PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Preliminary Report

For Portal Fantasy

09 June 2023

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 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 Portal Fantasy 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	Portal Fantasy
URL	https://portalfantasy.io/
Platform	Avalanche
Language	Solidity
Preliminary Contracts	https://github.com/PortalFantasy/smart-contracts/tree/a29ac1c14277ff4803333ac8591e6a7c60d401f4/contracts
Resolution 1	

1.2 Contracts Assessed

Name	Contract	Live Code Match
PFT		
Item		
Porble		
PFTStakingUpgradeable		
TokenVaultUpgradeable		

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	4			
Medium	6			
Low	8			
Informational	5			
Total	**23**	**0**	-	-

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 Issue

ID	Severity	Summary	Status
01	INFO	Lack of unit tests	

1.3.2 PFT

ID	Severity	Summary	Status
02	HIGH	Governance: Controllers can mint and burn arbitrarily	
03	LOW	Missing visibility for controllers	
04	INFO	Unnecessary Proxied import	

1.3.3 Item

ID	Severity	Summary	Status
05	HIGH	Potential issues with regards to minting with Ether	
06	MEDIUM	Change in tokenToPay can result in undesirable side-effects	
07	MEDIUM	Lack of validation for vault	
08	MEDIUM	EIP2981 is not implemented	
09	MEDIUM	Valid signature can be reused	
10	LOW	Lack of safeTransfer usage	
11	LOW	Change of vault is not representative for royalty logic	
12	INFO	Unnecessary Proxied import	

1.3.4 Porble

ID	Severity	Summary	Status
13	HIGH	Sacrificed tokenIDs can be minted again	
14	MEDIUM	EIP2981 is not implemented	
15	INFO	Unnecessary Proxied import	

1.3.5 PFTStakingUpgradeable

ID	Severity	Summary	Status
16	LOW	`startTime` is potentially reset	
17	LOW	tokenToStake is private	
18	LOW	Checks-effects-interactions pattern is not adhered to	

1.3.6 TokenVaultUpgradeable

ID	Severity	Summary	Status
19	HIGH	Governance: Contract owner can withdraw all funds	
20	MEDIUM	transferFromVault will revert if amount is only partially covered	
21	LOW	Lack of safeTransfer usage	
22	LOW	Checks-effects-interactions pattern is not adhered to	
23	INFO	`__Pausable_init()` is never called	

2 Findings

2.1 Global Issues

The issues in this section apply to the protocol as a whole or to several contracts.

Please read through the issues carefully and apply fixes to all relevant contracts.

2.1.1 Issues & Recommendations

Issue #01	Lack of unit tests
Severity	INFORMATIONAL
Description	The provided repository lacks unit testing. This should be the standard before an audit is requested.
Recommendation	Consider implementing some basic testing.
Resolution	

2.2 PFT

PFT is a simple ERC20 token which implements LayerZero's cross chain functionality (https://github.com/LayerZero-Labs/LayerZero). The contract owner has the ability to add addresses as controllers, which can then mint and burn tokens.

The cross-chain functionality is out of scope for this audit, therefore we expect LayerZero's functionality to work as intended.

The token has no maximum supply and no initial supply is minted during the initialization.

2.2.1 Privileged Functions

- `mint`
- `burn`

2.2.2 Issues & Recommendations

Issue #02	Governance: Controllers can mint and burn arbitrarily
Severity	HIGH SEVERITY
Description	Any controller address can mint tokens to anyone and burn tokens from anyone. This could be risky if such an address is ever compromised.
Recommendation	Consider only using a multi-signature wallet with known or doxxed participants as controllers.
Resolution	

Issue #03	Missing visibility for controllers
Severity	LOW SEVERITY
Description	All controller addresses cannot be easily fetched — this could be valuable for the users to assess potential risks.
Recommendation	Consider implementing a clean enumerable set logic and add all controller addresses to it.
Resolution	

Issue #04	Unnecessary `Proxied` import
Severity	INFORMATIONAL
Description	The Proxied contract is a simple extension for proxy contracts that ensures that only the proxy admin can execute functions using the proxied modifier. This contract is incorrectly imported here since it needs to be inherited by the actual proxy contract itself and not the implementation.
Recommendation	Consider removing the contract.
Resolution	

2.3 Item

`Item` is an ONFT contract which is LayerZero's newly developed omnichain token standard (https://github.com/LayerZero-Labs/LayerZero). It is expected that all LayerZero implementations work as desired since these contracts are not part of the audit scope.

Users are able to mint specific IDs when a valid signature is used. The signature includes the IDs, their corresponding prices as well as the receiver, respectively the `msg.sender` and the type of payment (native or ERC20).

Minting can either be done with the native token or an ERC20 token as payment. The vault will receive the corresponding fee and the contract owner can change the payment token and vault at anytime.

2.3.1 Privileged Functions

- `setBaseURIString`
- `setMintSigner`
- `setIsPaymentWithNativeTokenEnabled`
- `setIsERC20PaymentEnabled`
- `setTokenToPay`
- `setVault`
- `setDefaultRoyalty`

2.3.2 Issues & Recommendations

Issue #05	Potential issues with regards to minting with Ether
Severity	HIGH SEVERITY
Description	There is no check that `msg.value = 0` when minting with ERC20 tokens. There is also no refund when the provided `msg.value` is larger than the required amount. This can result in a) stuck funds and b) in situations where other users might abuse the ether stuck in the contract for their own minting purposes.
Recommendation	Consider explicitly checking the `msg.value` and ensuring it is the same as required for this minting purpose.
Resolution	

Issue #06	Change in `tokenToPay` can result in undesirable side-effects
Severity	MEDIUM SEVERITY
Description	Users are able to mint tokens using an ERC20 token, if the signature is valid for this. However, the fact that the owner can change the ERC20 tokens can result in a few issues: 1. The owner can front-run a user minting and change the ERC20 token to USDC, for example. Since this token has only 6 decimals, the owner can effectively steal a user's USDC. This requires upfront approval. 2. The change in the ERC20 token can result in a loss for the contract since different tokens have different values and this can potentially be abused by users. 3. Changing the address to `address(0)` will result in free minting since the call does not revert.
Recommendation	Consider keeping this in mind when changing the ERC20 token, and consider implementing a time-locked mechanism for that and communicate each change with the community.
Resolution	

Severity	MEDIUM SEVERITY
Description	The `vault` variable can be changed by the owner and set to `address(0)`. This would effectively prevent any minting with ERC20 tokens as well as resulting in a loss of Ether.
Recommendation	Consider validating the `_vault` parameter.
Resolution	

Issue #08	**EIP2981 is not implemented**
Severity	MEDIUM SEVERITY
Description	The contract internally calls the following function: `_setDefaultRoyalty(vault, _feeNumerator);` However, the corresponding logic is not even imported.
Recommendation	Consider implementing the EIP2981 logic as well as executing proper tests.
Resolution	

Issue #09	**Valid signature can be reused**
Severity	MEDIUM SEVERITY
Description	It is possible to simply reuse a signature after the corresponding tokens have been burned. Depending on how these NFTs are used, this might present a critical issue.
Recommendation	Consider validating that the signature can only be used once.
Resolution	

Issue #10	Lack of `safeTransfer` usage
Severity	LOW SEVERITY
Description	Tokens are transferred using the normal `transfer` call, however, this will not work for malformed tokens or tokens that return `false` on transfers.
Recommendation	Consider implementing `safeTransfer`.
Resolution	

Issue #11	Change of vault is not representative for royalty logic
Severity	LOW SEVERITY
Description	It is desired that the royalty fee goes to the vault address. However, if the vault address is changed, this does not automatically change the royalty receiver.
Recommendation	Consider changing the royalty receiver as well when changing the vault.
Resolution	

Issue #12	Unnecessary `Proxied` import
Severity	INFORMATIONAL
Description	The `Proxied` contract is a simple extension for proxy contracts that ensures that only the proxy admin can execute functions using the proxied modifier. This contract is incorrectly imported here since it needs to be inherited by the actual proxy contract itself, not the implementation.
Recommendation	Consider removing the contract.
Resolution	

2.4 Porble

`Porble` is an ONFT contract which is LayerZero's newly developed omnichain token standard (https://github.com/LayerZero-Labs/LayerZero). It is expected that all LayerZero implementations work as desired since these contracts are not part of the audit scope.

Users are able to mint specific IDs when a valid signature is used. The signature includes the IDs, as well as the corresponding address. Unlike the PFT contract, minting is for free.

Additionally, users can fuse specific NFTs using a valid signature — the signature includes the NFTs to be burned as well as the corresponding address and a `fusionID`. Once a fusion is successful, the corresponding `fusionID` will be marked as `true` which might have further implications on other contracts.

2.4.1 Privileged Functions

- `setBaseURIString`
- `setMintSigner`
- `setDefaultRoyalty`

2.4.2 Issues & Recommendations

Issue #13	Sacrificed `tokenIDs` can be minted again
Severity	HIGH SEVERITY
Description	Whenever a fusion is executed, the sacrificed `tokenIds` are burned. However, the user can simply mint them again using the valid signature.
Recommendation	Consider validating that the user can only mint once using the same signature.
Resolution	

Issue #14	EIP2981 is not implemented
Severity	MEDIUM SEVERITY
Description	The contract internally calls the following function: `_setDefaultRoyalty(vault, _feeNumerator);` However, the corresponding logic is not even imported.
Recommendation	Consider implementing the EIP2981 logic as well as executing proper tests.
Resolution	

Issue #15	Unnecessary Proxied import
Severity	INFORMATIONAL
Description	The Proxied contract is a simple extension for proxy contracts that ensures that only the proxy admin can execute functions using the proxied modifier. This contract is incorrectly imported here since it needs to be inherited by the actual proxy contract itself and not the implementation.
Recommendation	Consider removing the contract.
Resolution	

2.5 PFTStakingUpgradeable

`PFTStakingUpgradeable` allows users to stake PFT tokens. During contract deployment, a `minimumStakeAmount` is determined which can be changed by the owner. During the first deposit, the initial amount needs to be higher or equal to the `minimumStakeAmount` in order to successfully deposit tokens. For every consecutive deposit, the sum of balance and amount to be deposited must be higher than `minimumStakeAmount`.

Users then have three withdraw options:

1. Withdraw all tokens
2. Withdraw part of the tokens
3. Withdraw all tokens except the `minimumStakeAmount`

Whenever a deposit happens while the current balance is below the minimum amount, the `startTime` for the current deposit is set. If, for example, a user withdraws all tokens or decreases the balance to be below the minimum amount, the next deposit will then reset the `startTime`.

Any withdrawal operation will not have an effect on the `startTime`, even if the deposited balance falls below `minimumStakeAmount`.

There is no incentive for users to stake their PFT tokens in this contract.

2.5.1 Privileged Functions

- `setMinimumStakeAmount`

2.5.2 Issues & Recommendations

Issue #16	`startTime` is potentially reset
Severity	LOW SEVERITY
Description	Whenever a user withdraws tokens which then reduces the balance below the `minimumStakeAmount`, a future deposit will reset the `startTime`. Depending on the use of this variable, this can become an issue.
Recommendation	Consider if this becomes an issue, and if so, consider switching to a different logic.
Resolution	

Issue #17	`tokenToStake` is private
Severity	LOW SEVERITY
Description	This variable is private, which could prevent users from easily retrieving that value.
Recommendation	Consider making this variable public.
Resolution	

Issue #18	Checks-effects-interactions pattern is not adhered to
Severity	LOW SEVERITY
Description	Within the `stake` function, `transferFrom` is executed before further effects. This is against best practices: https://medium.com/returnvalues/smart-contract-security-patterns-79e03b5a1659
Recommendation	Consider following the CEI pattern.
Resolution	

2.6 TokenVaultUpgradeable

`TokenVaultUpgradeable` is a simple token storage contract. Users can pay for different `opID`s using the native gas token or the PFT token. Once a payment has been made, the mapping for the corresponding opId will be updated with the deposited value.

The owner has the privilege to issue refunds to addresses that have paid for `opID`s in the same way they have deposited (native/ERC20).

Addresses with a valid signature can claim PFT tokens — the signature includes the amount to claim, the address and a corresponding nonce which is incremented with each claim.

All `opID`s have a different use case: *Allows an address to pay for a specific game operation (e.g. synthesizing porbles, purchasing cosmetics etc.)*

We assume that signatures will be created based on the paid operations.

2.6.1 Privileged Functions

- `issueFullRefund`
- `issueFullRefundNative`
- `setPFTVaultTransferSigner`
- `withdrawNativeTokens`
- `withdrawTokens`

2.6.2 Issues & Recommendations

Issue #19	Governance: Contract owner can withdraw all funds
Severity	HIGH SEVERITY
Description	Due to the fact that the contract owner can withdraw all funds, it might prevent users from - receiving a refund - transfer tokens out using a valid signature
Recommendation	Consider clearly communicating all withdrawals with the community.
Resolution	

Issue #20	`transferFromVault` will revert if amount is only partially covered
Severity	MEDIUM SEVERITY
Description	Within the `transferFromVault` functions, an amount of tokens is transferred from the vault to the `msg.sender`. However, if the vault has fewer tokens than the withdrawal amount, the withdrawal will completely revert, and the user will be unable to withdraw funds.
Recommendation	Consider implementing a pattern that sends out the rest of the balance in the contract instead of reverting.
Resolution	

Issue #21	Lack of `safeTransfer` usage
Severity	LOW SEVERITY
Description	Tokens are transferred using the normal `transfer` call, however, this will not work for malformed tokens or tokens that return `false` on transfers.
Recommendation	Consider implementing `safeTransfer`.
Resolution	

Issue #22	Checks-effects-interactions pattern is not adhered to
Severity	LOW SEVERITY

Description	In the following snippet, `transfer` is executed before the effect:
	`IERC20Upgradeable(PFTAddress).transferFrom(msg.sender, address(this), amount );`
	`userPaymentsInfo[msg.sender][opId] = amount;`
	`emit Payment(msg.sender, opId, int256(amount), false);`
	This is against best practices: https://medium.com/returnvalues/smart-contract-security-patterns-79e03b5a1659

Recommendation	Consider following the CEI pattern.
Resolution	

Issue #23	`__Pausable_init()` is never called
Severity	INFORMATIONAL
Description	The `initialize` function does not call `__Pausable_init()` from the Pausable upgradeable contract it inherits from. While pausable is initialized by default, it is nice to follow the best practices and call its `_init_` method.
Recommendation	Consider calling `__Pausable_init()` in the initialize function.
Resolution	


PALADIN
BLOCKCHAIN SECURITY