PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Abachi (Presale)

14 December 2021

paladinsec.co

info@paladinsec.co

Table of Contents

Table of Contents 2

Disclaimer 3

1 Overview 4

1.1 Summary 4

1.2 Contracts Assessed 4

1.3 Findings Summary 5

1.3.1 PresaleAbachi 6

1.3.2 aAbachi 6

2 Findings 7

2.1 PresaleAbachi 7

2.1.1 Issues & Recommendations 8

2.2 aAbachi 13

2.2.1 Token Overview 13

2.2.2 Privileged Roles 14

2.2.3 Issues & Recommendations 15

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.

1 Overview

This report has been prepared for Abachi's presale 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 (Presale)
URL	https://www.abachi.io/
Platform	Polygon
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`PresaleAbachi`	`0x62fb6f77cebc8500448fbfb8d4f875f01d01531d`	MATCH
`aAbachi`	`0x835ba011ec70e695879185df3d46939dbfbef7e5`	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	1	1	-	-
Medium	1	1	-	-
Low	1	-	-	1
Informational	7	-	-	7
Total	**10**	**2**	**-**	**8**

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 PresaleAbachi

ID	Severity	Summary	Status
01	INFO	payout can be precalculated and set as a constant	ACKNOWLEDGED
02	INFO	State variables can be set to constant or immutable for gas optimization	ACKNOWLEDGED
03	INFO	Lack of events emitted for state changes	ACKNOWLEDGED
04	INFO	Unnecessary comparison with boolean types in require statement	ACKNOWLEDGED
05	INFO	`getUserRemainingAllocation` and `deposit` has unnecessary calculation	ACKNOWLEDGED
06	INFO	Unused fields in UserInfo	ACKNOWLEDGED

1.3.2 aAbachi

ID	Severity	Summary	Status
07	HIGH	Integer overflow allows minting of more tokens than `maxMint`	RESOLVED
08	MEDIUM	Presale contract address can be modified	RESOLVED
09	LOW	`mintAuction` cannot be called if the sum of `totalSupply` and `auctionMintAmount` exceeds `maxMint`	ACKNOWLEDGED
10	INFO	State variables can be set to constant or immutable for gas optimization	ACKNOWLEDGED

2 Findings

2.1 PresaleAbachi

The presale contract allows whitelisted users to purchase aABI tokens using DAI tokens, at a fixed price of 20 DAI for 1 aABI token. Tokens can only be purchased during between the start and end of the sale, which is settable by the owner.

There are no aABI tokens stored in the contract as they will be minted directly to the user's address when a purchase is done. The owner is able to withdraw any ERC20 token in the contract.

2.1.1 Issues & Recommendations

Issue #01	payout can be precalculated and set as a constant
Severity	INFORMATIONAL
Description	As the price and cap are already predefined and hardcoded into the contract, the payout can be set to a constant too, to optimize for gas in the `deposit` function.
Recommendation	Add the following constant payout state variable: `uint256 public constant payout = 50 * 10e9` If this is done, the price state variable can be removed too.
Resolution	ACKNOWLEDGED

Issue #02	State variables can be set to constant or immutable for gas optimization
Severity	INFORMATIONAL
Description	The following state variables can be set as constant to reduce gas consumption: - `price` - `cap` The following state variables can be set as immutable: - `aABI` - `DAI` - `DAO`
Recommendation	Update the above state variables to constant or immutable.
Resolution	ACKNOWLEDGED

Issue #03	Lack of events emitted for state changes
Severity	INFORMATIONAL
Description	Events are not emitted for the following functions with state changes: - `addWhitelist` - `addMultipleWhitelist` - `removeWhitelist` - `togglePause`
Recommendation	Emit events for the above functions.
Resolution	ACKNOWLEDGED

Issue #04	Unnecessary comparison with boolean types in require statement
Severity	INFORMATIONAL
Location	Line 133 `require(whitelisted[msg.sender] == true, 'msg.sender is not whitelisted user');`
Description	In the above `require` statement, there is an unnecessary comparison with `true`.
Recommendation	The require statement can be modified as such: `require(whitelisted[msg.sender], 'msg.sender is not whitelisted user');`
Resolution	ACKNOWLEDGED

Issue #05	**`getUserRemainingAllocation` and deposit has unnecessary calculation**
Severity	INFORMATIONAL
Description	As the amount a user can purchase has to be equal to the cap (1000 DAI tokens), once a user purchases, the remaining allocation for the user would be 0. Thus, the logic for whether a user can purchase can be done using a boolean instead of unnecessary arithmetic.

Recommendation	In `deposit`, the following is unnecessary and can be removed:
	
	Line 137~
	`require(`
	`cap >= user.amount.add(_amount),`
	`'new amount above user limit'`
	`);`
	`user.amount = user.amount.add(_amount);`
	
	Instead a boolean such as purchased can be used for a user.
	
	E.g.
	`mapping(address => boolean) public purchased;`

Check if a user has already purchased before, and then set the purchased for an address to be true once deposit is done.

```
function deposit(uint256 _amount) external checkIfPaused {
    require(started, 'Sale has not started');
    require(!ended, 'Sale has ended');
    require(whitelisted[msg.sender] == true, 'msg.sender is
not whitelisted user');
    require(_amount == cap, 'Deposit must equal 1000 DAI');
    require(!purchased[msg.sender], “Already purchased”);
    purchased[msg.sender] = true;

    totalRaisedDAI = totalRaisedDAI.add(_amount);

    uint256 payout =
_amount.mul(1e18).div(price).div(1e9); // aABI to mint for
_amount

    totalDebt = totalDebt.add(payout);
    DAI.safeTransferFrom( msg.sender, DAO, _amount );
    IAlphaABI( address(aABI) ).mint( msg.sender, payout );
    emit Deposit(msg.sender, _amount);
}
```

In `getUserRemainingAllocation`, it can return either 1000 or 0 depending on whether the user has already purchased.

```
function getUserRemainingAllocation(address _user) external
view returns ( uint256 ) {
    if (purchased[_user]) {
        return 0;
    }
    return cap;
}
```

Resolution ACKNOWLEDGED

Issue #06	Unused fields in UserInfo
Severity	INFORMATIONAL
Description	In the UserInfo struct, the following fields are never used in the contract: - `debt` - `claimed`
Recommendation	As mentioned in a previous issue about unnecessary calculation, it appears that the amount field is unnecessary too. The struct can be removed and replaced with a mapping that tracks if an address has already made a purchase. `mapping(address => boolean) public purchased;`
Resolution	ACKNOWLEDGED

2.2 aAbachi

The aAbachi token is an ERC20 token with 9 decimals. A maximum supply of 192,500 tokens can be minted, with a portion of 142,500 tokens mintable for auction, by the owner.

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 the latter has granted the former sufficient allowance.

2.2.1 Token Overview

Address	`0x835ba011ec70e695879185df3d46939dbfbef7e5`
Token Supply	192,500
Decimal Places	9
Transfer Max Size	No maximum
Transfer Min Size	No minimum
Transfer Fees	None

2.2.2 Privileged Roles

The following functions can be called by the Policy role:

- `setPresale`

The following functions can be called by the Presale role:

- `mint`

The following functions can be called by the owner:

- `mintAuction`

2.2.3 Issues & Recommendations

Issue #07	Integer overflow allows minting of more tokens than `maxMint`
Severity	HIGH SEVERITY

Location	Line 54~
	`function mint(address account_, uint256 amount_) external onlyPresale() {`
	`    require(totalSupply() + amount_ <= maxMint, 'Exceeds maximum allowed tokens to be minted');`
	`    _mint(account_, amount_);`
	`}`

Description	In the `mint` function, the check to ensure that the amount minted will not cause the `totalSupply` to exceed `maxMint` is done using normal addition, not `SafeMath`. This allows an amount which causes the sum of `totalSupply` and `amount_` to overflow and be less than `maxMint` to be minted, even if amount itself exceeds `maxMint`.
Recommendation	Use SafeMath addition, or use a solidity compiler version 0.8.0 or higher which has built in integer overflow/underflow protection.
Resolution	RESOLVED SafeMath is now used for addition.

Issue #08	Presale contract address can be modified
Severity	MEDIUM SEVERITY

Location	Line 12~
	`function setPresale( address presale_ ) external onlyPolicy() returns ( bool ) {`
	`    _presale = presale_;`
	
	`    return true;`
	`}`

Description	The presale address has the privilege of minting tokens using the `mint` function. It can be set by the policy role using this function. This allows the presale contract to be changed at any time, and can be used to arbitrarily mint tokens.

Recommendation	Add a check to ensure that the presale address can only be set once.
	
	`function setPresale( address presale_ ) external onlyPolicy() returns ( bool ) {`
	`    require(_presale == address(0), “presale set already”);`
	`    _presale = presale_;`
	
	`    return true;`
	`}`

Resolution	RESOLVED The check to ensure that presale can only be set if it has not been set has been added.

Issue #09	**`mintAuction` cannot be called if the sum of `totalSupply` and `auctionMintAmount` exceeds `maxMint`**
Severity	LOW SEVERITY
Description	If some tokens have already been minted before `mintAuction` is called, resulting in the sum of `totalSupply` and `auctionMintAmount` to exceed `maxMint`, `mintAuction` will always revert.
Recommendation	Consider minting the `auctionMintAmount` in the constructor. If that is done, `hasMintedAuction` and the `mintAuction` function can be removed. If this is done, the owner state variable is no longer required and can be removed.
Resolution	ACKNOWLEDGED

Issue #10	**State variables can be set to constant or immutable for gas optimization**
Severity	INFORMATIONAL
Description	The following state variables can be set as constant to reduce gas consumption: - `auctionMintAmount` - `maxMint` The following state variables can be set as immutable: - `owner`
Recommendation	Update the above state variables to constant/immutable.
Resolution	ACKNOWLEDGED


PALADIN
BLOCKCHAIN SECURITY