PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For 8.Finance

27 October 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 8.Finance on the BNB Smart Chain. 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	8.Finance
URL	https://8.finance/
Network	BNB Smart Chain
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`TokenUpgradeable`	`0xbc46d7ba32f9efd86d2abc6c375a2f35c795b011`	✓ MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
🔴 High	1	1	-	-
🟠 Medium	2	2	-	-
🟡 Low	0	-	-	-
🟣 Informational	5	4	1	-
Total	**8**	**7**	**1**	**-**

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 TokenUpgradeable

ID	Severity	Summary	Status
01	HIGH	Exploiters can completely bypass the fee on transfer by using `transferFrom` instead of `transfer`	✓ RESOLVED
02	MEDIUM	Governance risk: The contract is upgradeable which gives the proxy admin arbitrary power and the fees on transfer can be set up to 100%	✓ RESOLVED
03	MEDIUM	The contract can be compiled on 0.7 which would potentially render it vulnerable to over/underflow vulnerabilities	✓ RESOLVED
04	INFO	Configuration risk: There is presently no way for the owner to exclude wallets from the fees on transfer	✓ RESOLVED
05	INFO	Configuration risk: `setBurnTransferFee` has a different precision	✓ RESOLVED
06	INFO	The transfers might still call `burn` with a zero amount due to rounding	✓ RESOLVED
07	INFO	Typographical errors	PARTIAL
08	INFO	Lack of events for `setBurnTransferFee`	✓ RESOLVED

2 Findings

2.1 TokenUpgradeable

`TokenUpgradeable` is the smart contract for the 8.Finance token. It is an upgradeable ERC20 token which means that the proxy admin can change the token logic at will to any logic the admin pleases. The token has a "burn" feature where a portion of the tokens transferred are burned. The contract is owned by a single owner who can set the burn fee percentage. When tokens are transferred, the burn fee is calculated and deducted from the amount being transferred, and the remaining tokens are transferred to the specified address.

During deployment, the burn fee starts at 0% but we expect it to be changed eventually to a positive value.

A total of 888,888,888 tokens are minted during deployment. This can only be changed by upgrading the contract.

2.1.1 Privileged Functions

- `setBurnTransferFee`
- `transferOwnership`
- `renounceOwnership`

2.1.2 Issues & Recommendations

Issue #01	Exploiters can completely bypass the fee on transfer by using `transferFrom` instead of `transfer`
Severity	● HIGH SEVERITY

Location	Line 114-116
	`function transfer(address to, uint256 amount) public virtual override returns (bool) {`
	`    address owner = _msgSender();`
	`    uint256 fee = amount * burnTransferFee / feeMuplipier;`
	`    if (burnTransferFee > 0) {`
	`        _burn(owner, fee);`
	`    }`
	`    _transfer(owner, to, amount - fee);`
	`    return true;`
	`}`

Description	The 8.Finance token has a fee on transfer. This fee is deducted from all transfers and subsequently burned. The recipient then receives a lesser amount of tokens than what was transferred. However, a malicious exploiter can completely bypass this burn and send the full transfer amount to a recipient by using `transferFrom`. `transferFrom` is similar to `transfer` but is used for approval-based transfers. If the exploiter approves their own wallet, they can simply use `transferFrom` interchangeably with `transfer`. As a result, as `transferFrom` is used by the UniswapV2 router for sales, sales are presently untaxed, while purchases would still be taxed (the pairs use `transfer`) — this might not be great for psychological tokenomics as no fee needs to be paid on sales but it needs to be paid on purchases.

Recommendation

Consider instead levying the fee on any transfer by overriding `_transfer`:

```
_transfer(address from, address to, uint256 amount) internal
override {
    uint256 fee = amount * burnTransferFee / feeMultiplier;
    if (fee > 0) {
        _burn(from, fee);
    }
    super._transfer(from, to, amount - fee);
}
```

Resolution

✓ RESOLVED

The fee on transfer is now taken from both `transfer` and `transferFrom`.

Issue #02	**Governance risk: The contract is upgradeable which gives the proxy admin arbitrary power and the fees on transfer can be set up to 100%**
Severity	● MEDIUM SEVERITY
Description	The token contract is an upgradeable proxy. This means that the project owner, which is the proxy owner, can at any point in time change all the logic from the token to different logic which was not audited by Paladin. This not only poses a risk if the owner is malicious, but also poses a risk if the admin keys are ever compromised (eg. stolen), and the thief uses them to upgrade the contract to a version which mints a massive amount of tokens to their own account to dump and drain the LP pairs with. Finally, the contract owner can presently also set the transfer-tax up to 100%, which seems an excessive privilege to us.
Recommendation	We understand the value of upgradeability. If it is desired or necessary to retain this feature, consider undergoing KYC to address the risk of the admin being malicious. Consider also locking the admin behind a multi-signature set up and/or timelock to strongly mitigate the risk of the keys potentially being stolen. Consider also capping the maximum fee on transfer to a more sensible percentage, eg. 20%.
Resolution	✓ RESOLVED The client has completed KYC with RugDoc.io, one of our trusted partners.

Issue #03	The contract can be compiled on 0.7 which would potentially render it vulnerable to over/underflow vulnerabilities
Severity	● MEDIUM SEVERITY
Location	Line 2 `pragma solidity >=0.7.0 <0.9.0;`
Description	The contract can presently be compiled on both Solidity 0.7 and 0.8. During the transition to 0.8, the Solidity compiler has made important changes: specifically, it now automatically protects math against overflows/underflows. The code itself is not explicitly protected against these so it is crucial that this contract is compiled on 0.8.0 or above.
Recommendation	Consider fixing the pragma: `pragma solidity 0.8.17;`
Resolution	✓ RESOLVED The pragma is now `pragma solidity >= 0.8.2 <= 0.8.4`.

Issue #04	Configuration risk: There is presently no way for the owner to exclude wallets from the fees on transfer
Severity	● INFORMATIONAL
Description	Presently, every single transfer is subject to the fee on transfer. It might be desirable by the owner to exclude certain addresses from this tax (eg. staking contracts). This is presently impossible.
Recommendation	Consider adding an `excludedFromTax` mapping and a governance function to flag certain addresses to be excluded or not. Do remember to add an event and make the function `external`.
Resolution	● ACKNOWLEDGED The client confirmed that they plan to tax transfers from any sender. It is worth noting that this can still be changed through a contract upgrade.

Issue #05	Configuration risk: `setBurnTransferFee` has a different precision
Severity	INFORMATIONAL
Location	Line 26 `burnTransferFee = feePercent / 100;`
Description	The `setBurnTransferFee` is used by the contract owner to configure the burn percentage. However, due to line 26, the `setBurnTransferFee` input is 100 times bigger than what will actually be set as the `burnTransferFee`. Although the owner can simply input their desired precision multiplied by 100, this is rather error prone and might cause the owner to make several mistakes before finally entering in the correct number.
Recommendation	Consider using the same precision and not dividing by 100.
Resolution	✓ RESOLVED The precision is no longer divided by 100.

Issue #06	The transfers might still call burn with a zero amount due to rounding
Severity	INFORMATIONAL

Location	Line 32
	`if (burnTransferFee > 0) {`
	`    _burn(owner, fee);`
	`}`

Description	The transfer function calls `_burn` as soon as `burnTransferFee` is not zero. As `fee = amount * burnTransferFee / feeMuplipier`, the fee can round down to zero for small amounts and the `_burn` function would then still be called with a fee amount equal to zero. The client should note that calling burn with a zero value is not an issue, it simply uses unnecessary gas.
Recommendation	Consider checking if `fee > 0` instead of `burnTransferFee > 0`.
Resolution	✓ RESOLVED The contract will no longer call `_burn` if the fee is zero.

Issue #07	Typographical errors
Severity	● INFORMATIONAL

Description	We have consolidated the typographical errors into a single issue to keep the report brief and readable.
	
	Lines 8-10
	`* @title ContractName`
	`* @dev ContractDescription`
	`* @custom:dev-run-script ./deploy.js`
	
	These comments are outdated, they should be removed.
	
	Line 14
	`function initialize() initializer public {`
	
	Consider inverting `public` and `initializer` as is common practice.
	
	Line 17
	`burnTransferFee = 0;`
	
	Line 18
	`_mint(tx.origin, 888888888 * 1000000000000000000);`
	
	Consider rewriting the above as: `_mint(_msgSender(), 888_888_888 ethers);`
	
	We also prefer `_msgSender()` (or `msg.sender`) over `tx.origin`.
	
	Line 21
	`uint public burnTransferFee;`
	
	Consider sticking to `uint256` throughout the contract. Mixing `uint256` and uint has no effect but looks inconsistent.

Line 22

```
uint constant feeMuplipier = 100000000;
```

This line should be marked as `private` explicitly, the variable should be called `feeMultiplier` and the value should be explicitly set to `1e8` (though we prefer `1e4` as it represents basis points). Consider finally using UPPERCASE as is best practice for constants.

Line 24

```
function setBurnTransferFee(uint feePercent) public
onlyOwner {
```

Consider marking this function with `external` instead of `public`.

Recommendation	Consider fixing the typographical errors.
Resolution	● PARTIALLY RESOLVED Some of these errors have been resolved.

Issue #08	**Lack of events for `setBurnTransferFee`**
Severity	● INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the function.
Resolution	✓ RESOLVED

PALADIN

BLOCKCHAIN SECURITY