PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Arcadium LootBox

28 August 2021

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.

1 Overview

This report has been prepared for Arcadium LootBox 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	Arcadium LootBox
URL	http://darkside.finance/lootcrate
Platform	Polygon
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`LootCrate`	`0x5eD6da66f5E2256e177f3376d335Cd6cCC582804`	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	3	3	-	-
Medium	1	1	-	-
Low	4	3	-	1
Informational	9	9	-	-
Total	**17**	**16**	**-**	**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 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 LootCrate

ID	Severity	Summary	Status
01	HIGH	An exploiter can reject cards to carefully select the best cards possible at the cost of a single purchase by making the `fulfillRandomness` request fail	RESOLVED
02	HIGH	An exploiter can reenter into `buyThreeRandomCards` through the NFT receipt hook to give themselves a more favorable card allocation	RESOLVED
03	HIGH	Contract can sell more loot boxes then there are available	RESOLVED
04	MEDIUM	`getLastXUserPurchases` caches all user purchases to memory, causing it to eventually run out of gas if the user has too many purchases	RESOLVED
05	LOW	`setPriceInMatic` causes the price to smoothly decline downwards if the price point is lower, but instantly upwards if it is higher	RESOLVED
06	LOW	Chainlink, miners and potentially advanced frontrunners could reorder transactions to their advantage	ACKNOWLEDGED
07	LOW	The LootBox does not work with tokens with transfer tax or NFTs that can perish	RESOLVED
08	LOW	The contract will stop working after 02/07/2106 due to the Uniswap trade timestamp overflowing and could stop working earlier if the accumulated prices in the pair overflow	RESOLVED
09	INFO	`random2idx` reverts when the range is zero	RESOLVED
10	INFO	`addCard`, `withdrawCardByIdOwner` and `withdrawThreeCards` should be made `nonReentrant`	RESOLVED
11	INFO	`getLastXUserPurchases` and `getLastXPurchases` can be simplified	RESOLVED
12	INFO	Price manipulation during contract deployment could lead to a severely excessive price point	RESOLVED
13	INFO	The code organization has a few functions above the constructor	RESOLVED
14	INFO	Gas optimization: `onERC721Received` unnecessarily recalculates a function signature every run	RESOLVED
15	INFO	`LootCratePurchased` event does not contain the amount of spins purchased	RESOLVED
16	INFO	The `startBlock`, `arcadium`, `arcadiumLiqPool`, `cumulativePrice`, `priceLastUpdated`, `smoothedCardPrice`, `smoothingPercent`, `pendingPurchases` and `userPurchasesMap` variables are declared privately	RESOLVED
17	INFO	The `random` function can be made `pure`	RESOLVED

2 Findings

2.1 LootCrate

2.1.1 Privileged Roles

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

- `withdrawAccumulatedFunds`
- `setSmoothingPercent`
- `addCard`
- `setPriceInMatic`
- `withdrawCardByIdOwner`

2.1.2 Issues & Recommendations

Issue #01	**An exploiter can reject cards to carefully select the best cards possible at the cost of a single purchase by making the `fulfillRandomness` request fail**
Severity	HIGH SEVERITY
Description	One of the important rules from the VRF Security Considerations is: *If your contract could have multiple VRF requests in flight simultaneously, you must ensure that the order in which the VRF fulfillments arrive cannot be used to manipulate your contract's user-significant behavior.* This rule is broken within Arcadium Lootbox primarily due to the fact that after a request is made, another request might be fulfilled, or a new card might be added which affects the outcome of a fulfillment. In the first instance, this can only be exploited by ChainLink or the miners by carefully re-ordering transactions to their advantage. This case might not be realistic on Polygon. However, through more careful inspection, this failed requirement allows for more advanced exploits. For this we quote a second part of the VRF Security Considerations: *If your implementation does revert, you can independently re-send the fullfilment transaction to ensure that all requests issued by your contract are fulfilled, but the VRF service won't do so on your behalf.)* A ChainLink randomness transaction works as follows: First a commitment is made in a first transaction containing the financial cost. Later on, ChainLink comes back and calls `fulfillRandomness` to provide a random number to this request. As stated in the quote above, in case `fulfillRandomness` does revert, ChainLink will not try again.

But the exploit vector is that at this point, the user is at complete liberty to call `fulfillRandomness` whenever they want. The trick of a malicious exploiter is then to somehow make the initial `fulfillRandomness` request fail, so that they can call this function again at the optimal moment for them.

To understand how an optimal moment can be picked through simply delaying calling this function, we shall take a look at how the received card can change when a card is added or removed from the stack, which happens naturally over time as others fulfill rewards or governance adds new cards.

Consider the following example where the random number from ChainLink is 6190781072625605 and there are 300 cards in the stack.

The index of the card that the user would receive is then 6190781072625605 % 300 or index 5. However, when there is one more or one less card in the stack, the following scenario applies to this index:

Stack size	Card index
299	157
300	5
301	183

The card received is a completely different pseudo-random number. Thus, if the exploiter can make their initial `fulfillRandomness` request fail, they can simply wait until they get the best cards available and manually call `fulfillRandomness` through the ChainLink coordinator.

The final step which remains is to make the `fulfillRandomness` request fail: this can be made more likely by setting spins to 10, which increases the gas cost significantly (ChainLink only guarantees 200k gas for their fulfillment function).

More detrimentally, the code uses `safeTransferFrom` to transfer an NFT card to the user. This function actually calls a receipt hook on the user if that user is a contract (this is done to ask the contract if it can actually receive NFTs). A malicious party will make this hook return `false` during the first request to ensure this first request always fails. Afterwards they can go ahead and wait until they get their optimal allocation.

Any state where `withdrawThreeCards` could fail results in this setting where an exploiter can then wait for their optimal allocation.

Recommendation

In the short-term, consider the following defense mechanisms:

1. Only allowing EOAs to call `buyThreeRandomCards` to reduce the risk of hooks being called. As this is a game, we are unsure if contracts will participate unless they are trying to exploit it.
2. Using `transferFrom` instead of `safeTransferFrom` on the NFT so as to not call the hook.
3. Ensuring `fulfillRandomness` never goes out of gas nor makes any external calls by making it simply allocate a card to the user which is later claimed by them. Maximum gas usage should be 200k. By making external calls it automatically becomes unbound since you don't know how much logic will be executed when calling a transfer on an external token.

In the long-term, consider fundamentally redesigning the protocol to account for the ChainLink requirement that fulfillment order should not affect outcome. One way of doing this is through setting probabilities of the cards. Another way to do this is through requiring all cards to be requested before they are all fulfilled in a single request. We recommend the client to carefully go through the `fulfillRandomness` function to ensure all possible ways for a malicious party to make it revert are addressed.

Resolution

The client has incorporated all short term recommendations, and furthermore, the client has moved to a new design pattern where the risk of reversion is absolutely minimized. The core property is not satisfied but exploiting it without privileges looks unfeasible to us due to these changes.

Issue #02	**An exploiter can reenter into buyThreeRandomCards through the NFT receipt hook to give themselves a more favorable card allocation**
Severity	HIGH SEVERITY
Location	Line 304 `IERC721(card.contractAddress).safeTransferFrom(address(this), recipient, card.amountOrID);`

Description	Currently, the `withdrawThreeCards` function does not contain a `reentrancyGuard`, presumably because the developer assumed that as this can only be called by ChainLink, the function is less vulnerable to reentrancy.
	
	As discussed in the previous issue, adding one single card to the stack is enough to completely change the outcome for the user and can thus be abused to receive a better hand.
	`_____________________________`
	`| Stack size | Card index |`
	`|------------+------------|`
	`| 299 | 157 |`
	`|------------+------------|`
	`| 300 | 5 |`
	`|------------+------------|`
	`| 301 | 183 |`
	`_____________________________`

Recommendation	Consider the following defensive changes: 1. Use `transferFrom` instead of `safeTransferFrom` for the NFT transfer so as to not call a hook on receipt. 2. Only allow EOAs to call `buyThreeRandomCards` so that the recipient is never a contract. 3. Add a reentrancy guard to `withdrawThreeCards` to prevent reentrancy into `buyThreeRandomCards`.
Comments	RESOLVED There are no more external calls in `fulfillRandomness`.

Issue #03	Contract can sell more loot boxes then there are available
Severity	HIGH SEVERITY

Location	Lines 200-202
	`require(cards.length + rareCards.length >= pendingPurchases * 3 + 3, "Sold out!");`
	
	`uint256 remainingSpins = (cards.length + rareCards.length - pendingPurchases * 3) / 3;`

Description	By community request, the client has moved from a single spin system where a user can open one box at a time to a multi-spin system where a user can open up to ten boxes at a time. However, while doing this, the client did not update the variables that calculate how many spins are still unallocated in the contract. This means that if someone claimed 10 spins (30 cards), the contract still thinks only 3 cards have been allocated (1 spin). Because of this misaccounting, an excessive number of purchases are allowed to occur which would result in there being too few cards.
Recommendation	Consider renaming the `pendingPurchases` variable to `pendingSpins` to account for the amount of spins that are pending. All occurrences of the `pendingPurchases` variable should then be updated to add and subtract the amount of spins bought or fulfilled.
Resolution	RESOLVED The `spins` feature has been disabled to ensure the gas limit stays under 200k.

Issue #04	**`getLastXUserPurchases` caches all user purchases to memory, causing it to eventually run out of gas if the user has too many purchases**
Severity	MEDIUM SEVERITY
Location	Line 100 `Purchase[] memory userPurchases = userPurchasesMap[customerAddress];`
Description	The utility function `getLastXUserPurchases` loads all user purchases to memory before selecting a more limited length from them. This initial loading can and will cause this function to run out of gas if the user has too many purchases, regardless of the value of the length parameter.
Recommendation	Consider just accessing the purchases directly from storage instead of caching them to memory. `Purchase[] storage userPurchases = userPurchasesMap[customerAddress];`
Resolution	RESOLVED

Issue #05	**`setPriceInMatic` causes the price to smoothly decline downwards if the price point is lower, but instantly upwards if it is higher**
Severity	LOW SEVERITY

Location	Lines 413–417
	`function setPriceInMatic(uint256 p) external onlyOwner {`
	`  priceInMatic = p;`
	
	`  emit MaticPriceSet(priceInMatic);`
	`}`

Description	The `setPriceInMatic` function does not update the `smoothedCardPrice`. As a result of this, the actual price will slowly adjust downwards if the price point is lowered while it will instantly move upwards if it is increased. Although this is hardly a problem, the behavior might not be desired especially when smoothing is slow.
Recommendation	Consider whether this behavior is desired. If not, it might be better to not multiply the `smoothedCardPrice` by the `priceInMatic`. Instead, the raw price per Arcadium could be stored. This issue will be resolved in either case since this behavior could be desired.
Resolution	RESOLVED The client has explained that this is desired behavior.

Issue #06	**Chainlink, miners and potentially advanced frontrunners could reorder transactions to their advantage**
Severity	LOW SEVERITY
Description	As discussed in previous issues, the size of the card stack heavily changes the outcome of a draw. This in theory allows chainlink, miners and advanced frontrunners (in the sense that they can frontrun chainlink to re-order their transaction) to re-order the pending transactions in a way that gives them a favorable outcome.
Recommendation	Consider fundamentally rethinking the business logic as to not have transaction order affect the outcome of the draw. This is not an easy task of course.
Resolution	ACKNOWLEDGED

Issue #07	**The LootBox does not work with tokens with transfer tax or NFTs that can perish**
Severity	LOW SEVERITY
Description	The LootBox does not function properly for tokens with a transfer-tax or NFTs that can expire. This is because it only records the amount send to the LootBox and not the amount of tokens that arrived or the actual balance of the NFT.
Recommendation	Consider whether such tokens or NFTs will ever be added. To handle tokens with transfer taxes, a before-after pattern in the `addCard` function will suffice (remember to always combine this pattern with a reentrancy guard).
Resolution	RESOLVED Tokens with a transfer tax are now supported and the client will carefully inspect the NFTs they add.

Issue #08	**The contract will stop working after 02/07/2106 due to the Uniswap trade timestamp overflowing and could stop working earlier if the accumulated prices in the pair overflow**
Severity	LOW SEVERITY
Location	Line 323 `uint256 timeDelta = pairLastUpdated - priceLastUpdated;`
Description	Each trade in Uniswap updates a variable that keeps track of the latest trade timestamp, but as stated in the Uniswap whitepaper, this timestamp resets every so often: *The primary downside is that 32 bits isn't quite enough to store timestamp values that will reasonably never overflow. In fact, the date when the Unix timestamp overflows a uint32 is 02/07/2106.* Since a `timeDelta` is done with v0.8.0 between two of these timestamps, once the year 2106 passes, this subtraction will thus revert.
Recommendation	Consider whether the contract will survive until 2106, and if so, consider wrapping the subtraction of the two timestamps in an unchecked code block so as to allow for the overflow to occur. This overflow should not impact the difference between the two timestamps. This issue will also be resolved in case the client thinks it is unlikely that the contract will be still in use by this time, which is quite reasonable. Furthermore, they could simply swap out the contracts once this time approaches. In general, the Uniswap oracle feature is supposed to work well with unsafe math. The accumulated prices could actually overflow as well which might be an argument to not simply acknowledge this issue if that is seen as possible within a more reasonable timespan.
Resolution	RESOLVED The timestamp can now safely overflow. All though the accumulated sum cannot, the client has explained that this is very unlikely due to the magnitude of the variable. In case this sum does overflow, they can simply move to new contracts.

Issue #09	**random2idx reverts when the range is zero**
Severity	INFORMATIONAL

Location	Lines 281-283
	`uint256 idx = random2Idx(randomness, cards.length + (! hasHadRare || (hasHadRare && cards.length == 0) ? rareCards.length : 0));`
	
	`if (cards.length + rareCards.length > 0) {`

Description	When no more cards remain, the range in the `random2Idx` becomes zero and the function reverts. However, this call is then followed by a check that checks that a subset of this range is greater than zero which will then of course always pass. We do not mind defining logic twice if it results in more clarity and certainty, but we decided to point out this behavior since the client might not have known that `random2Idx` can revert, and they might believe that this state could occur. If this state could occur, it would be another way for an exploiter to reach the exploit state in issue 1.
Recommendation	Consider explicitly defining this requirement and carefully considering if and why it might occur.
Resolution	RESOLVED The client has removed this function. In any case, the client explained that this state should not occur.

Issue #10	**`addCard`, `withdrawCardByIdOwner` and `withdrawThreeCards` should be made `nonReentrant`**
Severity	INFORMATIONAL
Description	These functions should be made non-reentrant with a similar argument as was given in the first issue. Adjusting the card stack count on the flight might allow an exploiter to gain a favorable hand. Furthermore, the `id` counter in `addCard` is not secure to reentrancy since the count is only incremented at the end of this function. This issue is marked as informational since the `withdrawThreeCards` recommendation has already been included in Issue 1 and the other two functions are privileged to the owner.
Recommendation	Consider adding reentrancy guards to these functions.
Resolution	RESOLVED

Issue #11	**getLastXUserPurchases and getLastXPurchases can be simplified**
Severity	INFORMATIONAL

Description

The two utility functions `getLastXUserPurchases` and `getLastXPurchases` seem to contain unnecessary complexity.

Here is an overview of the logic we believe could be simplified. Finally we believe that it might not be desired for these functions to revert if there are no purchases yet.

Original	Simplified
`uint256 zeroOrXDown = userPurchases.length > length ? pos - (length - 1) : 0;`	`uint256 zeroOrXDown = pos - (length - 1);`
`Purchase[] memory fechedUserPurchases = new Purchase[](userPurchases.length - zeroOrXDown);`	`Purchase[] memory fechedUserPurchases = new Purchase[](length);`
`require(userPurchases.length > 0, "no purchases yet!");`	`if (userPurchases.length == 0) { return userPurchases; }`

Recommendation

Consider whether the complexity is warranted, if not, consider whether the proposed simplifications do not contain side effects and if not, consider implementing them.

Consider whether these functions should really return if there are no purchases. It would seem more reasonable to simply return an empty array in this case.

Resolution

RESOLVED

Issue #12	**Price manipulation during contract deployment could lead to a severely excessive price point**
Severity	INFORMATIONAL
Location	Line 189 `smoothedCardPrice = _arcadium == IUniswapV2Pair(arcadiumWmaticPair).token0() ? priceInMatic * reserve0 / reserve1 : priceInMatic * reserve1 / reserve0;`
Description	During contract deployment, the current price is fetched to instantiate the price smoothing functionality. If a malicious party strategically reorders this deployment between a sandwich price manipulation transaction, they could set this initial `smoothedCardPrice` to an arbitrarily high value. The result of this is that the price might smooth down for a very long time before hitting the actual price. It should be noted that this issue is highly theoretical and we would be surprised to see it manifest during deployment. Even if it manifested, the client can simply redeploy.
Recommendation	Consider checking the initial smoothed price after deployment just to make sure that this has not happened. If it is incorrect, consider adjusting the code to allow for the deployer to inject the smoothed price as an argument.
Resolution	RESOLVED The client will double check the price after deployment.

Issue #13	The code organization has a few functions above the constructor
Severity	INFORMATIONAL
Location	Line 166 `constructor(uint256 _startBlock, address _arcadium, address arcadiumWmaticPair)`
Description	The constructor of the contract is not placed at the top of code which is considered bad practice. In general, the constructor should be organized after the event and variable declaration.
Recommendation	Consider moving the constructor upwards.
Resolution	RESOLVED

Issue #14	**Gas optimization: onERC721Received unnecessarily recalculates a function signature every run**
Severity	INFORMATIONAL

Location	Line 18-25
	`function onERC721Received(`
	`...`
	`) external override returns(bytes4) {`
	`return`
	`bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"));`
	`}`

Description	In the current implementation of the `onERC721Received` hook, which is just a function indicating to NFT contracts that this contract can in fact receive NFT tokens, the response signature is recalculated every time. This is slightly more expensive and encoding a signature manually is prone to error.
Recommendation	Consider obtaining the signature directly through the interface selector `IERC721.onERC721Received.selector` as recommended in the OpenZeppelin documentation on this hook.
Resolution	RESOLVED

Issue #15	**`LootCratePurchased` event does not contain the amount of spins purchased**
Severity	INFORMATIONAL
Location	Line 72 `event LootCratePurchased(address buyerAddress, uint256 maticPrice, uint256 arcadiumPrice);`
Description	The `LootCratePurchased` event does not contain the amount of spins purchased – it might be desired to keep track of this variable through the event log.
Recommendation	Consider adding `spins` as a variable to the `LootCratePurchased` event.
Resolution	RESOLVED Spins has been removed from the business logic.

Issue #16	**The `startBlock`, `arcadium`, `arcadiumLiqPool`, `cumulativePrice`, `priceLastUpdated`, `smoothedCardPrice`, `smoothingPercent`, `pendingPurchases` and `userPurchasesMap` variables are declared privately**
Severity	INFORMATIONAL
Description	Important variables that are marked as private make it more difficult for third-party auditors to verify that these are not set to malicious contracts.
Recommendation	Consider making these variables public to ease the process of third-party verification.
Resolution	RESOLVED

Issue #17	The `random` function can be made `pure`
Severity	INFORMATIONAL
Description	Functions that do not rely on or affect any state can be indicated as such using the `pure` parameter. This allows for the compiler to further optimize said functions.
Recommendation	Consider changing the type of the functions mentioned above from `view` to `pure`.
Resolution	RESOLVED The function has been removed since the `spins` logic has been removed.


PALADIN
BLOCKCHAIN SECURITY