Final Report - Roulette

Johannes Garstenauer, Ece Sevenay, Niklas Otto, Bilgehan Cagiltay, Aysegül Rana Erdemli, Johann Sebastian Schicho

See “Roles” on the last page for work distribution

> “Well, what, what new thing can they say to me that I don't know myself? And is that the point? The point here is that--one turn of the wheel, and everything changes, and these same moralizers will be the first (I'm sure of it) to come with friendly jokes to congratulate me. And they won't all turn away from me as they do now. Spit on them all! What am I now?
>
> *Zéro.*
>
> What may I be tomorrow? Tomorrow I may rise from the dead and begin to live anew! I may find the man in me before he's lost!”
>
> **— Fyodor Dostoyevsky, in *The Gambler***

1. Introduction

This report explains the high level details of an online Roulette application whose internal structure is developed as a smart contract written in Solidity where also the application’s security is ensured. The application aims to provide a Roulette game where there will be just one owner of the contract, who will control the game and players who will not be able to manipulate the randomness of the game even though some might be malicious. Since generating randomness on the blockchain is hard, a third-party service is used. Chainlink VRF v2 provides provably fair and tamper-proof randomness, using a verifiable random function (VRF). Some of the main variables defined by the smart contract for this game are: Chainlink coordinator, the casino's

(owner) address, and the minimum and maximum bet amounts. It also includes structs for storing information about past and current bets, and events for logging various actions within the contract. The contract defines various functions that enable it to place a bet, to determine a winner after all the bets are made, and to set the minimum and maximum bet amounts. Additionally, it has a non-reentrancy modifier to prevent re-entrancy in the external functions.

2. Motivation

Online casino games, like Roulette, enjoy growing popularity. Contrary to physical casinos, they have low entry barriers. Players do not need to dress up to enter them. They can easily play from home while enjoying food and drinks. The minimum stakes required in online casinos are usually lower than in physical ones. There are almost no employees to be paid because software handles all the transactions. Additionally, gambling companies don't have to buy or rent fancy buildings for casinos. Therefore, online casinos have lots of advantages for players and service providers.

Nonetheless, there is an issue with online casinos. Usually, providers will ask you to deposit some money, your gambling stake, on their website. This means you must give up the ownership of your money to a trusted third party who will manage your money. Numerous scandals, scams, and hacking disasters speak volumes about the safety of this approach.

Decentralized blockchains offer a solution to this problem of trust. There is no need for a trusted third party anymore. The gambling process will be handled on the blockchain, utilizing Smart Contracts, which is a safer way of gambling. You can easily connect your wallet e.g., Metamask, gamble, and afterward get paid out immediately and transparently.

In summary, we can conclude that the blockchain offers a great use case for online gambling being more secure and transparent compared to traditional online gambling websites. Therefore, we are convinced of the usefulness of a Roulette DApp.

3. Literature Review

Decentralized gambling applications involve the wagering of cryptocurrencies, which means that when individuals place wagers using these applications, their transactions are recorded on blockchains. There exists studies on decentralized gambling applications, yet they lack profound discussion on data transparency and public availability. Taking into consideration that these applications run on blockchain, one can expect a difference between player behavior in traditional online casinos and player behavior in decentralized casinos.

A research conducted by Oliver J. Scholten (2020) focuses on the profile of gamblers while considering the effect of non-human players (bots). The paper explores 2,232,741 transactions from 24,234 unique addresses to three such applications operating on Ethereum network. The results display that a typical player in these DApps bets in a non-committal and non-intense way and the assumption that this typical player would play for one short session on a single application and then cease play or move to another application could be made. Findings of this research also suggest that the average decentralized gambling application player spends less than in other online casinos overall, but that the most heavily involved players in this new domain spend substantially more.

Furthermore, the longer an individual uses a decentralized gambling application, the more bets they will place and their total amount wagered will become greater. When a player places bets more frequently, they are likely to place more bets over their gambling career. Although we could assume that larger amounts of wagering would result in higher losses, this was not the case in these applications. Since we mentioned bots, we might as well discuss what kind of effect they can have on the system. Bots may artificially inflate the perceived popularity of the applications they are transacting with, or to attempt to win the jackpot from an application once it becomes statistically worthwhile to pursue; which creates the problem of illegal access to the service. To combat this problem, several location proof strategies have been proposed. One of them was developing a decentralized Proof-of-Location (PoL) system tailored to blockchain applications for energy trading. This solution ensures that automated transactions are issued by the right nodes by using smart contract-based random selection and a game-theoretic scenario suitable for blockchain energy trading, which deals with the concern of confidentiality (Merrad et. al., 2022).

Lastly, an existing app can be presented as an example to our application. A French roulette game is played on Our-Roulette application. This platform is decentralized and it also lets the users invest into the project in return for some passive income.

4. The Problem of Randomness

Randomness on the blockchain is a hard problem and when not applied properly, it might cause a smart contract to be vulnerable to attacks. Generating the number by deriving it from a block hash solution is vulnerable. Off-chain API's that will handle the randomness process off the blockchain where every information is public are a better solution when the third party provider is trusted. In this project, Chainlink VRF, which is fair, verifiable by the proof of cryptographic randomness that it publishes on chain and available on goerli testnet is used as the Off-chain API. VRFCoordinatorV2Interface is the specific interface that was used.

VRFCoordinatorV2Interface coordinates the generation and verification of verifiable random functions (VRFs) on the Ethereum blockchain. VRF is a cryptographic algorithm that generates a random output (a "proof") based on a secret input (a "seed"). The smart contract uses this proof to determine the outcome of a random event, such as the Roulette game's result.

The VRFCoordinatorV2Interface ensures that the random number generation is verifiable, meaning that the smart contract can check that the proof was generated correctly and that the seed was not tampered with. It also ensures that the seed is secret, meaning that only the person who generated the seed can know it. This is accomplished through the use of a cryptographic technique called a "non-interactive zero-knowledge proof," which allows the smart contract to verify the proof without knowing the seed.

The VRFCoordinatorV2Interface also ensures that the random number generated is unpredictable, meaning that it is computationally infeasible for anyone to predict the outcome of

a random event before it happens. This is accomplished through the use of a cryptographic algorithm called "VRF" which ensures that the output produced is unpredictable and can only be computed by the possessor of the private key.

5. The Game

The contract offers a single function to place and make a bet. Roulette is a luck based game with many different variations and options to make a bet. Therefore we implement a limited number of the most common options to place a bet in our game.

The options to place a bet are as follows. (1) The player can bet on even or odd numbers. A winning bet will be paid back to the player 1 to 1 in the amount of the initial bet. (2) The player can bet on the red or black fields. A winning bet will also be paid back 1 to 1. (3) The player can bet on a Dozen. There are three dozens, namely the numbers from 1-12, 13-24, 25-36. A winning bet will be paid back 2 to 1. (4) Finally, there is the option to bet on a single number. A winning bet will be paid back 35 to 1.

6. Roulette Contract: A Summary

Roulette is a smart contract that implements a simple roulette game using the VRFCoordinatorV2Interface and VRFConsumerBaseV2. The contract utilizes the VRF algorithm to generate a random number that determines the outcome of the game.

The contract has a struct, "Bet" that holds the player's address, the bet amount, bet type, bet number, and a boolean indicating if the player wins or not. The contract also has a mapping "book" that holds the history of games, and the key is the request ID.

The contract has an event, "LogResult" that records the player's address, message that says the player has won or not, and amount that is won by the player at the end of the game. It is used to log the result of the game for a single player. There are some other events like; "LogMinMax" that records the minimum and maximum bet amount allowed, "LogPayout" that records the payout message, "LogCurrentBet" that records the current bet, and "LogGameEnded" that records the end of the game message.

The contract has a constructor that takes in the subscription ID, and it sets the COORDINATOR and the s_subscriptionId variable. This subscription has to be made on the Chainlink website and funded with their Link tokens. The Coordinator helps with the random number generation.

There is also a private variable, "lockedPlacingBet" that is used as a re-entrancy guard.

The contract has a function, "setMinMaxBets" that sets the current minimum and maximum bet amounts allowed (in Wei). The function calculates the minimum and maximum bet amount based on the contract's balance.

Another function, "placeBet", allows a player to place a bet. The function takes in the bet type, bet number, and the bet amount. The function also checks if the bet amount is within the minimum and maximum bet amounts allowed, and also prevents multiple bets from being placed simultaneously.

"fullfillRandomNumber" function is called by the Chainlink oracle and it sets the current winning number. It also updates the current bet, and it calls the "payoutWinner" function to determine the winner.

The contract has another function, "payoutWinner" that checks if the player wins or loses, and it also updates the player's balance. The function also emits events to log the result, payout, and the end of the game.

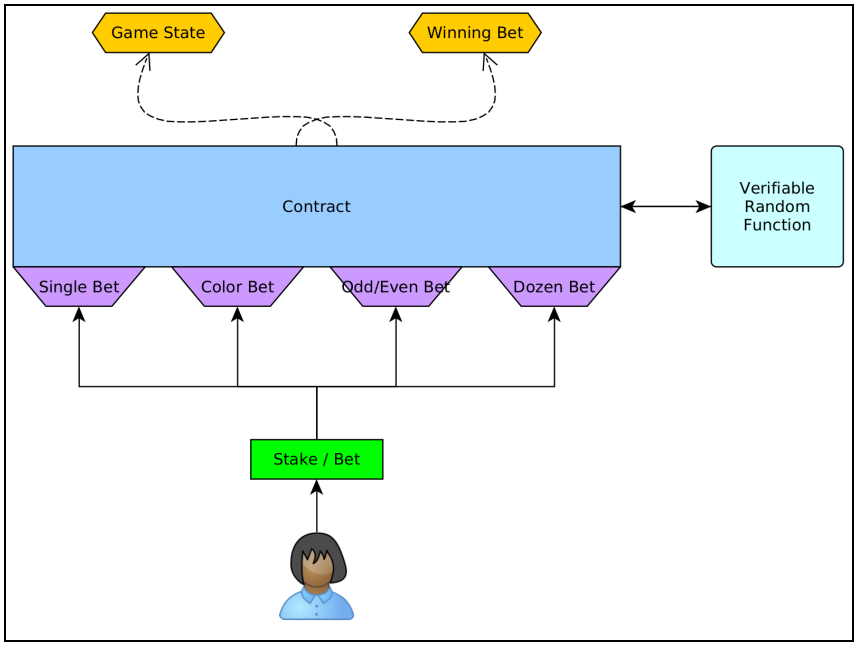

Figure 1: The Game

7. Security Analysis

7.1. Reentrancy Attack

A reentrancy attack is a type of security vulnerability that occurs when a smart contract calls an external contract and that external contract, in turn, calls back into the original contract. This can lead to an infinite loop, and in some cases, allow an attacker to drain the contract of its funds.

The Roulette contract includes a mechanism to prevent reentrancy attacks by using a boolean variable called "lockedPlacingBet" and a modifier called "nonReentrant". The "nonReentrant" modifier checks the value of "lockedPlacingBet" before executing the function. If "lockedPlacingBet" is true, the function will not execute and will revert with the error message "No re-entrancy". In practice, this modifier is applied to functions that perform critical operations like placeBet() and resolveBet().

The lockedPlacingBet variable is set to true at the beginning of the execution of the function, and set back to false at the end of the execution, thus ensuring that the function can only be executed once at a time, and preventing the possibility of reentrancy, thus providing a secure environment for players to interact with the contract and place their bets.

7.2. Integer Overflow-Underflow

A way to prevent integer overflow attacks is to use the new Solidity's require function with the assert function to explicitly check the values of integers before performing arithmetic operations.

```
// A maximum amount should not be exceeded when betting,
// so that the contract always remains able to pay out the full potential winnings.
require((_balance / 35) > minBet, "The funds in this contract are too low and need to be higher.");
maxBet = (_balance / 35) - minBet;
emit LogMinMax(maxBet, minBet);

assert(maxBet >= minBet);
```

Figure 2.

It can be seen in figure 2. that this practice is applied in Roulette. This way if an integer overflow or underflow occurs, the require will cause the transaction to be reverted and the assert will cause the program to throw an error.

7.3. Best Practices

The Roulette contract uses some best practices for smart contract development, including:

1. Use of events for Logging: The contract uses events to log important information, such as the outcome of a bet, the current bet, and the game status. This allows for easy tracking of the contract's state and can be used for auditing purposes and detecting intrusions.
2. Simplicity: By using structs, to group related variables together, splitting business logics into small functions, providing extensive documentation etc. the codebase is readable and easily understood. This decreases the likelihood of bugs and fosters understanding and therefore allows a focus on security aspects.
3. Access Control and Marking Visibility: The contract uses the require function for access control, employs the payable-keyword and explicitly marks the visibility of functions. This minimizes risks, by preventing bad actors from being able to access more endpoints than necessary or altering the programs state in unexpected ways.

7.4. Code Analysis

We employed code analyzers to test our code against common vulnerabilities.

We have employed Mythril (https://github.com/ConsenSys/mythril), which is a dynamic EVM bytecode analyzer. Mythril did not detect any vulnerabilities. Furthermore, we have used Echidna (https://github.com/crytic/echidna) to fuzz our contract against invariants. In multiple runs with over 80.000 transactions each no breakage against the invariants was detected. Additionally, Echidna provided us with a code coverage of over 90% in the fuzzing. Providing in-depth analysis and trust for the contract implementation.

The Web Interface

To play the game we implement a web interface. It will work in conjunction with MetaMask to allow payments.

The web interface allows placing a bet according to the options mentioned in the contract section of this document. The place can then be executed through a button press and the payment towards the contract can be confirmed in MetaMask.

The result of the game will be displayed once the contract has been executed.

We also define goals for the Web Interface, which we may implement given the amount of time available.

- Display a history of the results of games
- Display the currents game status and stakes

Conclusion

In summary, the Roulette smart contract is an implementation of a roulette game that utilizes a trustless third-party VRF algorithm to generate a random number that determines the outcome of the game.

Roles

1. Web-App Team
    1. Truffle Framework Expert (interact with Blockchain and Metamask)
    2. Design and implement the web-app in a Javascript Framework (compatible with Truffle)
2. Solidity Team
    1. Chainlink VRF Expert - Johannes Garstenauer
    2. Contract
        1. Roulette business logic - Johann Schicho, Johannes Garstenauer
        2. Interface - Johann Schicho, Johannes Garstenauer
        3. Events - Aysegul Rana Erdemli
    3. Security Experts: Johannes Garstenauer, Aysegul Rana Erdemli
    4. (Security) Code Analysis: Johann Schicho

References:

Chainlink. (2021). *VRF v2 Introduction.* https://docs.chain.link/vrf/v2/introduction.

Scholten, Oliver & Zendle, David & Walker, James. (2020). *Inside the Decentralised Casino : A Longitudinal Study of Actual Cryptocurrency Gambling Transactions*. PLOS ONE. 15. e0240693. 10.1371/journal.pone.0240693.

Merrad Y, Habaebi MH, Islam MR, Gunawan TS, Mesri M. (2022) *Robust Decentralized Proof of Location for Blockchain Energy Applications Using Game Theory and Random Selection.* Sustainability.; 14(10):6123. https://doi.org/10.3390/su14106123