PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Seijin

31 October 2023

paladinsec.co

info@paladinsec.co

Table of Contents

Table of Contents 2

Disclaimer 4

1 Overview 5

1.1 Summary 5

1.2 Contracts Assessed 6

1.3 Findings Summary 7

1.3.1 Global Issues 8

1.3.2 ARBR 8

1.3.3 Administrable 8

1.3.4 ArborDeposit 8

1.3.5 ArborMarketplace 9

1.3.6 ArborSale 9

1.3.7 ArborSalesFactory 9

1.3.8 ArborStaking 10

1.3.9 HARBR 10

1.3.10 MultiSigWallet 10

1.3.11 Signatories 10

2 Findings 11

2.1 Global Issues 11

2.1.1 Issues & Recommendations 12

2.2 ARBR 15

Key Functionalities 15

Security and Supply Management 16

2.2.1 Issues & Recommendations 17

2.3 Administrable 18

Key Characteristics 18

2.3.1 Privileged Functions 19

2.3.2 Issues & Recommendations 19
2.4 ArborDeposit 20
2.4.1 Privileged Functions 20
2.4.2 Issues & Recommendations 21
2.5 ArborMarketplace 24
2.5.1 Issues & Recommendations 25
2.6 ArborSale 29
2.6.1 Privileged Functions 29
2.6.2 Issues & Recommendations 30
2.7 ArborSalesFactory 35
2.7.1 Privileged Functions 36
2.7.2 Issues & Recommendations 36
2.8 ArborStaking 37
2.8.1 Privileged Functions 38
2.8.2 Issues & Recommendations 39
2.9 HARBR 43
2.9.1 Privileged Functions 43
2.9.2 Issues & Recommendations 44
2.10 MultiSigWallet 45
2.10.1 Privileged Functions 45
2.10.2 Issues & Recommendations 46
2.11 Signatories 48
2.11.1 Privileged Functions 48
2.11.2 Issues & Recommendations 49

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 the right to re-use any and all knowledge and expertise gained during the audit process, including, but not limited to, vulnerabilities, bugs, or new attack vectors. Paladin is therefore allowed and expected to use this knowledge in subsequent audits and to inform any third party, who may or may not be our past or current clients, whose projects have similar vulnerabilities. Paladin is furthermore allowed to claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for Seijin on the Sei 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	Seijin
URL	https://seijin.app
Platform	Sei
Language	Solidity
Preliminary Contracts	https://github.com/NIMA-Enterprises/launchpad-contracts/tree/ad2bce81ecb74dcb43c2e762fd8d281b220774a3/contracts
Resolution 1	https://github.com/NIMA-Enterprises/launchpad-contracts/tree/a7fa6084e0dd5b8248b6b362db00232c7b18046b/contracts

1.2 Contracts Assessed

Name	Contract	Live Code Match
ARBR		UNDEPLOYED
Administrable		UNDEPLOYED
ArborDeposit		UNDEPLOYED
ArborMarketplace		UNDEPLOYED
ArborSale		UNDEPLOYED
ArborSalesFactory		UNDEPLOYED
ArborStaking		UNDEPLOYED
HARBR		UNDEPLOYED
MultiSignWallet		UNDEPLOYED
Signatories		UNDEPLOYED

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
Governance	4	1	-	3
High	1	1	-	-
Medium	3	2	-	1
Low	12	9	-	3
Informational	9	3	-	6
Total	**29**	**16**	**-**	**13**

Classification of Issues

Severity	Description
Governance	Issues under this category are where the governance or owners of the protocol have certain privileges that users need to be aware of, some of which can result in the loss of user funds if the governance's private keys are lost or if they turn malicious, for example.
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 with 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	LOW	Lack of `safeERC20` usage	✓ RESOLVED
02	LOW	Risk of storage collision in a future smart contract implementation update	✓ RESOLVED
03	INFO	Cross-chain replay attack	ACKNOWLEDGED
04	INFO	Contracts are not compatible with tokens that have a fee on transfer	ACKNOWLEDGED

1.3.2 ARBR

ID	Severity	Summary	Status
05	INFO	Lack of `_totalSupply` validation	ACKNOWLEDGED

1.3.3 Administrable

No issues found.

1.3.4 ArborDeposit

ID	Severity	Summary	Status
06	GOV	Governance privilege	ACKNOWLEDGED
07	LOW	Checks-effects-interaction is not adhered to	✓ RESOLVED
08	LOW	Offer sale proceeds might become unintentionally part of a presale	ACKNOWLEDGED

1.3.5 ArborMarketplace

ID	Severity	Summary	Status
09	GOV	Governance privilege	✓ RESOLVED
10	HIGH	Offer sale can be frontrun	✓ RESOLVED
11	LOW	Whitelisting check can be circumvented by changing the price	✓ RESOLVED
12	LOW	Whitelisting logic is flawed	✓ RESOLVED
13	LOW	Checks-effects-interaction pattern is not adhered to	✓ RESOLVED
14	INFO	Hardcoded dust amount may affect usability of tokens with high value	ACKNOWLEDGED
15	INFO	Typographical issues	✓ RESOLVED

1.3.6 ArborSale

ID	Severity	Summary	Status
16	GOV	Governance privilege	ACKNOWLEDGED
17	MEDIUM	Supply management is flawed	✓ RESOLVED
18	MEDIUM	The check that a user has already participated can be circumvented	ACKNOWLEDGED
19	LOW	`SaleToken` can be stablecoin	ACKNOWLEDGED
20	LOW	Checks-effects-interaction pattern is not adhered to	ACKNOWLEDGED

1.3.7 ArborSalesFactory

No issues found.

1.3.8 ArborStaking

ID	Severity	Summary	Status
21	LOW	Checks-effects-interaction is not adhered to	✓ RESOLVED
22	LOW	Lack of fee validation	✓ RESOLVED
23	INFO	Lack of explicit error message	ACKNOWLEDGED
24	INFO	Typographical issues	✓ RESOLVED

1.3.9 HARBR

ID	Severity	Summary	Status
25	GOV	Governance privileges	ACKNOWLEDGED

1.3.10 MultiSigWallet

ID	Severity	Summary	Status
26	MEDIUM	pop is not executed	✓ RESOLVED
27	INFO	`replaceOwner` lacks zero check	✓ RESOLVED
28	INFO	`getTransactionCount` and `getTransactionIds` might run out of gas	ACKNOWLEDGED

1.3.11 Signatories

ID	Severity	Summary	Status
29	LOW	Signatories can be the same address	✓ RESOLVED

2 Findings

2.1 Global Issues

The issues listed in this section apply to the protocol as a whole. Please read through them carefully and take care to apply the fixes across the relevant contracts.

2.1.1 Issues & Recommendations

Issue #01	Lack of `safeERC20` usage
Severity	LOW SEVERITY
Description	There are multiple sections within the contract where `safeERC20` is not used — this will not work for tokens that return `false` on transfer or do not have a return value.
Recommendation	Consider using `safeERC20` throughout the codebase.
Resolution	RESOLVED

Issue #02	Risk of storage collision in a future smart contract implementation update
Severity	LOW SEVERITY

Description	The codebase uses OpenZeppelin's library contracts as a dependency at version ^4.9.3 and more specifically `Initializable.sol` at version 4.9.0.
	
	When we looked up the Initializable contract, we noticed that there are two variables of types `uint8` and `boolean` respectively stored in the first slot (index 0) of the contract's storage:
	`uint8 private _initialized;`
	`bool private _initializing;`
	
	However, in a more recent version of OpenZeppelin's `Initializable.sol`, these two variables are moved to a special storage position at `INITIALIZABLE_STORAGE`, meaning that the 0 slot is now unused and any other variables from inherited contract will be shifted and therefore their values will be corrupted.
	
	This can happen in a future upgrade of the implementation of any of the smart contract proxies used within the Launchpad system if the latest OpenZeppelin library is used.

Recommendation	Consider upgrading to the latest OpenZeppelin version. Also, always execute thorough testing and go through a security review before mainnet deployments of future implementation contracts to verify new logic and dependencies do not break the current storage layout.
Resolution	RESOLVED

Issue #03	Cross-chain replay attack
Severity	INFORMATIONAL
Description	The architecture does not adhere to `EIP712` nor implement a `chainID` for the messages. This can result in cross-chain replay attacks. This issue is rated as informational since we are already aware that the team will only deploy their contracts on the Arbitrum network.
Recommendation	Consider following `EIP712`.
Resolution	ACKNOWLEDGED

Issue #04	Contracts are not compatible with tokens that have a fee on transfer
Severity	INFORMATIONAL
Description	All contracts other than the `ArborSale` contract do not support tokens with a fee on transfer. This will lead to accounting issues.
Recommendation	Consider not using these tokens or implementing accounting logic for it. However, we highly assume that such tokens will never be used.
Resolution	ACKNOWLEDGED Only the `ArborMarketplace` and `ArborSale` contracts support tokens with a fee on transfer **during the initial deposit**. In other words, subsequent transfers out of the contract will have the fee deducted, resulting in the user receiving less tokens.

2.2 ARBR

ARBR is a simple token that adheres to the ERC-20 token standard. It is the implementation contract behind a proxy, ensuring upgradeability and maintainability in the architecture.

Key Functionalities

- **Burnable Feature**: ARBR comes with the intrinsic capability to burn tokens, thus reducing the total circulating supply. This burning mechanism can be triggered by users directly via the `burn` or `burnFrom` functions, despite there being no direct incentive to do so. A systematic burning process is also embedded within instant conversions in the `ArborStaking` contract, methodically influencing the total supply dynamics.
- **Initial Supply Minting upon initialization**: The ARBR token contract mints its entire supply and assigns it to a designated `_safe` address, thus establishing its maximum supply threshold. Notably, the maximum supply is subject to subsequent modifications only through the aforementioned burning processes, ensuring a diminishing supply trajectory post-launch.
- **Staking Mechanism Inextricably linked with the ArborStaking contract**: ARBR tokens can be staked to potentially yield rewards, engendering a utility beyond mere transferability for holders within the ecosystem. The staking functionality is designed to foster user participation and investment within the platform, offering an avenue for passive earning. It is important to note that the whole reward logic is calculated and handled off-chain.
- **Convertibility to HARBR**: ARBR tokens can be converted into HARBR tokens within the ArborStaking contract, furnishing a degree of interoperability and flexibility within the ecosystem. This conversion mechanism augments utility and provides options for the strategic maneuvers of token holders.

Security and Supply Management

While ARBR ensures a consistent and immutable maximum supply at the point of deployment, it institutes a judiciously balanced dynamic supply model through its burn mechanisms, serving to manage token scarcity and value in a controlled manner. Users retain the autonomy to execute burns, though primary supply alterations are anticipated to transpire through the instant conversion processes within the ArborStaking contract.

2.2.1 Issues & Recommendations

Issue #05	Lack of `_totalSupply` validation
Severity	INFORMATIONAL
Description	During the contract initialization, an arbitrary supply amount can be minted to the `_safe` address. This can be abused to embezzle tokens.
Recommendation	Consider setting a reasonable upper value for `_totalSupply`.
Resolution	ACKNOWLEDGED

2.3 Administrable

`Administrable` serves as a foundational access control mechanism within the smart contract system, ensuring that particular functionalities are exclusively accessible by a designated administrator. It is meant to be used as an implementation contract for a proxy pattern, and is thus upgradeable.

Initialized with a specified administrator address, this contract facilitates both the establishment and subsequent management of privileged access throughout the contract's lifecycle.

Key Characteristics

- **Initial Setup**: Upon deployment, an address is assigned as the "admin," orchestrating an initial layer of access control.

- **Mutable Administrative Privileges**: The admin possesses the capability to alter the administrative address, transferring control as and when deemed necessary.

- **Access Control Mechanism**: A modifier, typically denoted as `onlyAdmin`, is provided to restrict specific function calls exclusively to the admin, safeguarding against unauthorized access and ensuring that sensitive functionalities are shielded from external interactions.

- **Usage Across Multiple Contracts**: The `onlyAdmin` modifier is utilized across various contracts within the system, namely:
  - `ArborDeposit.sol`
  - `ArborMarketplace.sol`
  - `ArborSale.sol`
  - `ArborStaking.sol`

- `Signatories.sol`

  In these contracts, the `onlyAdmin` modifier acts as a gatekeeper, permitting only the admin to call certain functions, thereby providing a secure means to regulate interactions and alterations to the contract's state, processes, or other critical functionalities.

2.3.1 Privileged Functions

- `setAdmin`

2.3.2 Issues & Recommendations

No issues found.

2.4 ArborDeposit

`ArborDeposit` is the initial entry point for users to participate in the presale. Users can deposit stablecoins and will then receive a specific amount of sale tokens depending on their allocation. To illustrate the whole presale architecture consider the following flow:

Initially, users deposit stablecoins into the `ArborDeposit` contract, indicating their investment intent. Subsequently, an admin-controlled mechanism transfers these funds to the `ArborDeposit` contract, simultaneously allocating corresponding sale tokens to investors based on their contribution and predefined allocation rules. Users can then opt to either withdraw these sale tokens or leverage the `ArborMarketplace` contract, which acts as a secondary market, enabling them to list and sell their tokens to other investors. This structured architecture thus provides a seamless, secure, and user-friendly platform, managing the entire lifecycle of presale events from initial investment to token allocation and potential secondary market trading, thereby enhancing the overall user experience and investment management in the presale ecosystem.

It is important to mention that any sale proceeds from the `ArborMarketplace` contract will automatically deposit stablecoins on behalf of the sale creator to the `ArborDeposit` contract.

2.4.1 Privileged Functions

- `setManager (admin)`
- `migrateFundsToSale (manager)`

2.4.2 Issues & Recommendations

Issue #06	Governance privilege
Severity	GOVERNANCE
Description	Within the `migrateFundsToSale` function, stablecoins are approved to an arbitrary sale contract which is provided as an input parameter. While a check validates that this contract is in fact deployed by the factory, it is important to remember that the implementation can be subject to change which can result in lost funds. Moreover, there is no guarantee that the `ArborSale` contract has sufficient funds to cover all balances.
Recommendation	Presales always come with risk — all users should be aware that funds can be permanently lost when participating in a presale. A solution would be to make the implementation immutable as well as setting a reasonable price validation and checking that the contract has in fact a balance with can cover the whole presale. However, we assume that this flexibility is desired, therefore we simply recommend using a multi-signature contract as admin and manager.
Resolution	ACKNOWLEDGED

Issue #07	Checks-effects-interaction is not adhered to
Severity	LOW SEVERITY

Description	Several sections within the contract violate the check-effects-interactions pattern. This makes the contract vulnerable to reentrancy attacks.
	
	L78
	`function deposit(uint256 amount) external {`
	`    _stableCoin.transferFrom(msg.sender, address(this), amount);`
	`    unchecked {`
	`        _stableCoinBalance[msg.sender] += amount;`
	`    }`
	`    emit Deposit(msg.sender, amount);`
	`}`
	
	L90
	`function depositFor(address account, uint256 amount) external {`
	`    if (msg.sender != _marketplace) revert UnauthorizedCaller();`
	`    _stableCoin.transferFrom(msg.sender, address(this), amount);`
	`    unchecked {`
	`        _stableCoinBalance[account] += amount;`
	`    }`
	`    emit Deposit(account, amount);`
	`}`
	
	In the above examples, the transfer is done before the state variable change.

Recommendation	Consider rewriting the code to adhere to checks-effects-interactions.
Resolution	RESOLVED

Issue #08	Offer sale proceeds might become unintentionally part of a presale
Severity	LOW SEVERITY
Description	Whenever an offer is sold, the received payout is allocated towards this contract. Users can then simply withdraw it. However, this will also result in a risk of these funds being accidentally allocated to a new presale.
Recommendation	If it is desired that this flow is kept as-is, we highly recommend the team to keep track of these offer sale proceeds and exclude them from any presale.
Resolution	ACKNOWLEDGED The team indicated that this will be handled automatically by the backend.

2.5 ArborMarketplace

`ArborMarketplace` serves as a decentralized marketplace where users can trade tokens that they have acquired from the `ArborSale` contract. Specifically, users have the ability to list their tokens as "offers" for sale, modify the prices of these offers, remove their offers, or buy tokens from existing offers.

When a user lists an offer, the corresponding balance of tokens in their `ArborSale` account becomes "occupied", essentially locking up these tokens to secure the offer. These tokens remain locked until either the offer is removed or purchased by another user.

If a user wishes to update the offer, they can do so by modifying the price. This allows for flexibility and dynamic pricing strategies without affecting the underlying tokens that are locked up.

When an offer is removed or purchased, the corresponding balance in the `ArborSale` account is "released", making the tokens available for the user to either withdraw or use for another offer.

Additionally, `ArborMarketplace` implements a fee on every successful purchase. This fee is deducted from the sale proceeds and serves as a revenue stream for the platform. The deducted fee is calculated as a percentage of the sale price, ensuring that sellers receive their proceeds minus the applicable fee.

In summary, `ArborMarketplace` provides a comprehensive and secure platform for the trading of tokens acquired through the `ArborSale` contract, complete with dynamic pricing and a fee mechanism to facilitate smooth and efficient transactions.

2.5.1 Issues & Recommendations

Issue #09	Governance privilege
Severity	GOVERNANCE
Description	The admin can change the `_arborDeposit` address at any time, which can result in a loss of stablecoin funds.
Recommendation	Consider marking the `_arborDeposit` address as immutable, however, this flexibility might be desired.
Resolution	RESOLVED

Issue #10	Offer sale can be frontrun
Severity	HIGH SEVERITY
Description	`updateOfferPrice` allows the price to be changed — this can be done whenever a purchase transaction within the mempool is detected and a buyer's whole approved funds can be drained as a result.
Recommendation	Consider only allowing a reasonable change, limiting it with a time-factor.
Resolution	RESOLVED A price parameter can now be provided with the purchase, ensuring that the price in fact matches the desired buy price.

Issue #11	Whitelisting check can be circumvented by changing the price
Severity	LOW SEVERITY
Description	A user can bypass the whitelist restriction for a zero price by simply setting the price to zero during the `updateOfferPrice` function.
Recommendation	Consider not allowing that if the offer is not a whitelisted offer.
Resolution	RESOLVED

Issue #12	Whitelisting logic is flawed
Severity	LOW SEVERITY

Description	The contract intends to whitelist tokens for a specific buyer address which includes gifting the token. However, the following check is flawed:
	
	`// Make sure price is greater than 0 if not whitelisted`
	`if (price == 0 && buyer != address(0)) revert InvalidValueProvided();`

Recommendation	Consider reverting if `buyer == address(0)`.
Resolution	RESOLVED

Issue #13	Checks-effects-interaction pattern is not adhered to
Severity	LOW SEVERITY

Description	One or more sections within the contract violate the check-effects-interactions pattern. This makes the contract vulnerable to reentrancy attacks.
	
	L170
	`function buy(uint256 id) external {`
	`    [...]`
	`    // If this is a whitelist offer check msg.sender`
	`    if (_offer.buyer != address(0) && _offer.buyer != msg.sender) revert UnauthorizedCaller();`
	`    _stableCoin.transferFrom(msg.sender, address(this), _offer.price);`
	`    // Compute fees and user payout`
	`    uint256 fee = (_offer.price * _feePercent) / PRECISION;`
	`    unchecked {`
	`        _fees += fee;`
	`    }`
	`    [...]`
	`}`
	
	The transfer is done before state changes.

Recommendation	Consider rewriting the code to adhere to checks-effects-interactions.
Resolution	RESOLVED

Issue #14	Hardcoded dust amount may affect usability of tokens with high value
Severity	INFORMATIONAL

Location	L96-97
	`// Make sure amount is greater than one token`
	`if (amount < 10 ** IERC20Metadata(token).decimals()) revert InvalidValueProvided();`

Description	The `addOffer` function checks that the amount of sale tokens to sell is greater than 1 full token as above. This is done to prevent creating offers for negligible amounts (so-called dust). However, using the absolute amount of tokens (`10 ** decimals()`) may affect the usability of the marketplace in case the sell token has a high value/price like for example 1000 USD per 1 token.
Recommendation	Consider configuring the minimum amount per token in the `ArborSale` contract upon deployment along with the initial price based on the expected value of the token. Alternatively, consider checking that the price is higher than a certain amount of stablecoin tokens.
Resolution	ACKNOWLEDGED

Issue #15	Typographical issues
Severity	INFORMATIONAL
Description	L39-42 All events are unused, consider using them.
Recommendation	Consider fixing the typographical issues.
Resolution	RESOLVED

2.6 ArborSale

`ArborSale` serves as the core component of a decentralized presale platform, responsible for facilitating the token sale process.

Admin-authorized, this contract is seeded with a specific amount of sale tokens that are later allocated to users based on predefined criteria. Users who have previously deposited stablecoins into `ArborDeposit` can have their funds migrated to the `ArborSale` contract where they will receive their allocated share of sale tokens.

During contract initialization, `_price` is fixed, which is likely denominated in a `1e18` format, where `1e18` corresponds to a price of one. However, ultimately, this is based on the sale token decimal value.

Post-allocation, users have the flexibility to either withdraw their tokens directly or list them on the `ArborMarketplace` for secondary trading. Notably, listing tokens on the marketplace locks them within the `ArborSale` contract, marking them as part of an active offer and hence not withdrawable until either the offer is removed or executed.

This contract has been designed with robust security measures to ensure that only authorized users and administrators can perform sensitive actions, thereby preserving the integrity and fairness of the token sale.

2.6.1 Privileged Functions

- `withdrawEarnings`
- `withdrawLeftover`

2.6.2 Issues & Recommendations

Issue #16	Governance issue
Severity	GOVERNANCE
Description	The admin can simply withdraw all allocated sale tokens without accounting for these funds that have been allocated to users.
Recommendation	Consider restricting this function accordingly.
Resolution	ACKNOWLEDGED

Issue #17	Supply management is flawed
Severity	MEDIUM SEVERITY

Description	The supply management is flawed, as it will calculate an incorrect value if the contract is funded twice. To easily illustrate this issue consider the following PoC:
	
	1. Admin deposits 1 000 sale tokens into the contract
	`-> _supply = 1 000`
	
	2. `batchParticipation` is called, resulting in the whole supply being allocated towards presale participants
	`-> _supply = 0`
	
	3. Users can now either withdraw/keep funds in the contract/ create an offer
	
	4. Consider the scenario where only 500 tokens are withdrawn, i.e., the contract still has a balance, either by simply keeping them in the contract or creating an offer
	`-> supply = 0, balance = 500`
	
	5. The team decides to start a second presale, now again seeding the contract with 1000 tokens:
	`_supply += _saleToken.balanceOf(address(this)) - _supply;`
	
	Therefore, the supply will now become 1900, indicating that 1900 tokens can be distributed in the new participation. This issue happens because not all tokens from the previous allocation have been withdrawn, resulting in a larger balance.
	
	This will then result in potentially distributing 1900 tokens for the second round, which will then not be covered by the tokens in the contract, since from the first round, participants can still withdraw 900 tokens.
	
	Moreover, this function is publicly callable — in the scenario that the presale allocation was distributed the first time, anyone can simply call this function and increase the `_supply` variable.

Recommendation	Consider properly adjusting the supply within the `depositTokens` function — this can be done by simply deducting the unclaimed balance within the `_supply` setting. Another option is to not allow a second round if the first round has not been cleared yet. Additionally, consider only allowing the admin to call this function.
Resolution	RESOLVED

Issue #18	**The check that a user has already participated can be circumvented**
Severity	MEDIUM SEVERITY

Description	`batchParticipation` executes the following check:
	
	`// Check that user has not participated already`
	`if (_saleTokenBalance[account] > 0) revert AlreadyParticipated(account);`
	
	This check can be circumvented by simply listing an offer or withdrawing tokens (whenever it happens that the `batchParticipation` function is called multiple times). Moreover, it would block users that have purchased tokens via the marketplace.

Recommendation	Consider simply setting a nonce to true after the first participation. However, it should be noted that this scenario will also override any existing balance from users that might have purchased tokens from the marketplace, therefore the solution would include not overriding the balance but simply adding towards it. We strongly recommend thinking about the desired business logic and resolve this issue accordingly.
Resolution	ACKNOWLEDGED

Issue #19	`SaleToken` can be stablecoin
Severity	LOW SEVERITY
Description	In such a scenario as described above, the balance would return a flawed amount, resulting in `_supply` becoming too large.
Recommendation	Consider adding an explicit check.
Resolution	ACKNOWLEDGED

Issue #20	Checks-effects-interaction pattern is not adhered to
Severity	LOW SEVERITY

Description	Seceral sections within the contract violate the check-effects-interactions pattern. This makes the contract vulnerable to reentrancy attacks.
	
	L66
	`function depositTokens(uint256 supply_) external {`
	`    _saleToken.safeTransferFrom(msg.sender, address(this), supply_);`
	`    // Support taxables`
	`    _supply += _saleToken.balanceOf(address(this)) - _supply;`
	`    emit TokensDeposited(_supply);`
	`}`
	
	L79
	`function batchParticipation(address[] calldata accounts, uint256[] calldata amounts, uint256 total) external {`
	`    if (msg.sender != _arborDeposit) revert UnauthorizedCaller();`
	`    _stableCoin.safeTransferFrom(msg.sender, address(this), total);`
	`    // Gas optimizations`
	`    uint256 price_ = _price;`
	`    uint256 supply_ = _supply;`
	`    for (uint i; i < accounts.length; i++) {`
	`        [...]`
	`    }`
	`    // Reduce the supply`
	`    _supply = supply_;`
	`}`
	
	In the sections above, the transfer is done before all state changes.

Recommendation	Consider strictly adhering to the checks-effects-interactions pattern.
Resolution	ACKNOWLEDGED

2.7 ArborSalesFactory

`ArborSalesFactory` operates as a dynamic and flexible platform for deploying instances of presale contracts, enabling a streamlined and efficient launch of new presales on the Arbor ecosystem.

Utilizing a contract cloning methodology akin to the practices found in OpenZeppelin's Clone library, `ArborSalesFactory` facilitates the creation of new presale contracts—referred to as `ArborSale` contracts—through a non-deterministic deployment pattern.

In detail, `ArborSalesFactory` is designed with a primary objective of deploying multiple `ArborSale` contracts, each representing an individual presale. The contract inherently allows the authorized administrator to deploy these clones from a specified implementation, introducing a degree of modularity and upgradeability to the overall architecture.

An essential functionality embedded within `ArborSalesFactory` is the capability to alter the implementation address. This is especially crucial in environments that require upgradeability and future adaptability, as it allows developers or administrators to pivot or upgrade the core logic of the cloned contracts without necessitating a full redeployment of the factory contract itself. This effectively allows the underlying logic or structure of the subsequent presales to be improved, optimized, or modified according to evolving requirements, technological advancements, or to address potential security vulnerabilities identified post-deployment.

To foster transparency and traceability, every deployment of a new `ArborSale` contract is meticulously logged and stored, providing an accessible history of all presales instantiated by the `ArborSalesFactory`.

This not only enhances the verifiability and auditability of the presale deployments but also provides a mechanism for interfacing with these deployed contracts in future interactions or integrations within the Arbor ecosystem.

2.7.1 Privileged Functions

- `onlyAdmin`
- `setImplementation`
- `deploy`

2.7.2 Issues & Recommendations

No issues found.

2.8 ArborStaking

`ArborStaking` plays a pivotal role within the Arbor network and is a structured platform that is meticulously devised to navigate the sophisticated landscape of token staking, conversion, and presale allocation.

The contract includes an adept staking mechanism where users can stake their ARBR tokens which secures their allocation for upcoming token presales, and they can also withdraw their staked ARBR tokens by paying a fee.

Notably HARBR token holders have an advantage — for every HARBR token held in their wallet balance, users are presented with a presale allocation that quintuples according to a staked ARBR token. Such a mechanism not only fortifies the utility of the HARBR token but also curates a strategically favorable environment for its holders within the presale context.

Diving into the token conversion avenues facilitated by the contract, users find themselves amidst two distinctive paths when seeking to convert HARBR to ARBR: an Instant Conversion and a 100-day Conversion Process.

The former, while providing immediate gratification through swift conversions, enforces the `_instantConversionFee`, thereby establishing a transparent cost for immediacy.

Contrarily, the latter – the 100-day Conversion – operates under a presumably reduced fee, calculated off-chain, crafting an alternative for users whose conversion requirements are not bound by temporal immediacy.

A crucial facet, particularly in the realm of user autonomy, is the contract's conversion cancellation facility. While users maintain the liberty to abort any ongoing conversion processes, this functionality is not without cost — an incurred fee, which is also calculated off-chain, ensures that the cancellation mechanism is employed judiciously, providing a stabilizing factor within the token conversion

ecosystem. Furthering the versatility of the token conversion mechanisms, the contract also permits the reversion of HARBR tokens back to ARBR tokens, echoing the aforementioned conversion methodologies. Instant Conversion, albeit with its imposing 50% fee, coexists with the 100-day Conversion Process, which adheres to the off-chain fee calculation methodology, preserving the network's predilections for off-chain operational elements.

The admin can also freely convert any ARBR tokens to HARBR tokens, instantly and without any fee.

2.8.1 Privileged Functions

- `adminBatchedInstantConversionAToH`
- `withdrawFees`
- `setFees`

2.8.2 Issues & Recommendations

Issue #21	Checks-effects-interaction is not adhered to
Severity	LOW SEVERITY
Description	Multiple sections within the contract violate the check-effects-interactions pattern. This makes the contract vulnerable to reentrancy attacks.

L61~

```
function stakeArbr(uint256 amount, uint256
expirationTimestamp, bytes[2] calldata signatures) external
{
    // Verify signatures
    verifySignatures(amount, 0, expirationTimestamp,
this.stakeArbr.selector, signatures);
    // Transfer arbr from user to this contract
    _arbr.transferFrom(msg.sender, address(this), amount);

    […]
}
```

The arbr transfer is executed before the state variable change.

L115~

```
function adminBatchedInstantConversionAToH(
    address[] calldata accounts,
    uint256[] calldata amounts,
    uint256 total
) external onlyAdmin {
    uint256 _length = accounts.length;
    if (_length != amounts.length) revert
ArrayLengthMismatch();
    _arbr.transferFrom(msg.sender, address(this), total);
    uint256 _total;
    [...]
}
```

The transfer is executed at the beginning of the function.

L220~

```
function hundredDayConversionHToA(
    uint256 amount,
    uint256 expirationTimestamp,
    bytes[2] calldata signatures
) external {
    // Verifu signatures
    verifySignatures(amount, 0, expirationTimestamp,
this.hundredDayConversionHToA.selector, signatures);
    // Burn harbr
    _harbr.burn(msg.sender, amount);
    // Add burned amount to amount of harbr in conversion
    unchecked {
        _harbrInConversion[msg.sender] += amount;
    }
    // Emit event
    emit HundredDayConversionFromHToA(msg.sender, amount);
}
```

The burn is executed before the state variable change.

L345~

```
function withdrawFees() external onlyAdmin {
    _arbr.transfer(msg.sender, _accumulatedFees);
    unchecked {
        _accumulatedFees = 0;
    }
}
```

The transfer out is done before the fee reset.

Recommendation	Consider strictly adhering to the checks-effects-interactions pattern.
Resolution	RESOLVED

Issue #22	Lack of fee validation
Severity	LOW SEVERITY

Description	Both fees can be set up to 100%:
	`function _setFees(uint256 instantConversionFee_, uint256 redemptionFee_) private {`
	`    if (instantConversionFee_ > PRECISION || redemptionFee_ > PRECISION) {`
	`        revert InvalidValueProvided();`
	`    }`
	`    _instantConversionFee = instantConversionFee_;`
	`    _redemptionFee = redemptionFee_;`
	`}`
	
	This might result in undesirable side-effects for users, if for example the admin key is compromised.

Recommendation	Consider setting a reasonable fee limit.
Resolution	RESOLVED

Issue #23	Lack of explicit error message
Severity	INFORMATIONAL
Description	The contract has several sections where an explicit error message is not returned, for example within the `withdrawArbr` function, `_arbrBalance` will underflow instead of reverting if there is an insufficient balance.
Recommendation	Consider implementing explicit revert messages for these scenario.s
Resolution	ACKNOWLEDGED

Issue #24	Typographical issues
Severity	INFORMATIONAL

Description	L164, 238, 284, 307
	`// Make sure fee is less than 50%`
	
	This comment does not match the source code — either change the source code to reflect this, or change the commit towards that the fee cannot be greater than the amount.
	
	L217
	`// Verifu signatures`
	
	"Verify" is mis-spelled as "verifu".

Recommendation	Consider fixing the typographical issues.
Resolution	RESOLVED

2.9 HARBR

HARBR is a customized ERC-20 token that introduces specific minting and burning mechanisms to manipulate the token's total supply. It is uniquely engineered to serve particular use-cases, specifically being utilized within a conversion mechanism within the `ArborStaking` contract. The minting and burning processes appear to be tightly controlled, only executable under specific conditions within the conversion logic.

The contract incorporates access control logic via the `onlyAuthorized` modifier, which allows only the admin and the `ArborStaking` contract to mint and burn tokens. The admin can change the `ArborStaking` address at any time.

It is important to mention that this token is not transferable — once a user obtains it by converting ARBR -> HARBR, it will be simply sit in a user's wallet, granting allocation power towards a user which can then be used to participate in the presale. However, this logic is handled off-chain.

2.9.1 Privileged Functions

- `burn (onlyAuthorized)`
- `mint (onlyAuthorized)`
- `setArborStakingContract`

2.9.2 Issues & Recommendations

Issue #25	Governance privileges
Severity	GOVERNANCE
Description	The admin can freely change the `_arborStakingContract` as well as freely mint and burn tokens. This is a large governance privilege which can result in a loss of funds as well as resulting in a DoS for the cross-contract logic within the `ArborStaking` contract.
Recommendation	Consider using a multi-signature contract as admin.
Resolution	ACKNOWLEDGED A multi-signature wallet will be used as the admin.

2.10 MultiSigWallet

`MultiSigWallet` provides a robust multi-signature wallet implementation, designed to require a certain number of out of a group of owners to approve a transaction before it can be executed. It is developed to ensure an added layer of security through collective authorization of transactions, and it is derived from the renowned Gnosis MultiSig Wallet.

2.10.1 Privileged Functions

- `addOwner`
- `removeOwner`
- `replaceOwner`
- `changeRequirement`
- `confirmTransaction`
- `revokeConfirmation`
- `executeTransaction`

2.10.2 Issues & Recommendations

Issue #26	**pop is not executed**
Severity	MEDIUM SEVERITY
Description	Whenever an owner is removed, either the swap and pop logic is executed or a direct pop is applied. However, the pop call is not executed due to missing parenthesis.
Recommendation	Consider fulfilling this call.
Resolution	RESOLVED

Issue #27	**`replaceOwner` lacks zero check**
Severity	INFORMATIONAL
Description	This function does not explicitly check the zero address case for the new owner. This might lead into an undesired owner length.
Recommendation	Consider adding a zero-check.
Resolution	RESOLVED

Issue #28	**`getTransactionCount` and `getTransactionIds` might run out of gas**
Severity	INFORMATIONAL
Description	Both functions loop over the `transactionCount` variable. This can lead to an out-of-gas error in rare edge-cases where there is a large transaction count.
Recommendation	Practically, we do not see this as an issue as there are no storage operations within the loop, which means that `transactionCount` would need to be incredibly large for such an issue to occur. However, if the team wishes to fix this issue, we recommend switching to a paginated loop.
Resolution	ACKNOWLEDGED

2.11 Signatories

`Signatories` operates as a cryptographic signature verification mechanism where transactions and certain critical paths within a smart contract system require approval from two distinct signatory addresses.

It uses OpenZeppelin's ECDSA library to properly recover the signer address for a specific message and signature. The admin of the contract can freely change both signatories.

2.11.1 Privileged Functions

- `setAdmin`
- `setSignatories`

2.11.2 Issues & Recommendations

Issue #29	Signatories can be the same address
Severity	LOW SEVERITY
Description	To create an extra layer of security, the contract is designed to have two signers. However, since both signers can be set to the same address, this security layer can be bypassed.
Recommendation	Consider not allowing to set both signers to the same address.
Resolution	RESOLVED


PALADIN
BLOCKCHAIN SECURITY