PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Slot Machine

08 October 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 Slot Machine on the Binance 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	Slot Machine
URL	https://darkside.finance/slots/
Platform	Binance Smart Chain
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`SlotMachine`	`Plush:` `0xD6dc648E811Bb2709740C850E03662616803D9Ef` `BUSD:` `0x043876Ea113db00C707A2e4029d9428FAED6e084`	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	3	3	-	-
Medium	2	2	-	-
Low	1	1	-	-
Informational	9	9	-	-
Total	**15**	**15**	**-**	**-**

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 SlotMachine

ID	Severity	Summary	Status
01	HIGH	`pendingPrecalcPayouts` is not correctly decremented causing the owner to be unable to withdraw all income	RESOLVED
02	HIGH	DoS allows attacker to burn all Link in contract since there is no minimum wager amount	RESOLVED
03	HIGH	`rand2pos` uses the same random variable or two of the three slot columns	RESOLVED
04	MEDIUM	Lack of validation on `playGame` inputs	RESOLVED
05	MEDIUM	maxPayoutPerGameWager does not indicate the true maximum payout for the TWO and THREE game types causing there to potentially be insufficient funds to fulfill these games	RESOLVED
06	LOW	Gov privilege: Owner could receive a favorable payout by frontrunning ChainLink	RESOLVED
07	INFO	Lack of validation to important governance functions could prevent wagers being made or fulfilled if variables are set wrongly	RESOLVED
08	INFO	Gas optimization: Unnecessary position increment if only one game is played	RESOLVED
09	INFO	currency can be made public	RESOLVED
10	INFO	Lack of events for gameFulfillment, withdrawal and setPayout	RESOLVED
11	INFO	Typographical error in event PlayerFundaDrained	RESOLVED
12	INFO	viewPlayerPayout does not show the game payout after withdrawGamePayout has been called	RESOLVED
13	INFO	requestRandomness error message still refers to testnet	RESOLVED
14	INFO	Spinning with two or three rows at once does not truly respin the wheel; instead, it simply takes the following positions in the permutated wheel indices	RESOLVED
15	INFO	drainFunds cannot drain exactly all unallocated funds	RESOLVED

2 Findings

2.1 SlotMachine

The Slot Machine is a provably fair slot machine contract. It allows the player to wager an amount of tokens and their payout is determined based on the three random slots that are generated during their game. These random slots are provably randomly generated through ChainLink VRF. Furthermore each slot has an equal chance of presenting itself as there are no weighted fields in the contract (except if the frontend presents two different slots as the same one). Finally, the player also has the option to play two or three times.

The contract owner can freely set the payout of each individual combination of slots. Any profit that the owner might receive since the expected payoff is likely lower than the wager can be withdrawn by the owner.

2.1.1 Privileged Roles

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

- `setPayout`
- `setMaxPayoutPercent`
- `setWagerReserveMultiple`
- `drainFunds`
- `setMinWager`
- `updateOperator (only callable once)`
- `transferOwnership`
- `renounceOwnership`

2.1.2 Issues & Recommendations

Issue #01	**pendingPrecalcPayouts is not correctly decremented causing the owner to be unable to withdraw all income**
Severity	HIGH SEVERITY

Location	Line 144
	`pendingPrecalcPayouts += maxPayout;`
	
	Line 180
	`pendingPrecalcPayouts -= payout;`
	
	Lines 310-311
	`function drainFunds(address recipient, uint256 amount) external onlyOwner {`
	`require(currency.balanceOf(address(this)) > amount + pendingPrecalcPayouts + pendingCalculatedPayouts);`

Description	A game is executed in two transactions. First, `playGame` is called to start the game. Then in a later transaction, ChainLink calls the `fulfillRandomness` function to provide the game with a genuine and secure randomness value to determine the outcome with. As the admins do not want to accidentally take tokens out of the contract which potentially are still required for the games payout, they keep track of variables called `pendingPrecalcPayouts` and `pendingCalculatedPayouts`: - `pendingPrecalcPayouts`: The total maximum payout possible for any game in progress that still needs to be fulfilled by ChainLink. - `pendingCalculatedPayouts`: The total amount of payouts that still need to be withdrawn. However, when a game is fulfilled by ChainLink, the actual payout is decremented instead of decrementing the maximum possible payout again. This causes `pendingPrecalcPayouts` to increase naturally over time with `maxPayout - actualPayout` for every game. Since the admin cannot withdraw any tokens that are allocated as `pendingPrecalcPayouts` (line 311), this will eventually make them unable to withdraw any fund income.

Recommendation Consider adding a `maxPayout` field to the PlayerGame struct:

Lines 43-50

```
struct PlayerGame {
    uint256 wager;
    uint256 payout;
    uint256 maxPayout;

    uint256 randn;

    Games game;
}
```

Then set the `maxPayout` during `playGame`:

Line 142

```
playerGames[requestId] = PlayerGame({ game: game, payout: 0, maxPayout: maxPayout, wager: wager, randn: 0 });
```

And finally decrement this `maxPayout` instead of payout within the fulfillment:

Lines 179-183

```
if (pendingPrecalcPayouts > maxPayout) {
    pendingPrecalcPayouts -= maxPayout;
} else {
    pendingPrecalcPayouts = 0;
}
```

❗ This solution remains insufficient because `setPayout` and `setMaxPayoutPercent` still adjust `pendingPrecalcPayouts`. Therefore, these two functions should only be callable when there are no more games in progress. This final requirement could be added by keeping track of a `gamesInProgress` counter.

Resolution RESOLVED

The recommendation has been implemented including the recommended safeguard in `setPayout` and `setMaxPayoutPercent`.

Issue #02	**DoS allows attacker to burn all Link in contract since there is no minimum wager amount**
Severity	HIGH SEVERITY
Description	Because the contract uses ChainLink, each play burns a bit of Link tokens which the owner has to provide to the contract. Since there is currently no minimum on the amount that needs to be wagered, an attacker could keep playing games with an infinitesimally small wager to drain the contract of Link. Although this is not directly profitable to the attacker, the attacker might be motivated by the fact that they can break the game through this attack.
Recommendation	Consider adding a minimum wager amount that in expected return to the house terms covers the Link fee. `require(wager >= minimumWager, "Wager too small");` ❗ Note that this requirement is best called after the previous game payout is incorporated to allow people to roll-over without any transfer.
Resolution	RESOLVED A `minimumWager` variable as was recommended has been introduced. Initially this is set to 1 token.

Issue #03	**rand2pos uses the same random variable or two of the three slot columns**
Severity	HIGH SEVERITY

Location	Lines 247-249
	`function rand2pos(uint256 randomness) internal view returns (uint256, uint256, uint256) {`
	`    return (randomness % wheel1.length, expand(randomness) % wheel2.length, expand(randomness) % wheel3.length);`
	`}`

Description	The `rand2pos` function takes a random number and outputs 3 pseudo-randomly slot positions from it. It does so by using the randomness directly for the first position and then hashing the randomness to receive a new value for the other positions. However, the hash is not re-hashed for third position which causes the two final slots to always end up in the same positions if the wheel lengths are the same.
Recommendation	Consider rehashing the hashed randomness for the third slot to receive yet another new value. `return (randomness % wheel1.length, expand(randomness) % wheel2.length, expand(expand(randomness)) % wheel3.length);`
Resolution	RESOLVED The third wheel is now expanded twice as was recommended.

Issue #04	Lack of validation on `playGame` inputs
Severity	MEDIUM SEVERITY
Description	The `playGame` function can be called with the game type set to finished and the wager set to zero. Since both these values have special meaning within the contract this state might have side-effects and should be avoided to reduce the state-space.

Recommendation	Consider adding requirements to `playGame` to ensure these values are never used as parameters.
	`require(game != Games.FINISHED, "Finished");`
	`require(wager >= minimumWager, "Wager too small");`
	❗ Note that this last requirement is best called after the previous game payout is incorporated to allow people to roll-over without any transfer.

Resolution	RESOLVED The recommended validations have been included.

Issue #05	**`maxPayoutPerGameWager` does not indicate the true maximum payout for the TWO and THREE game types causing there to potentially be insufficient funds to fulfill these games**
Severity	MEDIUM SEVERITY

Location	Lines 211-219
	`function maxPayoutPerGameWager(Games game, uint256 wager) public view returns (uint256) {`
	`  if (game == Games.ONE) {`
	`  return wager * maxPayoutPercent / 100;`
	`  } else if (game == Games.TWO) {`
	`  return wager * maxPayoutPercent / 200;`
	`  } else if (game == Games.THREE) {`
	`  return wager * maxPayoutPercent / 300;`
	`  }`
	`}`

Description	There are three types of games: ONE, TWO and THREE. Each indicates how many times the game will be played and the wager is split amongst all plays equally. In theory, the maximum payout of TWO and THREE plays thus remains approximately the maximum payout of a single play, since you can nearly hit the maximum payout two or three times causing the average payout to become the maximum payout again. Thus the `maxPayoutPerGameWager` function is incorrect in dividing the maximum payout by 2 and 3 for the latter two game types.
Recommendation	Consider simply always returning the true upper limit for the payout. `return wager * maxPayoutPercent / 100;`
Resolution	RESOLVED The recommended simplification has been implemented.

Issue #06	Gov privilege: Owner could receive a favorable payout by frontrunning ChainLink
Severity	LOW SEVERITY
Location	Line 283 `function setPayout(uint256 label1, uint256 label2, uint256 label3, uint256 payout) external onlyOwner {`
Description	The `setPayout` function allows the governance to adjust the payouts of slot combinations. In theory, they could frontrun the ChainLink fulfillment to adjust their payout in a favorable fashion if they inspect the ChainLink transaction to figure out the slots they would have. Additionally, the `setPayout` function can be called when the owner notices large, profitable bettors playing the game, and thus adjust the payouts to low amounts to deny these users their expected windfall (especially given that front-running of randomness and thus outcome is very possible given the congested block times on Polygon). This issue is marked as low risk since the rewards for the owner would likely not be worth it since there is already a `drainFunds` function to drain any rewards that have not been allocated to users that have either not withdrawn yet or are still playing. The only funds that could thus be stolen by a malicious governance are those funds which are not withdrawn/played yet.
Recommendation	Consider implementing Pausable and making `playGame` `whenNotPaused`. The `setPayout` function can then have a requirement that there should be no more games in progress: `require(gamesInProgress == 0);` Note that a `gamesInProgress` counter will need to be added that is incremented during `playGame` and decremented during `fulfillRandomness`. ℹ️ This solution is somewhat desirable because the fact that payouts can be adjusted while games are in progress is the only thing that still breaks the ChainLink requirement that outcomes should not be externally manipulatable.
Resolution	RESOLVED The recommendation has been added and setPayout can only be changed while no more games are in progress.

Issue #07	**Lack of validation to important governance functions could prevent wagers being made or fulfilled if variables are set wrongly**
Severity	INFORMATIONAL
Description	**Constructor** There is lack of validation that the `payoutLabels` cover all the wheel combinations. **`setMaxPayoutPercent` and `setWagerReserveMultiple`** The payout and ratio variables in these functions are unbounded. These could force people into playing games while there are insufficient tokens in the contract to handle the payout.
Recommendation	Consider validating and adding the following requirements: **Constructor** `require(payoutLabels.length == wheel1.length * wheel2.length * wheel3.length);` Although this requirement is still insufficient to guarantee that each combination is covered, it makes the room for error smaller. **`setMaxPayoutPercent` and `setWagerReserveMultiple`** It is more difficult for us to recommend appropriate values for these functions. The reason for this is that there is currently no way for the `maxPayoutPercent` to decrease if the maximum payout goes down. It might thus be easiest to acknowledge this shortcoming and put these functions behind a timelock for public inspection.
Resolution	RESOLVED The constructor validation has been added and `setMaxPayoutPercentage` can only be called when there are no games in progress.

Issue #08	Gas optimization: Unnecessary position increment if only one game is played
Severity	INFORMATIONAL
Location	Line 262 `return wager * pos2payout(pos1 + 1, pos2 + 1, pos3 + 1) / 100;`
Description	The `game2payout` function increments the positions for every spin. However, when only a single spin is played, these positions are still incremented for some reason.
Recommendation	Consider explaining to us why this increment might be desirable in this case. If it is not necessary, consider removing it in light of readability and efficiency.
Resolution	RESOLVED The client has indicated that this is desirable behavior for their frontend.

Issue #09	`currency` can be made `public`
Severity	INFORMATIONAL
Description	Variables that define important parts of the contract state should be marked as `public` for easy inspection by third parties.
Recommendation	Consider making the above variable explicitly `public`.
Resolution	RESOLVED

Issue #10	Lack of events for `gameFulfillment`, `withdrawal` and `setPayout`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions. ❗ In addition, the `requestId` is not indexed or returned during `playGame` which might make figuring it out on the frontend more difficult. The client should consider indexing both the `gameId` for the `GameStarted` event and `GameFulfilled` event so the frontend can easily discover when a pending game has been fulfilled. Indexing the player themselves is likely enough as well since there will only be one game at a time for a player anyways.
Resolution	RESOLVED

Issue #11	Typographical error in event `PlayerFundaDrained`
Severity	INFORMATIONAL
Location	Line 19 `event PlayerFundaDrained(address recipient, uint256 amount);`
Description	The `PlayerFundsDrained` event contains a typographical error.
Recommendation	Consider fixing the typographical error.
Resolution	RESOLVED

Issue #12	**`viewPlayerPayout` does not show the game payout after `withdrawGamePayout` has been called**
Severity	INFORMATIONAL
Location	Line 229 `function viewPlayerPayout(address player) external view returns (uint256) {`
Description	After the payout is withdrawn to the user, the payout is set to zero. This causes `viewPlayerPayout` to return zero for the last game which might be confusing for the user as they did earn a payout on the last game.
Recommendation	Consider whether this is desired behavior.
Resolution	RESOLVED The client has indicated this as desired behavior.

Issue #13	**`requestRandomness` error message still refers to testnet**
Severity	INFORMATIONAL
Location	Line 161 `require(LINK.balanceOf(address(this)) >= fee, "Not enough LINK - fill contract with faucet");`
Description	When there are insufficient Link tokens in the contract to pay the ChainLink fee, the error thrown will say that the contract needs to be filled up again through a Link faucet. We assume this error is only relevant on testnet.
Recommendation	Consider rewriting the error message to explain that the contract needs to be topped-up with Link tokens manually.
Resolution	RESOLVED

Issue #14	**Spinning with two or three rows at once does not truly respin the wheel; instead, it simply takes the following positions in the permutated wheel indices**
Severity	INFORMATIONAL
Description	The game allows for a user to spin two or three rows at once. However, the wheel positions are incremented for the second and third wheel which causes correlated outcomes. This issue is marked as informational as we suspect this may have been an optimization to stay within the ChainLink gas limit requirement.
Recommendation	Consider further expanding the randomness to get completely new positions for the second and third spin. ❗ There should be no overlapping expansions as this might cause two spins or rows to be correlated with each other.
Resolution	RESOLVED The client has indicated that this correlation is desired.

Issue #15	**drainFunds cannot drain exactly all unallocated funds**
Severity	INFORMATIONAL
Location	Line 310 `require(currency.balanceOf(address(this)) > amount + pendingPrecalcPayouts + pendingCalculatedPayouts);`
Description	The `drainFunds` function always requires the balance to be larger than the amount withdrawn plus the allocated funds. This means that after a withdrawal, there will always need to be some unallocated funds remaining.
Recommendation	Consider making the equality greater or equal to: `require(currency.balanceOf(address(this)) >= amount + pendingPrecalcPayouts + pendingCalculatedPayouts);`
Resolution	RESOLVED


PALADIN
BLOCKCHAIN SECURITY