PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Preliminary Report

For Abachi
(Token & Redemption)

04 January 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 Abachi's token and pre-sale redemption contracts on the Polygon 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	Abachi (Token & Redemption)
URL	https://www.abachi.io/
Platform	Polygon
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
Abachi	0x6d5f5317308c6fe7d6ce16930353a8dfd92ba4d7	✓ MATCH
AbachiRedemption	0xd3255b8B12E67006f822FF1F8cB2beDa065345DE	✓ MATCH
AbachiAuthority	0x4b2bd29b81d32e3dbceb47260f0bbc76a6a0b8cd	✓ MATCH
AbachiAccessControlled	Dependency	✓ MATCH
Policy	Dependency	✓ MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
🔴 High	0	-	-	-
🟠 Medium	1	-	-	1
🟡 Low	3	1	-	2
🟣 Informational	8	1	-	7
Total	**12**	**2**	**-**	**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 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 Abachi

ID	Severity	Summary	Status
01	LOW	`mint` function can be used to mint large amounts of tokens by vault owners	ACKNOWLEDGED
02	INFO	Gas optimization: Contract uses hardcoded strings in SafeMath functions	ACKNOWLEDGED
03	INFO	`permit` can be frontrun and cause denial of service	ACKNOWLEDGED

1.3.2 aAbachi

ID	Severity	Summary	Status
04	INFO	Lack of events for `pause`, `unpause` and `withdrawTo`	ACKNOWLEDGED
05	INFO	Lack of `safeTransfer`/`safeTransferFrom` usage within `_swapFor` and `withdrawTo`	✓ RESOLVED
06	INFO	aABI and ABI can be made immutable	ACKNOWLEDGED
07	INFO	aABI transferred to the contract is never burned	ACKNOWLEDGED
08	INFO	`swapFor` could be abused for phishing	ACKNOWLEDGED

1.3.3 AbachiAuthority

ID	Severity	Summary	Status
09	LOW	Wrong parameters on events for `policy`, `vault`, `guardian` and `governor`	ACKNOWLEDGED

1.3.4 AbachiAccessControlled

ID	Severity	Summary	Status
10	INFO	Gas Optimization: `UNAUTHORIZED` can be constant	ACKNOWLEDGED

1.3.5 Policy

ID	Severity	Summary	Status
11	MEDIUM	The last policy can be reclaimed	ACKNOWLEDGED
12	LOW	New owner variable is `internal`	✓ RESOLVED

2 Findings

2.1 Abachi

Abachi is a simple ERC20 token. It implements the `permit` functionality which can be used to change an account's ERC20 allowance by presenting a message signed by the account without the actual need of an approval transaction. This functionality does not cost any gas. The Abachi token will be used as the main token within the Abachi ecosystem.

Tokens can be minted only by the entities that have the policy of `onlyVault`. Tokens can be burned using the `burn` and `burnFrom` functions. The former burns from the balance of the transaction sender, while the latter allows an address to burn another address' tokens, provided that the executing party has been granted sufficient allowance.

2.1.1 Privileged Roles

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

- `mint`
- `setAuthority`

2.1.2 Issues & Recommendations

Issue #01	**`mint` function can be used to mint large amounts of tokens by vault owners**
Severity	● LOW SEVERITY
Description	The contract contains a `mint` function which allows addresses with the `onlyVault` permission to mint new tokens. This could be used to mint and dump tokens by the governance addresses with the `onlyVault` permission either with malicious intent or by being hacked. This risk is prevalent amongst less-reputable projects, and any mints can be prominently seen on the Blockchain.
Recommendation	Consider being forthright if this `mint` function is to be used by letting your community know how much was minted, where they are currently stored, if a vesting contract was used for token unlocking, and finally the purpose of the mints.
Resolution	● ACKNOWLEDGED

Issue #02	Gas optimization: Contract uses hardcoded strings in SafeMath functions
Severity	● INFORMATIONAL
Location	Line 38 `uint256 decreasedAllowance_ = allowance(account_, msg.sender).sub(amount_, "ERC20: burn amount exceeds allowance");`
Description	The contract injects the error message into SafeMath. This is known to cost extra gas, even on the happy path, as it causes memory allocation.
Recommendation	Consider checking the identity explicitly using a `require` statement and then using non-SafeMath to do the subtractions and additions instead. SafeMath has also created the `trySub` and `tryAdd` functions in more recent versions to address this gas usage concern.
Resolution	● ACKNOWLEDGED

Issue #03	permit can be frontrun and cause denial of service
Severity	● INFORMATIONAL
Description	Many of the tokens contain a transactionless approval scheme based on EIP-2612. This mechanism is most well-known by users when they break up Uniswap LP tokens without having to explicitly send an approval transaction, instead they just have to approve a signature. Just like with Uniswap permits, if `permit` is executed twice, the second execution will be reverted. It is thus in theory possible for a bot to pick up `permit` transactions in the mempool and execute them before a contract can. The issue with this is that the rest of said contract functionality would be lost as well. This could allow for denial of service. This is present in the: ERC20Permit dependency contract
Recommendation	Within derivative protocols, one can consider using try-catch for permit and validating the approval afterwards.
Resolution	● ACKNOWLEDGED

2.2 AbachiRedemption

AbachiRedemption is a contract that will be used to let users convert their presale Alpha Abachi (aABI) to the main Abachi token (ABI). The swap ratio is 1 aABI = 1 ABI.

2.2.1 Privileged Roles

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

- `withdrawTo`
- `pause`
- `unpause`
- `renounceOwnership`
- `transferOwnership`

2.2.2 Issues & Recommendations

Issue #04	Lack of events for `pause`, `unpause` and `withdrawTo`
Severity	● INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions.
Resolution	● ACKNOWLEDGED

Issue #05 | **Lack of `safeTransfer/safeTransferFrom` usage within `_swapFor` and `withdrawTo`**

Severity

INFORMATIONAL

Location

Line 74
```
require(ABI.transfer(_recipient, _amount), "Failed to
transfer ABI");
```

Line 81
```
require(
    aABI.transferFrom(msg.sender, address(this), _amount),
    "Failed to transfer aABI"
);
```

Line 86
```
require(ABI.transfer(_recipient, _amount), "Failed to
transfer ABI");
```

Description

In the above functions, the `transfer/transferFrom` methods are used to transfer tokens. This will not work for tokens that will return false on transfer (or malformed tokens that do not have a return value).

Within the AbachiRedemption contract, this will not cause any issue as the tokens that are used for transfer are validated to be ERC20 compliant. This issue is included as a reminder for the developers that such patterns are not always desired in our attempt to increase awareness of issues that are not always known by developers.

Recommendation

No action is required.

Resolution

✓ RESOLVED

The client has stated they are now aware of this issue.

Issue #06	aABI and ABI 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 aforementioned variables explicitly immutable.
Resolution	● ACKNOWLEDGED

Issue #07	aABI transferred to the contract is never burned
Severity	● INFORMATIONAL
Description	While aABI is transferred into the contract, it is not burned. This causes the aABI supply to remain high while in reality these tokens are taken out of circulation.
Recommendation	Consider whether this is desirable. If not, consider using `burnFrom` instead of `transferFrom` (alternatively a burn call can be made after `transferFrom`).
Resolution	● ACKNOWLEDGED

Issue #08	`swapFor` could be abused for phishing
Severity	● INFORMATIONAL
Description	Presently, the contract allows you to immediately send your swapped tokens to another wallet using the `swapFor` method. If the frontend were to be hacked, this could cause users to lose their Abachi tokens even if they are careful to check their contract interactions.
Recommendation	Consider only allowing `swapFor` to be called by a contract `msg.sender` or a whitelisted set of addresses.
Resolution	● ACKNOWLEDGED

2.3 AbachiAuthority

The AbachiAuthority is the main contract that defines the RBAC (Role Based Access Control) functionality throughout the Abachi ecosystem. This contract is used to give different tiers of permissions to different entities. These permissions are used to restrict different actions throughout the contracts within the Abachi ecosystem.

2.3.1 Privileged Roles

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

- `pushGovernor`
- `pushGuardian`
- `pushPolicy`
- `pushVault`
- `pullGovernor`
- `pullGuardian`
- `pullPolicy`
- `pullVault`

2.3.2 Issues & Recommendations

Issue #09	**Wrong parameters on events for policy, vault, guardian and governor**
Severity	● LOW SEVERITY

Location	Lines 57-61 (example)
	`function pushGuardian(address _newGuardian, bool _effectiveImmediately) external onlyGovernor {`
	`    if( _effectiveImmediately ) guardian = _newGuardian;`
	`    newGuardian = _newGuardian;`
	`    emit GuardianPushed(guardian, newGuardian, _effectiveImmediately);`
	`}`

Description	A governor can push a permission with `_effectiveImmediately` `true` and the push/pull strategy for giving permissions is skipped. By doing this, the events emitted by the permission functions are wrong as the from parameter will show the new permission owner not the old one.
Recommendation	Consider caching the old permission's owner and use it in the emitting of the event.
Resolution	● ACKNOWLEDGED

2.4 AbachiAccessControlled

AbachiAccessControlled is an abstract contract that uses AbachiAuthority contract to define modifiers that can be used to define an RBAC (Role Based Access Control) mechanism across different contracts within the Abachi ecosystem.

2.4.1 Privileged Roles

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

- `setAuthority`

2.4.2 Issues & Recommendations

Issue #10	Gas Optimization: `UNAUTHORIZED` can be `constant`
Severity	INFORMATIONAL
Description	The `UNAUTHORIZED` variable is used as a return message for different checks inside the contract. As this variable never changes, it can be made a `constant` to save gas.
Recommendation	Consider making this variable a `constant`.
Resolution	ACKNOWLEDGED

2.5 Policy

Policy is a contract that is used to define one of the permissions within the Abachi ecosystem. This mimics the push/pull approach of ownership pattern, meaning the previous owner needs to push the ownership to the new owner and the new owner needs to claim it.

2.5.1 Privileged Roles

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

- `renouncePolicy`
- `pushPolicy`
- `pullPolicy`

2.5.2 Issues & Recommendations

Issue #11	The last policy can be reclaimed
Severity	● MEDIUM SEVERITY

Location	Lines 37-40
	`function renouncePolicy() public virtual override onlyPolicy() {`
	`    emit OwnershipPushed( _owner, address(0) );`
	`    _owner = address(0);`
	`}`
	
	Lines 48-52
	`function pullPolicy() public virtual override {`
	`    require( msg.sender == _newOwner, "Ownable: must be new owner to pull");`
	`    emit OwnershipPulled( _owner, _newOwner );`
	`    _owner = _newOwner;`
	`}`

Description	Within the policy implementation, the policy can be renounced. However, the last policy can reclaim this at any moment as the new policy variable was never reset. It should also be noted that before the first policy transfer is made, the zero address can claim the policy. This is hardly problematic as the zero contract is not known to be owned by anyone and probabilistically speaking, under the current address scheme, the chances of anyone ever owning it are negligible.
Recommendation	Consider using BoringOwnable implementation. https://github.com/boringcrypto/BoringSolidity/blob/f05de5f250056730c3fd3e5a5d1e572c2d113023/contracts/BoringOwnable.sol
Resolution	● ACKNOWLEDGED No changes were made. The client will be upgrading the contract with staking and bonding contract implementations.

Issue #12	New owner variable is `internal`
Severity	● LOW SEVERITY
Location	Line 18 `address internal _newOwner;`
Description	Within the policy implementation contract the variable that denotes the new owner is internal. 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 new owner variable as `public`.
Resolution	✓ RESOLVED The variables on the Policy are marked as `public`.

PALADIN

BLOCKCHAIN SECURITY