GUARDIAN AUDITS

SMART CONTRACT SECURITY AUDIT OF

UltiBets

Betting on Blockchain

Summary

Audit Firm: Guardian Audits

Client Firm: UltiBets

Final Report Date: July 4th, 2022

Audit Summary

After a line by line manual analysis and automated review, Guardian Audits has concluded that:

- UltiBet's smart contracts have an LOW RISK SEVERITY
- UltiBet's smart contracts have an **ACTIVE OWNERSHIP**
- UltiBet's smart contract owner has multiple "write" privileges. Centralization risk correlated to the active ownership is **MEDIUM**

Notice that the examined smart contracts are not resistant to internal exploit. For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

🔗 Blockchain network: **Fantom Opera**

✅ Verify the authenticity of this report on Guardian's GitHub: https://github.com/guardianaudits

Table of Contents

Project Overview

Project Summary

Project Name	UltiBets
Language	Solidity
Codebase	https://github.com/UltiBets/Audits
Commit	e3f94f795e786310c7e4b8b5dd9d31721d19feb9

Audit Summary

Delivery Date	July 4th, 2022
Audit Methodology	Static Analysis, Manual Review, Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	2	0	0	0	0	2
High	3	0	0	2	0	1
Medium	8	0	0	1	0	7
Low	21	0	0	0	0	21

Audit Scope & Methodology

Scope

ID	File	SHA-1 Checksum
UB	UltiBets.sol	5D31E73864814199009EF6B7A524C88B67976CA1
UBF	UltiBetsFactory.sol	8D9969E4A39B2833021FB42FF409C92A5A8BCEE0
UBT	UltiBetsTreasury.sol	DA47EEF6B65084C51665FC42B05C5064E3CA306D
CA	CustomAdmin.sol	D63F2D5DB0B73F063662DF9D7E79F0244CD21DA6
20A	ERC20Airdrop.sol	C9C23B1C0925A2A3E4FC4A361D6B2DD762918ED2
721A	ERC721Airdrop.sol	D7E8F739B2B2EAF118393CC80E9AE76BEE865739
MS	Multisig.sol	B5AF097CF9BFDF2D2B995F12B924CBCBDA0E7FCB
SQDR	SquidBetPlayersRegistration.sol	49C530151B67EBB59AFEA18BE1938CFFEECA45AF
SQD1	SquidBetStartRound.sol	A2FE0D5BB7DA22F58E4D801E6DF8E1816E18409B
SQD2	SquidBetSecondRound.sol	D1F9F72B9C18BFA3A9973F26683BA4FB90B95494
SQD3	SquidBetThirdRound.sol	402ECCE8B15A3983CD05DF7FF964111FB4C065F6
SQD4	SquidBetForthRound.sol	1764409DA4A8AE4F975B6F879E4E6EA8B08A1529
SQDF	SquidBetFinalRound.sol	7D747CFF25CE4B32C220E021E75826FC45FB7F46
SQDP	SquidBetPrizePool.sol	2D9437C942392EFCA865743EA43EF157C1C3D089

Audit Scope & Methodology

Methodology

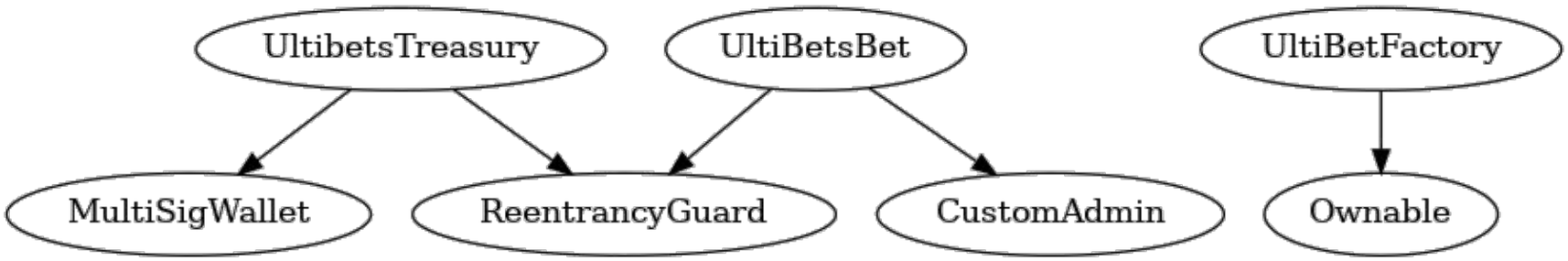
The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Vulnerability Classifications

Vulnerability Level	Classification
🔴 Critical	Easily exploitable by anyone, causing loss/manipulation of assets or data.
🟠 High	Arduously exploitable by a subset of addresses, causing loss/manipulation of assets or data.
🟡 Medium	Inherent risk of future exploits that may or may not impact the smart contract execution.
🟢 Low	Minor deviation from best practices.

Inheritance Graph



Findings & Resolutions

ID	Title	Category	Severity	Status
GLOBAL-1	Poor Practices	Best Practices	Low	Resolved
GLOBAL-2	Centralization Risk	Centralization / Privilege	High	Acknowledged
UB-1	Total Bets Not Updated	Logical Error	Critical	Resolved
UB-2	Order of Operations	Logical Error	High	Resolved
UB-3	Report Result Twice	Logical Error	Medium	Resolved
UB-4	Same Winner and Loser	Logical Error	Medium	Resolved
UB-5	Potential DoS	Denial-of-Service	Low	Resolved
UB-6	Inaccurate Comments	Inaccurate Comments	Low	Resolved
UB-7	Superfluous Code	Optimization	Low	Resolved
UB-8	Superfluous Code	Optimization	Low	Resolved
UBT-1	Treasury Cannot Receive ETH	Logical Error	Critical	Resolved
UBT-2	Superfluous Code	Optimization	Low	Resolved
UBT-3	Typo	Typo	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
UBT-4	Superfluous Code	Optimization	Low	Resolved
UBT-5	Vague Event Information	Events	Low	Resolved
UBT-6	Inefficient Addition	Optimization	Low	Resolved
MS-1	Inaccurate Comment	Inaccurate Comments	Low	Resolved
UBF-1	Inaccurate Variable Name	Best Practices	Low	Resolved
CA-1	Inaccurate Comment	Inaccurate Comments	Low	Resolved
SQDF-1	Report Result Twice	Logical Error	Medium	Resolved
SQDF-2	Arbitrary Voting Results	Logical Error	Medium	Resolved
SQDF-3	Potential DoS	Denial-of-Service	Low	Resolved
SQDF-4	Superfluous Code	Optimization	Low	Resolved
SQDF-5	Weak Source of Randomness	Randomness	High	Acknowledged
SQDR-1	No Check For Max Players	Logical Error	Medium	Resolved
SQDR-2	Can’t Update Betting Fee	Logical Error	Medium	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
SQPR-1	Can’t Get Prize	Denial-of-Service	Medium	Resolved
SQPR-2	Arbitrary Prize Distribution	Arbitrary Control	Medium	Acknowledged

GLOBAL-1 | Poor Practices

Category	Severity	Location	Status
Best Practices	● Low	Global	Resolved

Description

Throughout the contracts there are myriad instances of:

- Lack of camelcase
- Use of SafeMath when Solidity version ^0.8.0 has overflow and underflow checks
- Unnecessary local variables which waste gas e.g. `address to = payable(ultibetsTreasury)`
- Redundant boolean checks e.g. `== true`
- Typos in the comments
- Many functions can be declared `external`

Recommendation

Use camelcase throughout the contracts, remove redundant and unnecessary local variables, do not perform redundant boolean checks, revise comments, remove SafeMath operations to save gas, declare functions not used internally as `external`.

Resolution

- Typos, code-style, function declarations, and frivolous code has been addressed.

GLOBAL-2 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	High	Global	Acknowledged

Description

Throughout the contracts there is a risk of admins and oracles using their privilege to benefit themselves or act malicious toward user's holdings. Contracts utilizing the CustomAdmin access control model face more risk as the number of admins or oracles increase because it only takes one to act mischievous.

Recommendation

Ensure that privileged addresses such as admins are all a multi-sig and/or introduce a timelock for improved community oversight. Secure a KYC for increased community trust.

Resolution

- The risk is acknowledged and will be limited with appropriate measures such as multi-sig addresses and community transparency.

UB-1 | Total Bets Not Updated

Category	Severity	Location	Status
Logical Error	● Critical	UltiBets.sol	Resolved

Description

The `bets` mapping is not being updated in `placeBet`. When `withdrawGain` is called, `.div(bets[result.winner])` would lead to a division by 0 error every time. Therefore, no one would be able to withdraw their gains, leading to complete loss of funds.

Recommendation

In `placeBet`, increment the `bets` mapping with `bets[_side] += betAmount`.

Resolution

- The `bets` mapping is now updated in `placeBet`.

UB-2 | Order of Operations

Category	Severity	Location	Status
Logical Error	● High	UltiBets.sol	Resolved

Description

The logic does not correctly calculate the `gain` due to the SafeMath order of operations. With the SafeMath operations the addition is performed first, but what is needed is to first calculate the winner's part of the loser funds and then finally adding it to `BettorBetWinner`.

For example:

3.add(10).mul(3).div(6) = 6

3 + 10 * 3 / 6 = 8

Recommendation

Replace with `BettorBetWinner + bets[result.loser] * BettorBetWinner / bets[result.winner]`.

Resolution

- The logic has been replaced as suggested.

UB-3 | Report Result Twice

Category	Severity	Location	Status
Logical Error	Medium	UltiBets.sol: 174	Resolved

Description

`reportResult` does not require the event to be finished. Therefore, the result can be reported multiple times and the winner and loser swapped while the treasury fee is taken multiple times.

Recommendation

Add `require(!isEventFinished)` in `reportResult` so a result cannot be reported twice.

Resolution

- The suggested requires has been added.

UB-4 | Same Winner and Loser

Category	Severity	Location	Status
Logical Error	Medium	UltiBets.sol: 174	Resolved

Description

`reportResult` does not prevent the oracle from setting the winner and loser to the same side which can lead to loss of funds for many users.

Consider a 100 ether bet on one side and 10 ether bet on another, but the 100 ether side is chosen as both the winner and loser. From the calculation in `withdrawGain`, the contract will attempt to payout a total of 200 ether while it only holds 110. Therefore the users who claim first will deplete the contract and the ones who claim later will experience a complete loss of funds.

Recommendation

Add a safety check that the arguments `_winner` and `_loser` are different.

Resolution

- The suggested requires has been added.

UB-5 | Potential DoS

Category	Severity	Location	Status
Denial-of-Service	● Low	UltiBets.sol: 185-187	Resolved

Description

`reportResult` depends on the ETH transfer to the treasury to be successful in order for the result to be reported. Such a call can fail, leading to a DoS condition.

Recommendation

Remove lines 185-187 as there is no need to have the transfer logic there. External calls should ideally be isolated into another transaction and the admin calling `withdrawEarnedFees` serves that purpose.

Resolution

- Unnecessary transfer logic has been removed.

UB-6 | Inaccurate Comment

Category	Severity	Location	Status
Inaccurate Comments	● Low	UltiBets.sol: 144	Resolved

Description

The comment for `stopBet` states that "... this action can only be performed by an administrator" but the modifier is `OnlyOracle`. In addition, the comment states that it is an emergency function, yet it is required to be called in order for a bettor to `withdrawGain`.

Recommendation

Update the comments to accurately reflect the function. In addition, verify if `stopBet` is needed. If it is truly only used for emergency situations, it does not make sense to have it required so user's can withdraw their gains.

Resolution

- The comment has been updated.

UB-7 | Superfluous Code

Category	Severity	Location	Status
Optimization	Low	UltiBets.sol: 21, 22	Resolved

Description

The `betAmountForYes`, and `betAmountForNo` variables are never assigned in the `bettorBetHistory` mapping.

Recommendation

Remove them from the struct or use them to replace the `betsAmountPerBettor` mapping.

Resolution

- The variables have been removed.

UB-8 | Superfluous Code

Category	Severity	Location	Status
Optimization	● Low	UltiBets.sol: 36, 37	Resolved

Description

The `inArrayYes` and `inArrayNo` variables are never used.

Recommendation

Remove them.

Resolution

- The variables have been removed.

UBT-1 | Treasury Cannot Receive ETH

Category	Severity	Location	Status
Logical Error	● Critical	UltiBetsTreasury.sol	Resolved

Description

Because there is no `receive` or fallback `function`, any contract that relies on sending ETH to the treasury will face unintended consequences such as stuck fees.

Recommendation

Add a `receive() external payable { }` function to the contract.

Resolution

- A receive function has been added.

UBT-2 | Superfluous Code

Category	Severity	Location	Status
Optimization	● Low	UltiBetsTreasury.sol	Resolved

Description

The `require(msg.sender == Admin)` can be deduplicated into an `onlyAdmin` modifier used for every permissioned function.

Recommendation

Deduplicate the `require` logic into an `onlyAdmin` modifier and apply the `onlyAdmin` modifier to every permissioned function.

Resolution

- The modifier has been created and added to relevant functions.

UBT-3 | Typo

Category	Severity	Location	Status
Typo	Low	UltiBetsTreasury.sol: 148	Resolved

Description

The `_descreption` parameter in `createAllocation` is misspelled.

Recommendation

Correct it to `_description`.

Resolution

- The typo has been fixed.

UBT-4 | Superfluous Code

Category	Severity	Location	Status
Optimization	Low	UltiBetsTreasury.sol: 198	Resolved

Description

Since the `amount` variable is only used once and the `allocations[msg.sender].salary` value is not changed before the use of `amount`, it is unnecessary to declare the `amount` variable.

Recommendation

Remove the declaration and use of the `amount` variable for gas optimization.

Resolution

- The variable has been removed.

UBT-5 | Vague Event Information

Category	Severity	Location	Status
Events	● Low	UltiBetsTreasury.sol: 243	Resolved

Description

The `SalaryChanged` event is emitted with just the `block.timestamp` and `_newSalary`, leaving no record of which address the salary was changed for.

Recommendation

Include the `_address` as a part of the `SalaryChanged` event data.

Resolution

- The address is now included in the `SalaryChanged` event data.

UBT-6 | Inefficient Addition

Category	Severity	Location	Status
Optimization	Low	UltiBetsTreasury.sol: 274	Resolved

Description

The `allocations[msg.sender].totalPayout = allocations[msg.sender].totalPayout.add(amount)` computation inefficiently references the allocations state data twice.

Recommendation

Use `+=` in order to save gas.

Resolution

- `+=` is now used.

MS-1 | Inaccurate Comment

Category	Severity	Location	Status
Inaccurate Comments	● Low	MultiSig.sol: 98-100	Resolved

Description

The comment states “Public Functions” yet `initialize` directly below is internal.

Recommendation

Move comment to where the section below is solely public functions.

Resolution

- The comment has been updated.

UBF-1 | Inaccurate Variable Name

Category	Severity	Location	Status
Best Practices	● Low	UltiBetFactory.sol: 18	Resolved

Description

The mapping `addressToId` is really the contract id to the address of the contract.

Recommendation

Change name to `idToAddress`.

Resolution

- The name of the mapping has been updated.

CA-1 | Inaccurate Comment

Category	Severity	Location	Status
Inaccurate Comments	Low	CustomAdmin.sol: 46	Resolved

Description

The comment states that the function adds the specified address to the list of `administrators` but it updates the address in the mapping of `Oracles`.

Recommendation

Update the comment to reflect what the function does.

Resolution

- The comment has been updated.

SQDF-1 | Report Result Twice

Category	Severity	Location	Status
Logical Error	Medium	SquidBetFinalRound.sol: 108,119	Resolved

Description

`isResultReported` is not set to true after the call to `reportResult`. Therefore, the result can be reported multiple times with varying arguments. Furthermore, line 119 `isVotingClosed = false` can be used to prevent a winner from ever getting picked in `pickWinner()`.

Recommendation

Add `isResultReported = true` to the end of the function.

Resolution

- The suggested addition was made.

SQDF-2 | Arbitrary Voting Results

Category	Severity	Location	Status
Logical Error	Medium	SquidBetFinalRound.sol: 163	Resolved

Description

In the `resultVote` function, the `if` statements that determines which `finalVoteDecision` to return are placed inside of the `for` loop that counts the votes. Therefore the `finalVoteDecision` will be arbitrarily based on whichever votes happen to be first in the `votes` list.

Recommendation

Move the `if` statements determining `finalVoteDecision` after the `for` loop, or preferably refactor the voting entirely per SQDF-3.

Resolution

- The `if` statement has been moved outside of the for loop

SQDF-3 | Potential DoS

Category	Severity	Location	Status
Denial-of-Service	Low	SquidBetFinalRound.sol: 156	Resolved

Description

The `for` loop is reliant on the number of votes. If the number of votes is very high, the calculation may exceed the block gas limit and fail.

Recommendation

Another way to approach this problem is to have an int variable that increases by 1 when the player votes one way and decreases by 1 when the player votes the other way in Vote. The sign of the end value once voting is finished will tell you which choice had more votes.Therefore, a for loop can be entirely avoided.

Resolution

- The suggested approach was implemented.

SQDF-4 | Superfluous Code

Category	Severity	Location	Status
Optimization	Low	SquidBetFinalRound.sol: 115-116	Resolved

Description

`reportResult` does not need to contain logic for transferring funds to the prize pool because `transferTotalEntryFeestoPrizePool` exists.

Recommendation

Remove transfer logic from `reportResult`.

Resolution

- Transfer logic was removed from `reportResult`.

SQDF-5 | Weak Source of Randomness

Category	Severity	Location	Status
Randomness	High	SquidBetFinalRound.sol: 176, 188	Acknowledged

Description

`pickWinner` uses weak sources of on-chain randomness. A validator can exploit this in order to obtain a winner that is beneficial to themselves.

Recommendation

Utilize a strong source of randomness whether it be the on-chain randomness pattern or an oracle.

Resolution

- Randomness will be secured with the implementation of Chainlink VRF.

SQDR-1 | No Check For Max Players

Category	Severity	Location	Status
Logical Error	Medium	SquidBetPlayersRegistration.sol: 68	Resolved

Description

`registerPlayer` does not check if registration exceeds the `maxNumberOfPlayers`. As a result, contracts that loop over the winners such as SquidBetPrizePool may face a DoS attack.

Recommendation

Add the check in `registerPlayer`.

Resolution

- The suggested requires has been added.

SQDR-2 | Can’t Update Betting Fee

Category	Severity	Location	Status
Logical Error	Medium	SquidBetPlayersRegistration.sol: 112	Resolved

Description

The betting fee is unable to be updated because `updateBettingFees` updates the `maxNumberofPlayers` rather than `bettingFee`.

Recommendation

Change the function body to `bettingFee = _newBettingFee`.

Resolution

- The function now updates `bettingFee`.

SQPR-1 | Winners Can’t Get Prize

Category	Severity	Location	Status
Denial-of-Service	Medium	SquidBetPrizePool.sol: 81	Resolved

Description

`winnersClaimPrizePool` uses a push pattern such that it loops over all winners and sends them their funds. If there are too many equal winners then no one would be able to claim due to the for loop exceeding the block gas limit. Furthermore, if a winner is a contract that is unable to receive ether upon `.transfer()`, no one would be able to receive their prize money.

Recommendation

Utilize a pull-over-push pattern such that when a user calls `winnersClaimPrizePool`, they receive solely their funds and no other funds are dispatched.

Resolution

- A pull over push pattern was introduced.

SQPR-2 | Arbitrary Prize Distribution

Category	Severity	Location	Status
Arbitrary Control	Medium	SquidBetPrizePool.sol: 71	Acknowledged

Description

Anyone can call the function `winnersClaimPrizePool` and end the prize pool as long as some winners have been added.

Recommendation

Make sure to always add all winners at once.

Resolution

- The risk has been acknowledged.

Auditor’s Verdict

After a line by line manual analysis and automated review, Guardian Audits has concluded that:

- UltiBet’s smart contracts have an LOW RISK SEVERITY
- UltiBet’s smart contracts have an **ACTIVE OWNERSHIP**
- UltiBet’s smart contract owner has multiple “write” privileges. Centralization risk correlated to the active ownership is **MEDIUM**

Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Guardian’s position is that each company and individual are responsible for their own due diligence and continuous security. Guardian’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Guardian Audits

Founded in 2022 by DeFi experts, Guardian Audits is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian Audits upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit https://guardianaudits.com

To view our audit portfolio, visit https://github.com/guardianaudits

To book an audit, message https://t.me/guardianaudits