `collateralToken` or `paymentToken`.

Issue #92	`processSale` might revert if `feeRecipient` is an EOA or an incompatible contract
Severity	INFORMATIONAL
Description	At line 636 in `processSale`, the `baseFee` gets distributed to the `feeRecipient` via `deposit`. However, if the `feeRecipient` is an EOA or a contract without a deposit implementation, the function will revert.
Recommendation	Consider being careful when adding the `feeRecipient`.
Resolution	ACKNOWLEDGED

Issue #93	Governance privileges: pause and `setGracePeriod`
Severity	INFORMATIONAL
Description	The pause function affects the state of the contract and can cause several functions to revert: - `bid` - `acceptBid` - `buy` - `cancelSale` - `startAuction` - `startDirectSell` The function `setGracePeriod` can set the `gracePeriod` to zero which will then affect `acceptBid` and `cancelSale`. If a seller does not monitor the `gracePeriod`, he might fail to accept a bid because `require(block.timestamp < sale.endTime + gracePeriod, "expired")` could already be expired. The `cancelSale` function could also be canceled earlier than expected: `if (block.timestamp > sale.endTime + gracePeriod && sale.highestBidder != address(0)).`
Recommendation	Consider only pausing the contract when its really necessary and always communicate this upfront with the community.
Resolution	ACKNOWLEDGED

Issue #94	Genesis auction can be longer than desired
Severity	INFORMATIONAL
Description	During the function startAuction it takes the `_duration` as argument, however, during isGenesisActive() it validates the `_duration` only with `MAX_GENESIS_DURATION`, which is 7 days. Due to this mechanism, if a new auction is generated, near the end of the genesis timeframe, this auction can exceed the usual genesis timeframe because it is calculated with block.timestamp + `_duration`.
Recommendation	Consider either adding a safeguard which only allows the auction to run until the end of the genesis period or accepting this case.
Resolution	ACKNOWLEDGED

Issue #95	`collectFees` could revert if `baseFee` and `royaltyFee` exceed 100%
Severity	INFORMATIONAL
Description	Within the `collectFees` function, it calculates the leftover amount as following: `accounting.amountLeftOver = _amount - accounting.baseFee - accounting.royaltyFee;` Since there is only a 100% cap for `baseFee`, the math could easily underflow when `royaltyFee + baseFee` is over 100%. However, there is no way to validate `royaltyFee` before.
Recommendation	Consider setting the `baseFee` within a reasonable range to prevent any math underflows.
Resolution	ACKNOWLEDGED

Issue #96	Seller can always revoke his approval
Severity	INFORMATIONAL
Description	Due to the mechanism of the protocol, the NFT will only be transferred within the `processSale` function, however, the seller can prevent this transfer when he revokes his approval and this results in a function revert.
Recommendation	Consider either changing the contract logic or simply acknowledge this possible case.
Resolution	ACKNOWLEDGED


PALADIN
BLOCKCHAIN SECURITY