Smart Contract Security Assessment

Final Report

For ChainBet

14 August 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 ChainBet 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	ChainBet
URL	https://chainbet.gg/
Network	Polygon
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`Bankroll`	`0xB0060D8439269d45A1A68426c07f9C2C5243de63`	MATCH
`BankrollShare`	`Deployed by Bankroll contract`	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	1	0	1	-
Medium	1	1	-	-
Low	3	2	-	1
Informational	14	13	1	-
Total	**19**	**16**	**2**	**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 Bankroll

ID	Severity	Summary	Status
01	HIGH	Governance privilege: Governance can whitelist new games that are allowed to withdraw all pool balances	PARTIAL
02	MEDIUM	`emergencyWithdraw` lacks debt validation	RESOLVED
03	LOW	Protocol is vulnerable to JIT quitting of liquidity	ACKNOWLEDGED
04	LOW	Return values of `burnFrom` are not handled	RESOLVED
05	LOW	`maxBetChanged` emits the wrong parameters	RESOLVED
06	INFO	Unnecessary indexing of event amounts	RESOLVED
07	INFO	Lack of validation	RESOLVED
08	INFO	Typographical errors	RESOLVED
09	INFO	Contract does not support tokens with a fee on transfer	RESOLVED
10	INFO	Various functions can be made external	RESOLVED
11	INFO	Lack of `safeTransfer` usage	RESOLVED
12	INFO	`payDebt` does not adhere to checks-effects-interactions	RESOLVED

1.3.2 BankrollShare

ID	Severity	Summary	Status
13	INFO	`decimals` can be made immutable	RESOLVED
14	INFO	Unused function: `burn`	RESOLVED
15	INFO	Typographical errors	PARTIAL
16	INFO	Lack of events for `withdrawUnderlying` and `_useAllowance`	RESOLVED
17	INFO	Lack of `safeTransfer` usage	RESOLVED
18	INFO	Non-zero requirements should always be done within `_transfer`	RESOLVED
19	INFO	Various functions can be made external	RESOLVED

2 Findings

2.1 Bankroll

The `Bankroll` contract is the primary contract within the ChainBet ecosystem. It manages all tokens for the "house" of ChainBet: the tokens (ETH, USDC….) that users can win.

The owner of the `Bankroll` contract is the ChainBet governance. The governance can register new tokens as house tokens with the `addPool` function. After a token has been registered, users are free to deposit and withdraw tokens into this token's pool via the `deposit` and `withdraw` functions.

The core feature of the `Bankroll` contract is that it allows users to provide their tokens as liquidity for the ChainBet house. Users then share the risks, but also the expected profits, of the ChainBet system. In case the ChainBet games earns a profit for the house, the users staking in that pool will see their staked balance value increase. In case the ChainBet games ends up losing money for the house, the users staking in the pools will see their staked balance value decrease.

Finally, as games usually take a few transactions to complete, the contract contains accounting functionality where individual games can reserve potential profits for withdrawal. Games reserve tokens using `reserveDebt` and whenever a payout is made, the won amount is deducted via `payDebt` while the missed amount is deducted via `clearDebt`. We expect that in many cases, the games will need to call both functions. The Bankroll contract validates that any individual `reserveDebt` call does not exceed a certain percentage of the house balance.

Disclaimer: It should be noted that not a single ChainBet game was included within the scope of this audit and this report should therefore not be considered as any form of safety over the individual games. As the games have full access over users' funds within this contract, the users should be careful as they can pose a real and severe threat upon their deposits. Users could lose up to all of their deposited funds.

2.1.2 Privileged Functions

The following functions can be called by the owner and other privileged roles of the contract:

- `setWhitelist`
- `setMaxWin`
- `addPool`
- `removePool`
- `clearDebt [ games ]`
- `payDebt [ games ]`
- `reserveDebt [ games ]`
- `transferOwnership`
- `renounceOwnership`

2.1.3 Issues & Recommendations

Issue #01	Governance privilege: Governance can whitelist new games that are allowed to withdraw all pool balances
Severity	HIGH SEVERITY
Description	The ChainBet system uses a centralized Bankroll contract to store all house tokens provided by users. Users can stake tokens into the contract to take a cut of the house profits. However, as the ChainBet team needs to be able to add new games over time, they could one day add a bad game that drains the whole Bankroll. This could be done through malicious intent, in what is called a "rug" or more commonly and innocently through key theft. If a hacker ever gets hold of the owner key of the Bankroll contract they can drain all provided tokens from the contract. There is therefore both a risk of malicious intent and a risk for key theft. We will make recommendations to address both.
Recommendation	In the long term, consider putting ownership for addPool behind a timelocked multisig (pausing pools should arguably still be instantly doable as a safeguard). In the short term to reduce the risk of malicious intent, the client can consider undergoing a KYC program. To summarize: Both risks can be addressed by carefully designing a governance structure using a multisig and/or timelock. The client should carefully ensure that they can still however pause the relevant functionality instantly to ensure that they can take action when a component malfunctions. In the short term and to boost user confidence, the governance can consider undergoing a KYC program. A final recommendation with regards to theft of the provided tokens that can be made is to add daily or hourly limits of the pool balances that can be withdrawn to any single game. If a game ever turns out to be exploitable, it cannot drain the whole pool this way. We highly encourage the team to carefully consider adding in a safeguard like this as well as the games were not audited by Paladin and we therefore cannot give a single safety guarantee over them.

Resolution	PARTIALLY RESOLVED The client has added daily limits to what an individual game can take from the contract. The governance issue remains for now although the client has indicated they will delegate ownership to multisig and later to governance contract, as well as putting it behind a timelock.

Issue #02	`emergencyWithdraw` lacks debt validation
Severity	MEDIUM SEVERITY

Description	The `withdraw` function contains a validation to ensure that the amount is not allocated to a game.
	
	Line 357-360
	`require(`
	`    (reserves(address(token)) - amount) >=`
	`debtPools[address(token)],`
	`    "BR: remaining reserves less than debt"`
	`);`
	
	This validation is however not present on the `emergencyWithdraw` function (alongside the `hasBankrollPool` modifier).
	
	While we understand and encourage having an `emergencyWithdraw` function that will always work, we fail to understand what the use case of the `withdraw` function is as it does not provide any benefit to users over the `emergencyWithdraw` function.
	
	These inconsistencies therefore seem like mistakes to us and we believe that it would be sufficient to only keep the `emergencyWithdraw` logic, or stick to the `withdraw` implementation by having `emergencyWithdraw` call the `withdraw` function.

Recommendation	Consider whether these inconsistencies are desired. If not, consider either removing one of the functions or having `emergencyWithdraw` call the `withdraw` implementation (make sure to adjust the `nonReentrant` modifiers adequately by having an internal `withdraw` function).
Resolution	RESOLVED The client moved `emergencyWithdraw` to the same implementation as `withdraw`. The only difference is that even if a pool is de-whitelisted, emergency withdraw still works.

Issue #03	Protocol is vulnerable to JIT quitting of liquidity
Severity	LOW SEVERITY
Description	Liquidity providers can leave at any point in time after waiting the necessary depositing day. If a provider sees a large loss in the mempool they might decide to unstake before the transaction confirms, causing the loss to be borne by the other liquidity providers. Even worse, an exploiter might even provide a large part of the liquidity (say 50%) and make a large set of bets. If most of them lose, the exploiter leaves the pool before his liquidity bears the loss. If most of the bets win, the exploiter stays to reap the profits. This issue is marked as low severity as the contract already makes some attempt at reducing the profitability of this by requiring users to stake for a full day.
Recommendation	As this issue is quite fundamental, consider carefully designing additional safeguards to prevent it in case it is considered a significant threat.
Resolution	ACKNOWLEDGED

Issue #04	Return values of `burnFrom` are not handled
Severity	LOW SEVERITY

Description	Line 326
	`shareToken.burnFrom(_msgSender(), poolBalance);`
	
	Line 362
	`shareToken.burnFrom(_msgSender(), shares);`
	
	The `burnFrom` function returns a `success` boolean. This is however not used. This issue has been marked as low severity due to the fact that this boolean is in fact not really used within the system as the contract currently always returns true.

Recommendation	Consider either removing the return value from the share token contract or requiring it to be true.
Resolution	RESOLVED The return value is removed.

Issue #05	`maxBetChanged` emits the wrong parameters
Severity	LOW SEVERITY

Location	Line 246-250
	`function setMaxWin(uint256 new_max) external onlyOwner {`
	`    require(new_max > 0, "BR:invalid new max win");`
	`    max_win = new_max;`
	`    emit MaxBetChanged(max_win, new_max);`
	`}`

Description	The `MaxBetChanged` event has the old maximum and the new maximum as parameters. However, the old maximum is set to the new maximum before the event gets emitted causing the event to emit the same variable twice.

Recommendation	Consider caching the old maximum in a new variable before changing it to the new maximum:
	`uint256 old_max = max_win;`
	`max_win = new_max;`
	`emit MaxBetChanged(old_max, new_max);`

Resolution	RESOLVED

Issue #06	Unnecessary indexing of event amounts
Severity	INFORMATIONAL
Location	Line 198 `event MaxBetChanged(uint256 indexed oldMax, uint256 indexed newMax);`
Description	The contract unnecessarily indexes the old and new maximum amounts. Indexing is only relevant for values that are worth filtering on. In general, numbers which are not qualitative indices themselves are not good candidates for indexing.
Recommendation	Consider removing the indexed modifier from both parameters to save gas.
Resolution	RESOLVED

Issue #07	Lack of validation
Severity	INFORMATIONAL
Description	The contract contains functions with parameters which are not properly validated. Having unvalidated parameters could allow the governance or users to provide variable values which are unexpected and incorrect. This could cause side-effects or worse exploits in other parts of the codebase. Consider validating the following function parameters: Line 246 `function setMaxWin(uint256 new_max) external onlyOwner {` The `new_max` should be at most `10_000`. The variable name should furthermore avoid snake casing.
Recommendation	Consider validating the function parameters mentioned above.
Resolution	RESOLVED

Issue #08	Typographical errors
Severity	INFORMATIONAL

Description	The contract contains a number of typographical errors which we have consolidated below in a single issue in an effort to keep the report size reasonable.
	
	<u>`Line 212`</u>
	`uint256 public max_win = 100;`
	
	Using snake case (_) in Solidity variable names is generally discouraged as it is not the accepted practice. This might make the code look unappealing to third party reviewers.
	
	<u>`Line 229`</u>
	`isWhitelisted(msg.sender),`
	
	The contract almost consistently uses `_msgSender()`, consider being consistent and using it here as well. Not using `msgSender()` consistently makes the usage of it useless as the function cannot be overridden.
	
	<u>`Line 350`</u>
	`uint256 amount = (shares *`
	`token.balanceOf(address(shareToken))`
	
	The contract sometimes uses `balanceOf(address(shareToken))` and other times uses the `reserves(address(token))` function. Consider using just one of them to be consistent.
	
	<u>`Line 276`</u>
	`require(hasPool(address(token)), "BR:no pool for token");`
	
	This requirement is already asserted in the `hasBankrollPool` modifier.

Line 380

```
require(token.balanceOf(_msgSender()) >= amount,
"insufficent balance");
```

This error should say "insufficient", not "insufficent".

Line 405

```
/// @param amount amount tto remove from the debtPool
```

This comment should say "to", not "tto".

Line 406

```
function clearDebt(address token, uint256 amount)
```

Token can be provided as IERC20 directly. The `debtPools` mapping can also use IERC20 as the key type.

Line 421

```
function payDebt(
```

Line 438

```
function reserveDebt(address token, uint256 amount)
```

`token` can be provided as IERC20 directly. The `debtPools` mapping can also use IERC20 as the key type.

Line 443

```
require(reserves(token) >= amount, "BR:amount exceeds
reserves");
```

This requirement is indirectly validated later on and can therefore be removed to save gas.

Recommendation	Consider fixing these typographical errors.
Resolution	RESOLVED Most of the issues have been resolved.

Issue #09	Contract does not support tokens with a fee on transfer
Severity	INFORMATIONAL
Description	The contract does not support tokens with a fee on transfer due to the deposit crediting too much shares for the actually received amount of tokens.
Recommendation	Consider whether such tokens ever need to be supported. In case they do not need to be supported nothing needs to be done. In case they need to be supported Paladin will assist to implement logic to support them. It should be noted that this logic increases gas usage.
Resolution	RESOLVED The client does not plan to add such tokens.

Issue #10	Various functions can be made external
Severity	INFORMATIONAL
Description	Functions that are not used within the contract but only externally can be marked as such with the external keyword. Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases. The following functions can be marked as external: - `balanceOf` - `withdraw` - `deposit` - `clearDebt` - `payDebt` - `reserveDebt`
Recommendation	Consider marking the functions mentioned above as external.
Resolution	RESOLVED

Issue #11	Lack of `safeTransfer` usage
Severity	INFORMATIONAL
Location	Line 389 `token.transferFrom(_msgSender(), address(shareToken), amount);`
Description	In the `deposit` function, the `transferFrom` method is used to transfer tokens from the user to the `shareToken`. This will not work for tokens that returns `false` on `transfer` (or malformed tokens that do not have a return value).
Recommendation	Consider using `safeTransferFrom` instead of `transferFrom`.
Resolution	RESOLVED

Issue #12	`payDebt` does not adhere to checks-effects-interactions
Severity	INFORMATIONAL
Location	Line 429 `pools[token].withdrawUnderlying(recipient, amount);`
Description	The `payDebt` function does not adhere to the checks-effects-interactions best practice. This issue is marked as informational given that it cannot be abused. However, it has still been included as to communicate best practices with the client.
Recommendation	Consider doing `withdrawUnderlying` last in the function.
Resolution	RESOLVED

2.2 BankrollShare

`BankrollShare` is an `ERC20` token implementation used by the main `Bankroll` contract. It stores an underlying token like `ETH` and each `BankrollShare` (`brETH`) represents a percentage ownership share of the underlying tokens. If the underlying ETH bankroll in the contract increases over time, the users' claim increases accordingly.

The `BankrollShare` token therefore represents the shareholdership over the stored tokens within the contract. It can be transferred to other users and can be redeemed at any time through the `Bankroll` contract.

A single restriction does apply: once tokens are minted to the user when they deposit say `ETH` into the `Bankroll` token in exchange for shares, the user cannot transfer or redeem the shares for a whole day. This policy is an effective safeguard against malicious parties quickly entering and leaving the `BankrollShare` to capture the "house value" of the ChainBet games.

2.2.1 Privileged Functions

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

- `mint`
- `withdrawUnderlying`
- `transferOwnership`
- `renounceOwnership`

2.2.2 Issues & Recommendations

Issue #13	`decimals` can be made immutable
Severity	INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the immutable keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the variable explicitly immutable.
Resolution	RESOLVED

Issue #14	Unused function: burn
Severity	INFORMATIONAL
Description	Functions which are defined in a contract but remain unused could confuse third-party auditors. They also increase the contract length unnecessarily.
Recommendation	Consider removing the function to keep the contract short and simple.
Resolution	RESOLVED The function has been removed.

Issue #15	Typographical errors
Severity	INFORMATIONAL

Description	The contract contains a number of typographic mistakes which we've enumerated below in a single issue in an effort to keep the report size reasonable.
	
	<u>`Line 24`</u>
	`address token_,`
	
	`token_` can be provided as the `IERC20Metadata` type to avoid casting it later on.
	
	<u>`Lines 43-45`</u>
	`mapping(address => UserBalances) public users;`
	`/// @notice owner > spender > allowance mapping.`
	`mapping(address => mapping(address => uint256)) public override allowance;`
	
	These variables should be moved above the `withdrawUnderlying` function.
	
	The contract is also inconsistent with using msg.sender as `Bankroll` uses `_msgSender()`. We recommend sticking to `_msgSender()` consistently if it is used anywhere in the codebase.

Recommendation	Consider fixing the typographical errors.
Resolution	PARTIALLY RESOLVED `token_` is still provided as an address.

Issue #16	Lack of events for `withdrawUnderlying` and `_useAllowance`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications. The `_useAllowance` function should emit an `Approval` event to allow all approval changes to be tracked through events. This is generally considered best practice in ERC20 implementations and is done within the OpenZeppelin implementation as well.
Recommendation	Add events for the functions. The client does not strictly need to add an event to `withdrawUnderlying` as an event is emitted at the `Bankroll` level. This issue will be resolved regardless of whether the client decides to add an event to this function.
Resolution	RESOLVED

Issue #17	Lack of `safeTransfer` usage
Severity	INFORMATIONAL
Location	Line 39 `token.transfer(recipient, amount);`
Description	In the `withdrawUnderlying` function, the `transfer` method is used to transfer tokens from the contract to the recipient. This will not work for tokens that returns `false` on transfer (or malformed tokens that do not have a return value).
Recommendation	Consider using `safeTransfer` instead of `transfer`.
Resolution	RESOLVED

Issue #18	Non-zero requirements should always be done within `_transfer`
Severity	INFORMATIONAL
Description	The contract only checks that transfers are not made to the zero address if the transferred amount is greater than zero. This causes an issue with off-chain indexers that are trying to index mint or burn transactions. Essentially a user can call `transferFrom(address(0), target, 0)` to muddle the mint event history with nonsense mints to the target.
Recommendation	Consider always checking both the from and the to address to not be zero at the beginning of the `_transfer` function.
Resolution	RESOLVED

Issue #19	Various functions can be made external
Severity	INFORMATIONAL
Description	Functions that are not used within the contract but only externally can be marked as such with the external keyword. Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases. The following functions can be marked as external: - `withdrawUnderlying` - `balanceOf` - `lockedUntil` - `transfer` - `transferFrom` - `approve` - `mint` - `burn` - `burnFrom`
Recommendation	Consider marking the functions mentioned above as external.
Resolution	RESOLVED

PALADIN
BLOCKCHAIN SECURITY