ARMUR

SMART CONTRACT
AUDIT REPORT
SOLIDITY

Infini3

12TH FEB 2023

akhil@armur.ai
amritansh@armur.ai

Tournament Contract

The Tournament smart contract implements a tournament system for players to participate in and compete for prizes. The tournament has an owner who is responsible for managing the tournament, registering players, and updating the leaderboard. The contract is built using OpenZeppelin libraries, specifically Ownable and PullPayment, as well as an implementation of the EnumerableSet data structure.

Contract Properties

- **details**: A struct that holds the details of the tournament, including the id, title, participationFee, endTime, and prizes.
- **players**: An instance of the EnumerableSet.AddressSet library, which is used to store the addresses of all players participating in the tournament.
- **leaderboard**: An array of PlayerData structs, which stores the wallet addresses and highscores of each player.
- **isClosed**: A boolean variable that indicates whether the tournament has ended or not.
- **playerDetails**: A mapping that maps an address to its corresponding PlayerData struct.
- **isRegistered**: A mapping that maps an address to a boolean value, indicating whether the player has registered or not.

This is a public report. The security threats mentioned in this audit report are critical. Armur A.I is not responsible for security hacks that may happen due to negligence and failure to fix the issues before deployment.

Contract Events

- **PlayerRegistered**: Emitted whenever a player is registered for the tournament. The event holds the wallet address of the player who was registered.

- **LeaderboardUpdated**: Emitted whenever the leaderboard is updated. The event holds the wallet address and highscore of the player whose score was updated.

Contract Methods

- **getDetails()**

This function returns the details of the tournament, including the id, title, participationFee, endTime, and prizes.

- **getLeaderboard()**

This function returns the leaderboard, which is an array of PlayerData structs, sorted by the player's highscore. This function can only be called after the tournament has ended.

- **getPlayers()**

This function returns a list of all players participating in the tournament, including their wallet addresses and highscores.

- **register(address wallet)**

This function allows the owner of the contract to register a player for the tournament. The wallet address of the player must be passed as an argument. The function can only be called while the tournament is ongoing.

- **addScore(address wallet, uint256 score)**

This function allows the owner of the contract to update the highscore of a player. The wallet address and the new score of the player must be passed

This is a public report. The security threats mentioned in this audit report are critical. Armur A.I is not responsible for security hacks that may happen due to negligence and failure to fix the issues before deployment.

as arguments. The function can only be called while the tournament is ongoing.

Contract Modifiers

- **onlyEnded:** This modifier is used to check if the tournament has ended. If the tournament has not ended, the function calling the modifier will throw an error.

- **onlyOngoing:** This modifier is used to check if the tournament is ongoing. If the tournament has ended, the function calling the modifier will throw an error.

- **onlyClosed:** This modifier is used to check if the tournament has been closed. If the tournament is still open, the function calling the modifier will throw an error.

- **onlyOpen:** This modifier is used to check if the tournament is still open. If the tournament has been closed, the function calling the modifier will throw an error.

Functions

- **The addScore function:**

This function allows the owner to update the highscore of a player. To use this function, the player must have participated in the tournament and should have registered first. The isRegistered mapping is checked to ensure that the player has registered, and the players set is used to check if the player has participated. If the new score is higher than the current highscore, the highscore is updated in the playerDetails mapping.

- **The end function:**

This is a public report. The security threats mentioned in this audit report are critical. Armur A.I is not responsible for security hacks that may happen due to negligence and failure to fix the issues before deployment.

This function is used to close the tournament and sort the leaderboard. The onlyEnded modifier is used to ensure that the tournament has ended, and the isClosed variable is updated to true. The players set is converted to an array and stored in the leaderboard variable. The leaderboard is then sorted using the _quickSort function.

- **The withdraw function:**

This function is used to withdraw the prize of a player who has placed in the tournament. The onlyClosed modifier is used to ensure that the tournament has ended and the leaderboard is available. The require statement is used to check if the player's rank is within the range of available prizes. The prize is then transferred to the player's wallet using the transfer function.

- **The PullPayment contract:**

The Tournament contract inherits from the PullPayment contract, which is part of the OpenZeppelin Contracts library. This contract provides a pull-based payment mechanism, where the recipient can initiate a payment when it wants to. In the Tournament contract, this contract is used to transfer the prize to the winner's wallet when they call the withdraw function.

Code structure

- The contract is written in Solidity, using version 0.8.9 of the compiler.

- The contract imports Ownable and PullPayment contracts from OpenZeppelin, as well as the EnumerableSet utility contract.

- The contract implements the Ownable contract, which adds basic access control functionality to the contract, such as the ability to

This is a public report. The security threats mentioned in this audit report are critical. Armur A.I is not responsible for security hacks that may happen due to negligence and failure to fix the issues before deployment.

manage ownership of the contract, and restrict access to certain functions to only the contract owner.

- The contract also implements the PullPayment contract, which facilitates sending funds to the contract as part of a transaction.

- The contract uses the EnumerableSet utility contract to keep track of the players participating in the tournament.

Functionality

- The contract allows for tournament registration, and tracks the players' high scores.

- The contract also calculates and maintains a leaderboard of the highest-scoring players.

- The contract can be closed by the owner, which freezes the leaderboard and prevents players from adding new scores.

- The contract provides functions for accessing the tournament details, such as the prize pool and end time, as well as the list of players and the current leaderboard.

- The contract also provides events for logging when players are registered and when the leaderboard is updated.

Considerations

There are a few security considerations to be aware of when using this contract:

This is a public report. The security threats mentioned in this audit report are critical. Armur A.I is not responsible for security hacks that may happen due to negligence and failure to fix the issues before deployment.

- The contract is marked as payable, which means that it can receive Ether. However, there is no mechanism to prevent a malicious attacker from sending an excessive amount of Ether to the contract, which could lead to a denial-of-service attack.

- There are a few functions in the contract that have onlyOwner modifiers, but no mechanism for changing the contract owner in case the original owner is compromised or loses access to their account.

- The register function can be used to register a player to the tournament, but there is no mechanism for enforcing the participation fee. An attacker could potentially register without paying the fee.

- The addScore function is used to update a player's score, but it can be called by the owner without any restrictions. This means that the owner could potentially manipulate the leaderboard.

- The getLeaderboard function returns the sorted leaderboard after the tournament has ended, but there is no mechanism to verify that the leaderboard has not been tampered with.

Security Threats To Be Addressed Urgently -

Here are some of the security issues with the smart contract:

- **Lack of access control:** The register and addScore functions can be called by anyone, not just the contract owner. This can allow anyone to add scores to the tournament even if they haven't participated, and skew the results.

- **Unsecured use of mapping:** The playerDetails and isRegistered mappings are public and can be read and written by anyone,

This is a public report. The security threats mentioned in this audit report are critical. Armur A.I is not responsible for security hacks that may happen due to negligence and failure to fix the issues before deployment.

potentially exposing sensitive information or allowing unauthorized updates.

- **Unchecked input:** There is no validation of the input parameters to the addScore function, making it possible to add arbitrary scores to the tournament, potentially affecting the outcome.

- **Unsecured use of require statements:** Many of the require statements can be easily bypassed if the calling code has enough knowledge of the contract, and are not sufficient to prevent malicious behavior.

- **Unsecured sort function:** The _quickSort function does not handle edge cases such as overflow or underflow, and can be vulnerable to a DoS attack by an attacker sending a malicious input to cause the function to run indefinitely.

- **Inadequate error handling:** The contract does not have sufficient error handling, and errors can cause the contract to enter an unexpected state or produce unexpected results.

- **Lack of state management:** The contract does not have any mechanism to manage its state, such as storing the current time or checking if the tournament has ended.

It's important to consider these and other potential security issues before deploying a smart contract to the blockchain.

Recommendations

To address some of the security considerations, consider adding the following modifications:

This is a public report. The security threats mentioned in this audit report are critical. Armur A.I is not responsible for security hacks that may happen due to negligence and failure to fix the issues before deployment.

- Consider adding a mechanism for controlling the maximum amount of Ether that can be sent to the contract to prevent a denial-of-service attack.

- Consider adding a mechanism for changing the contract owner in case the original owner is compromised or loses access to their account.

- Consider adding a mechanism for enforcing the participation fee, such as requiring that players send the fee as part of the registration transaction.

- Consider adding a mechanism for verifying that the leaderboard has not been tampered with, such as adding a signature from the owner or a trusted third party.

Conclusion

- The contract implements the PullPayment interface from OpenZeppelin to handle the participation fee that is paid by players when they register for the tournament. It also uses the Ownable contract from OpenZeppelin to manage the ownership of the contract, which is a good practice.

- The contract uses a mapping to keep track of the details of players who have registered and another mapping to check if a player has already registered or not. It also uses an EnumerableSet to store the addresses of all the players that have registered for the tournament.

- The register function is used to add a player to the tournament and is only accessible by the contract owner. The addScore function is used to update the highscore of a player and is also only accessible by the

This is a public report. The security threats mentioned in this audit report are critical. Armur A.I is not responsible for security hacks that may happen due to negligence and failure to fix the issues before deployment.

contract owner. The leaderboard is sorted using the _quickSort function which implements the Quick Sort algorithm.

- The contract also has several events to notify clients about important changes in the contract state such as PlayerRegistered and LeaderboardUpdated.

- In conclusion, the contract appears to be well-structured and implements several best practices, such as using the OpenZeppelin contracts for common functionality and using events to notify clients about changes in the contract state. However, it is important to thoroughly test the contract in a test environment before deploying it to the main network to ensure that it functions as expected and to avoid any potential security vulnerabilities.

Tournament Manager Contract

The Tournament Manager Contract is a smart contract written in Solidity language and follows the ERC2771 standard. It is designed to create and manage tournaments, handle player registrations, and process referrals. The contract imports several libraries from OpenZeppelin for various functionalities such as ownership, pull payments, reentrancy protection, and more.

Contract Structure

The contract extends the following contracts:

- ERC2771Recipient
- Ownable
- PullPayment
- ReentrancyGuard

This is a public report. The security threats mentioned in this audit report are critical. Armur A.I is not responsible for security hacks that may happen due to negligence and failure to fix the issues before deployment.

Additionally, it imports several libraries:

- ERC2771Recipient.sol
- Ownable.sol
- PullPayment.sol
- ReentrancyGuard.sol
- EnumerableSet.sol
- Context.sol
- Strings.sol
- Tournament.sol

The contract structure includes:

- A struct Player to store player information such as the wallet address, username, tournaments they have registered for, and referrals they have made.

- Two enumerable sets, tournaments and players to store the list of tournaments and players respectively.

- A mapping playerData to store player information using the player's wallet address as the key.

- Constants SAFETY_INTERVAL and REFERRAL_MIN_WITHDRAWAL to set a safety interval for player registration and the minimum withdrawal amount for referrals.

- Several events for logging player updates, tournament creation and ending, referral addition and usage.

- Modifiers onlyRegistered, onlyOfficial, and onlyOngoing to enforce certain conditions before executing certain functions.

Contract Functions

This is a public report. The security threats mentioned in this audit report are critical. Armur A.I is not responsible for security hacks that may happen due to negligence and failure to fix the issues before deployment.

The contract provides several functions:

- **receive** - Function to receive funds and is called when the contract receives external payments.

- **fallback** - Function to catch any transactions that do not match any other functions.

- **constructor** - The constructor function to initialize the contract and set the trusted forwarder.

- **setTrustedForwarder** - Function to set the trusted forwarder, available only to the contract owner.

- **createTournament** - Function to create a new tournament, available only to the contract owner.

- **changeUsername** - Function to change the username of a registered player.

- **register** - Function to register a player for a tournament.

- **withdrawReferrals** - Function to allow players to withdraw their referral rewards.

- **useReferral** - Function to allow players to use referrals to get a discount on tournament participation fees.

- **addReferral** - Function to allow players to refer others to the platform and earn rewards.

- **tournamentsWithdrawable()**: This function returns the list of all tournaments that are end and players can withdraw their winnings.

This is a public report. The security threats mentioned in this audit report are critical. Armur A.I is not responsible for security hacks that may happen due to negligence and failure to fix the issues before deployment.

- **withdraw**(address tournamentAddress): This function allows players to withdraw their winnings from a tournament. The player must first check if the tournament has ended and then they can call this function to claim their winnings. The function will transfer the winnings to the player's wallet.

- **transferReferralAmount**(address referrer): This function allows players to transfer their referral winnings to their wallet. The player must have a minimum amount of winnings and the tournament must have ended for them to transfer their referral winnings.

- **addReferral**(address player): This function allows a player to add another player as their referral. The new player will receive a discount when they register for a tournament, and the referrer will receive a portion of their winnings.

- **canUseReferral**(address player): This function returns true if the player has a referral and the tournament has not yet started, otherwise it returns false.

- **getPlayerDetails**(address player): This function returns the details of a player including their wallet, username, and a list of tournaments they have registered for.

- **getTournamentDetails**(address tournamentAddress): This function returns the details of a tournament including its title, participation fee, end time, and prizes.

- **getTournaments():** This function returns a list of all the tournaments managed by the TournamentManager contract.

This is a public report. The security threats mentioned in this audit report are critical. Armur A.I is not responsible for security hacks that may happen due to negligence and failure to fix the issues before deployment.

Contract Variables

- **tournaments** - Enumerable set to store the addresses of all the tournaments created.
- **players** - Enumerable set to store the addresses of all registered players.
- **playerData** - Mapping to store the information of all registered players.
- **SAFETY_INTERVAL** - Constant to set a safety interval for player registration.
- **REFERRAL_MIN_WITHDRAWAL** - Constant to set the minimum withdrawal amount for referrals.

Deployment and Usage

The contract can be deployed on a compatible blockchain network like Ethereum. The contract owner must set a trusted forwarder for the contract to receive funds. Players can interact with the contract to register for tournaments, use referrals, add referrals, and withdraw referral rewards.

Function Restrictions

The register function allows players to register for a tournament. This function has some restrictions and requirements that need to be met before the player can register.

- onlyOfficial(tournamentAddress): The tournament must be an official tournament, as verified by the tournaments set.

This is a public report. The security threats mentioned in this audit report are critical. Armur A.I is not responsible for security hacks that may happen due to negligence and failure to fix the issues before deployment.

- The tournament must still be ongoing, as checked by calling Tournament(tournamentAddress).isOngoing().

- If the player has a referral, the participation fee must be (8 * tournamentDetails.participationFee) / 10. If the player does not have a referral, the participation fee must be equal to tournamentDetails.participationFee.

- If these conditions are met, the function delegates the player's registration to the tournament contract by calling Tournament(tournamentAddress).register().

- If the player has a referral, the referral reward is calculated as (4 * tournamentDetails.participationFee) / 10 and sent to the referrer using _asyncTransfer(referrer, referrerReward). The ReferralUsed event is also emitted.

The withdrawReferral function allows players to withdraw their referral rewards if they have earned more than REFERRAL_MIN_WITHDRAWAL ETH. The function has some restrictions and requirements that need to be met before a player can withdraw their referral rewards.

- onlyRegistered(_msgSender()): The player must be a registered player, as verified by the players set.

If these conditions are met, the player's referral rewards are transferred to their wallet using _asyncTransfer(wallet, referralRewards)

Referral System

This is a public report. The security threats mentioned in this audit report are critical. Armur A.I is not responsible for security hacks that may happen due to negligence and failure to fix the issues before deployment.

- The contract implements a referral system that allows players to receive discounts on their tournament participation fees by referring other players.

- A player's referral information is stored in the playerData mapping, and the referrer field stores the address of the player's referrer.

- The canUseReferral function returns a boolean indicating whether the player has a referrer and has not used their referral before.

- The addReferral function allows players to set their referrer. The function has some restrictions and requirements that need to be met before a player can set their referrer.

- onlyRegistered(_msgSender()): The player must be a registered player, as verified by the players set.

- The player's referrer must also be a registered player, as verified by the players set.

- If these conditions are met, the player's referrer is set in the playerData mapping and the ReferralAdded event is emitted.

Considerations -

A few things to consider for improvement:

- **Error handling**: Some parts of the code can throw exceptions without providing an error message, for example, the require statements in register function. Providing meaningful error messages can improve the user experience and debugging.

- **Trust in Trusted Forwarder**: The contract relies on a trusted forwarder to handle incoming payments. This design choice could

This is a public report. The security threats mentioned in this audit report are critical. Armur A.I is not responsible for security hacks that may happen due to negligence and failure to fix the issues before deployment.

pose security risks if the forwarder is compromised. Consider using a different mechanism, such as a smart contract, to handle incoming payments.

- **Updating tournament details:** There is no way to update the details of a tournament after it is created, such as the title, prizes, etc. It would be useful to have the functionality to update these details.

- **Testing:** The code should be thoroughly tested before deployment to ensure it meets the intended requirements and to catch any potential bugs or security vulnerabilities.

- **Code efficiency:** Some parts of the code can be optimized for efficiency, such as the use of mappings and structs, to reduce gas costs.

- One thing that can be improved is that the contract doesn't check the tournamentAddress parameter passed to register to ensure it is a valid contract that implements the Tournament interface. This can lead to potential security issues. It's better to check if the contract at tournamentAddress is a valid Tournament contract before calling its methods.

- Another thing to consider is the hardcoded value SAFETY_INTERVAL for the safety interval for tournament registrations. It might be better to make this value configurable.

- The contract uses a mapping to store player data, which can quickly become very expensive for large numbers of players. An alternative would be to use a more memory-efficient data structure, such as a dynamic array.

- The onlyRegistered modifier is used to check if a player is registered. However, it only checks if the player is present in the players set, but

This is a public report. The security threats mentioned in this audit report are critical. Armur A.I is not responsible for security hacks that may happen due to negligence and failure to fix the issues before deployment.

doesn't check if the playerData mapping contains the player's information. This might lead to inconsistencies in the player data.

Security Threats To Fix Urgently -

- **Reentrancy attack:** A reentrant attack could be launched on this contract if a malicious contract calls the register function and then calls the contract again before the first call is complete. This could result in the malicious contract being able to repeatedly register and drain the contract's balance.

- **Unchecked Ether transfer:** The contract has a fallback function that allows anyone to send ether to the contract without any restrictions. This increases the risk of the contract's balance being drained by an attacker.

- **Unrestricted tournament creation:** The createTournament function is only restricted to the contract owner, which means that anyone who has access to the contract owner's private key can create tournaments and potentially drain the contract's balance by charging high participation fees.

- **Unrestricted tournament participation:** The register function does not have any checks in place to restrict the number of times a player can participate in a tournament. This could result in a player repeatedly participating in a tournament and draining the contract's balance.

- **Inadequate input validation:** The register function does not validate the input parameters, such as the tournament address, properly, which increases the risk of the contract's state being manipulated by an attacker.

This is a public report. The security threats mentioned in this audit report are critical. Armur A.I is not responsible for security hacks that may happen due to negligence and failure to fix the issues before deployment.

- **Inadequate error handling:** The contract does not have proper error handling mechanisms in place, which increases the risk of the contract's state being corrupted in case of errors or unexpected conditions.

This is a public report. The security threats mentioned in this audit report are critical. Armur A.I is not responsible for security hacks that may happen due to negligence and failure to fix the issues before deployment.