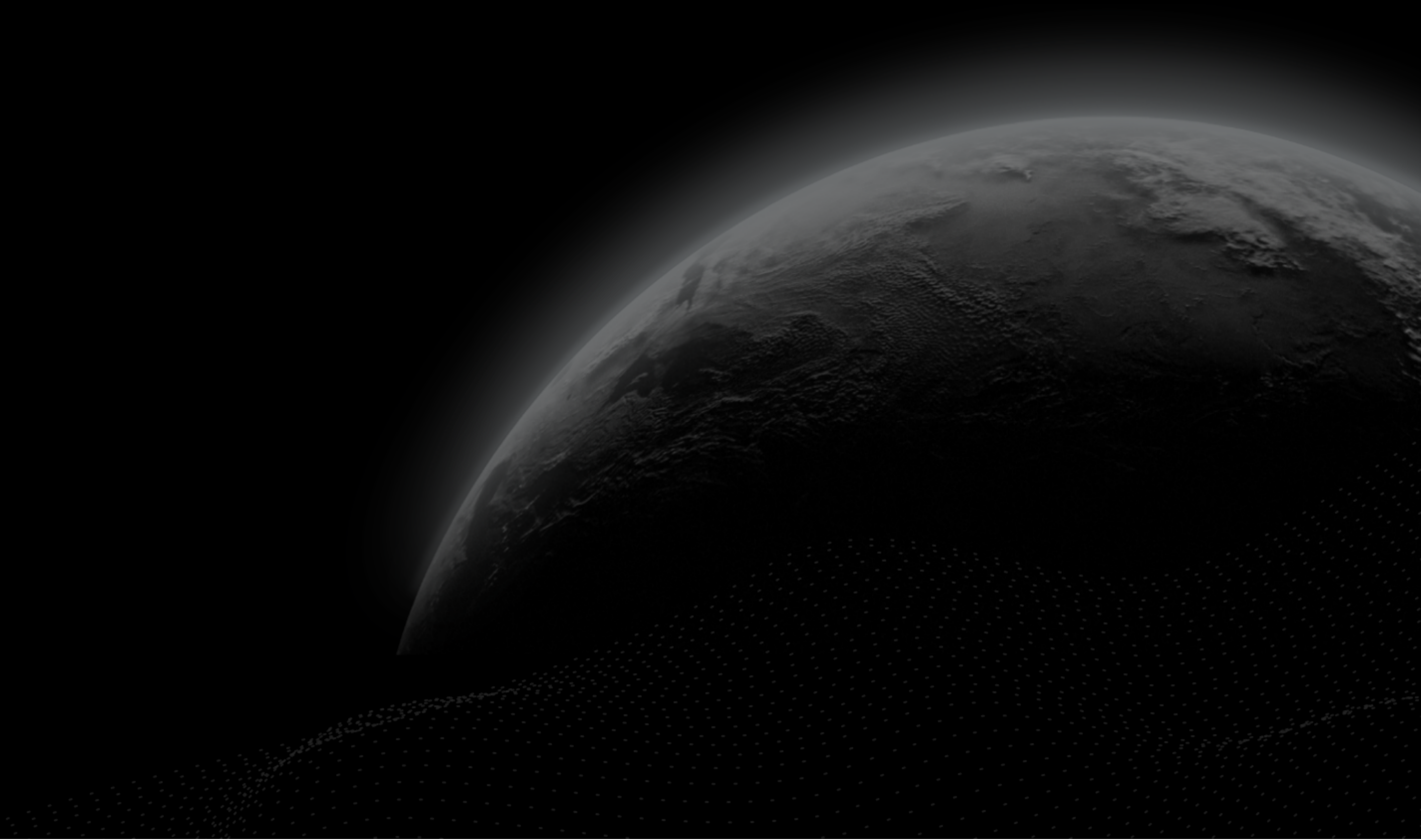
CERTIK

Security Assessment

Playnance - Audit 2

CertiK Assessed on Jun 19th, 2024



CertiK Assessed on Jun 19th, 2024

Playnance - Audit 2

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

Executive Summary

TYPES	ECOSYSTEM	METHODS
DeFi	EVM Compatible	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 06/19/2024	N/A

CODEBASE

https://github.com/playnance-games/PlayBlock

View All in Codebase Page

COMMITS

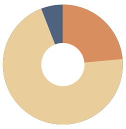
- d54cce2f3963b755ad2c63fa9853b8f9454bb0ae
- d24b9fe0043ea6bacb5d943e047690742bfe5153

View All in Codebase Page

Highlighted Centralization Risks

- ⓘ Privileged role can mint tokens
- ⓘ Transfers can be paused
- ⓘ Fees are bounded by 100%
- ⓘ Has blacklist/whitelist

Vulnerability Summary



17	4	0	0	13	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.
■ 4	Major	4 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.
■ 0	Medium		Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
■ 12	Minor	4 Resolved, 8 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.
■ 1	Informational	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 | PLAYNANCE - AUDIT 2

CODEBASE | PLAYNANCE - AUDIT 2

Repository

https://github.com/playnance-games/PlayBlock

Commit

- d54cce2f3963b755ad2c63fa9853b8f9454bb0ae
- d24b9fe0043ea6bacb5d943e047690742bfe5153

AUDIT SCOPE | PLAYNANCE - AUDIT 2

2 files audited ● 2 files without findings

ID	Repo	File	SHA256 Checksum
● GTB	playnance-games/PlayBlock	usd.sol	5d2f059a28ec588dfd6e55a66c849fd86d6fc030770f1d1c0080026330fc47dd
● UVD	playnance-games/PlayBlock	UpVsDownGameV8.sol	66ada63f63c3c7aef548ec5d0360e811a0b22dd4e5fc6cdef03eaf77d13b7cf2

APPROACH & METHODS | PLAYNANCE - AUDIT 2

This report has been prepared for Playnance to discover issues and vulnerabilities in the source code of the Playnance - Audit 2 project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Formal Verification, Manual Review, and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

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

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

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

REVIEW NOTES | PLAYNANCE - AUDIT 2

Overview

This audit concerns a stable coin contract (USDP) and a game contract (UpVsDownGame) for Playnance.

USDP

The USDP token is meant to be a stable coin pegged to the US dollar. This is a token where transfers are taxed by at most 10%, unless either the sender or recipient are whitelisted.

UpVsDownGame

This game is one where bettors bet on whether the price of Bitcoin will go down or up at some point in the future. In the event of a tie, all bettors recover their original bet. Otherwise, the winning bettors receive a share of the losing bettors' bets. These winnings are taxed, by up to 200%.

External Dependencies

This project inherits or uses a few 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 following values are initialized either at deploy time or by specific functions in smart contracts.

- **USDP**: `tokenAddress`
- **UpVsDownGame**: `token`

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

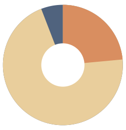
Privileged Functions

In this project, privileged roles are adopted to ensure the dynamic runtime updates of the project, which are specified in the following finding: `Centralization Related Risks`.

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 | PLAYNANCE - AUDIT 2



17	0	4	0	12	1
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Playnance - Audit 2. Through this audit, we have uncovered 17 issues ranging from different severity levels. Utilizing the techniques of Formal Verification, Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
GTP-01	**Centralized Balance Manipulation**	**Centralization**	**Major**	● **Acknowledged**
GTP-02	Any User Can Burn Tokens	Design Issue	Major	● Acknowledged
PBB-01	**Centralization Related Risks**	**Centralization**	**Major**	● **Acknowledged**
UVD-01	Bets May Be Taken From Other Pools	Logical Issue	Major	● Acknowledged
GTP-03	Not Compliant With The EIP20 Standard	Design Issue	Minor	● Acknowledged
GTP-04	Mints And Burns Are Taxed	Logical Issue	Minor	● Acknowledged
GTP-05	Incorrect Upper Bound On Tax Rate	Inconsistency	Minor	● Acknowledged
PBB-02	Missing Zero Address Validation	Volatile Code	Minor	● Acknowledged
UVD-02	Unchecked ERC-20 `transfer()` / `transferFrom()` Call	Volatile Code	Minor	● Acknowledged
UVD-03	Potential Reentrancy Attack (Sending Tokens)	Concurrency	Minor	● Acknowledged
UVD-04	Potential Divide By Zero	Logical Issue	Minor	● Acknowledged

ID	Title	Category	Severity	Status
UVD-05	Locked Blockchain Native Tokens	Inconsistency	Minor	● Resolved
UVD-06	Use Of Old Compiler Versions	Language Version	Minor	● Resolved
UVD-07	Fees Can Exceed 100%	Logical Issue	Minor	● Resolved
UVD-08	Potential Denial-Of-Service Attack	Design Issue	Minor	● Acknowledged
UVD-09	Insufficient Check For EOA	Logical Issue	Minor	● Resolved
UVG-11	Missing Emit Events	Coding Style	Informational	● Acknowledged

GTP-01 | CENTRALIZED BALANCE MANIPULATION

Category	Severity	Location	Status
Centralization	● **Major**	**usd.sol (005bd6b): 27**	● **Acknowledged**

Description

In the contract `USDP` , the role `owner` has the authority to mint tokens to an arbitrary account without any restrictions.

Any compromise to the `owner` account may allow a hacker to take advantage of this authority and manipulate users' balances. Since `USDP` is meant to be a stable coin, a hacker with access to the `owner` account would be able to acquire a large number of `USDP` stable coins or greatly manipulate the price of `USDP` .

Recommendation

We recommend the team makes efforts to restrict access to the private key of the privileged account. A strategy of 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 mint more tokens or engage in similar balance-related operations.
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 multi signature (⅔, ⅗) wallet *mitigate* the risk by avoiding a single point of key management failure.

- 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 **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 DAO for controlling the operation *mitigate* the risk by applying transparency and decentralization.

- Introduction of a DAO, governance, or voting module to increase decentralization, transparency, and user involvement;
  AND
- A medium/blog link for sharing the multi-signers' addresses, and DAO information with the community.

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

- 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:

The following actions can *fully* resolve the risk:

- Renounce the ownership and never claim back the privileged role.
  OR
- Remove the risky functionality.
  OR
- Add minting logic (such as a vesting schedule) to the contract instead of allowing the owner account to call the sensitive function directly.

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

[Playnance Team, June 19, 2024]: Issue acknowledged. I will fix the issue in the future, which will not be included in this audit engagement.

The owner's private key is stored only in cold storage. We minted 1B tokens and there are no plans for minting more tokens in the future. In the future we will use multisig wallet for managing the tokens.

[CertiK, June 19, 2024]: While this strategy will reduce the risk, it is crucial to note that it will not completely eliminate it. CertiK strongly encourages the project team to periodically revisit the private key security management of all centralized roles and addresses.

GTP-02 | ANY USER CAN BURN TOKENS

Category	Severity	Location	Status
Design Issue	● Major	usd.sol (005bd6b): 11	● Acknowledged

Description

The USDP tokens is meant to be a stable coin and should only be burnable by the contract's owner, as seen from the way that the `ERC20Burnable.burn()` function is overwritten:

```
    function burn(uint256 amount) public override onlyOwner {
        super._burn(owner(), amount);
    }
```

However, the function `ERC20Burnable.burnFrom()` is not overwritten, allowing any user to burn their own tokens by first giving themself an allowance and then calling `burnFrom()`.

As the USDP token is meant to be a stable coin, its supply is incredibly important for its tokenomics. Allowing arbitrary users to burn the stable coin causes the token to be at risk of depegging.

Proof of Concept

The following test written using foundry demonstrates that any user is able to burn their tokens:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {Test, console} from "forge-std/Test.sol";
import {USDP} from "../src/usd.sol";

contract USDPTest is Test {
    USDP usdp;
    address owner = vm.addr(1);
    address notOwner = vm.addr(2);
    address taxWallet = vm.addr(3);

    function setUp() public {
        usdp = new USDP(owner, taxWallet);
    }

    function testNonOwnerCanBurn() public {
        assert(owner != notOwner);

        // Mint tokens to notOwner
        vm.prank(owner);
        usdp.mint(notOwner, 100);

        // notOwner cannot burn directly
        uint256 beforeBalance = usdp.balanceOf(notOwner);
        vm.expectRevert();
        usdp.burn(100);
        assert(beforeBalance == usdp.balanceOf(notOwner));

        // notOwner can burn using burnFrom
        beforeBalance = usdp.balanceOf(notOwner);
        assert(beforeBalance != 0);

        vm.startPrank(notOwner);
        usdp.approve(notOwner, beforeBalance);
        usdp.burnFrom(notOwner, beforeBalance);
        vm.stopPrank();

        assert(usdp.balanceOf(notOwner) == 0);
    }
}
```

Recommendation

It is recommended to overwrite the `ERC20Burnable.burnFrom()` function to prevent any user from burning the token.

Alleviation

[Playnance Team, June 19, 2024]: Issue acknowledged. I won't make any changes for the current version.

[CertiK, June 19, 2024]: It should be noted that the `burn()` function has the `onlyOwner` modifier, only allowing the owner to burn tokens, while `burnFrom()` has no such restriction.

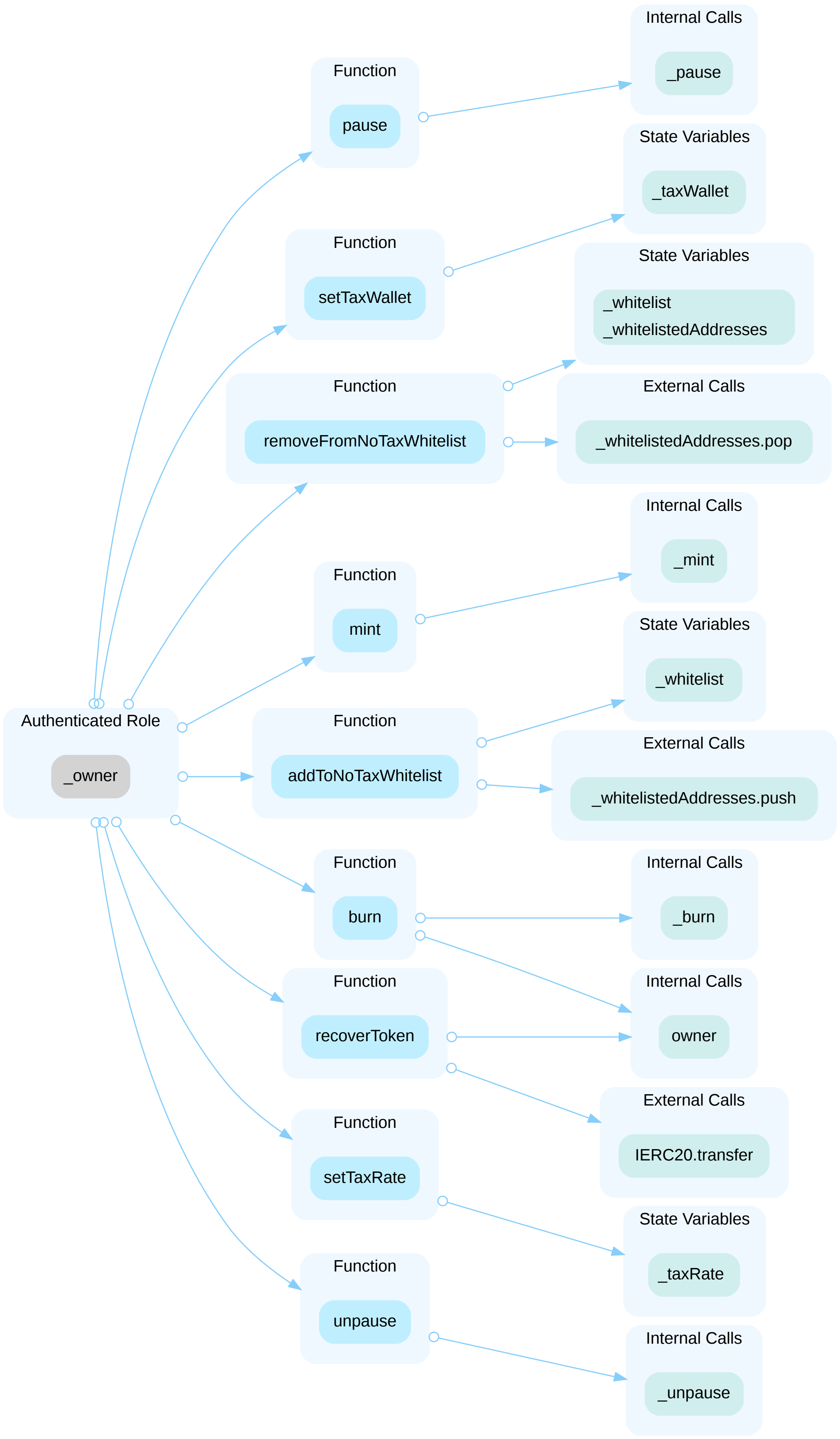
In addition, documentation for USDP at this link states that burning is vital for regulating the token supply, suggesting users burning tokens may be an issue.

PBB-01 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization	● **Major**	**usd.sol (005bd6b): 27, 31, 35, 39, 58, 64, 73, 82, 88; UpVsDownGameV8.sol (d54cce2): 158, 169, 175, 186, 195, 206, 212, 218, 230, 310, 595, 602**	● **Acknowledged**

Description

In the contract `USDP` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and mint stable coins to themself, take taxes for themself, or greatly increase taxes.


Internal Calls
_pause
Function
pause
State Variables
_taxWallet
Function
setTaxWallet
State Variables
_whitelist
_whitelistedAddresses
Function
removeFromNoTaxWhitelist
External Calls
_whitelistedAddresses.pop
Internal Calls
_mint
Function
mint
State Variables
_whitelist
Authenticated Role
_owner
Function
addToNoTaxWhitelist
External Calls
_whitelistedAddresses.push
Function
burn
Internal Calls
_burn
Function
recoverToken
Internal Calls
owner
External Calls
IERC20.transfer
Function
setTaxRate
State Variables
_taxRate
Function
unpause
Internal Calls
_unpause

In the contract `UpVsDownGameV8` , the role `_owner` has authority over the following functions:

- `changeGameControllerAddress` : changes the address of the `gameController` role
- `changeGameFeePercentage` : changes the fee rate
- `changeGameFeeJackpotPercentage` : changes the jackpot fee rate
- `changeGameFeeAddress` : changes the address that fees are sent to
- `changeGameFeeJackpotAddress` : changes the address that jackpot fees are sent to
- `stopGame` : stops the game from running, preventing the start of rounds and bets
- `startGame` : starts the game, allowing the start of rounds and bets
- `allowContract` : whitelists a contract to make bets
- `removeContract` : removes a contract from the whitelist

Additionally, the role `gameController` has authority over the following functions:

- `createPool` : starts a new pool
- `trigger` : starts or ends a round
- `distribute` : returns bets or transfers bet winnings

Any compromise to either of the above roles allows a hacker to disrupt the game and decide how a game will end.

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.

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

[Playnance Team, June 19, 2024]: Issue acknowledged. I will fix the issue in the future, which will not be included in this audit engagement. We will swicth to multisig wallet in the future.

[CertiK, June 19, 2024]: While this strategy will reduce the risk, it is crucial to note that it will not completely eliminate it. CertiK strongly encourages the project team to periodically revisit the private key security management of all centralized roles and addresses.

UVD-01 | BETS MAY BE TAKEN FROM OTHER POOLS

Category	Severity	Location	Status
Logical Issue	● Major	UpVsDownGameV8.sol (d54cce2): 431	● Acknowledged

Description

When distributing winnings, winnings are distributed to all winners and then the fee and remaining tokens in the contract are distributed to the fee address. In the case of multiple pools, these remaining tokens may consist of bets made to other pools.

```
420         if (winners.distributedCount == winners.bets.length) {
421             sendToken(
422                 feeAddress,
423                 ((dist.fee + dist.totalMinusFee) * feePercentage) / 100
424             );
425             sendToken(
426                 feeJackpotAddress,
427                 ((dist.feeJackpot + dist.totalMinusJackpotFee) *
428                     feeJackpotPercentage) / 100
429             );
430             //Send leftovers to fee address
431             sendToken(feeAddress, getContractTokenBalance());
```

As such, these leftover tokens that were supposed to be used as bets for another pool are stolen by the fee address.

Proof of Concept

The following proof-of-concept written in foundry shows that when there are two pools, the bets made in one pool can be sent to the fee address if the other pool distributes winnings first.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {Test, console} from "forge-std/Test.sol";
import {UpVsDownGameV8} from "../src/UpVsDownGameV8.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract USDPTest is Test {
    UpVsDownGameV8 game;
    address gameController = vm.addr(1);
    MockERC20 token;
    address feeAddr = vm.addr(2);

    bytes poolOne = new bytes(1);
    bytes poolTwo = new bytes(2);

    address bettor = vm.addr(3);

    function setUp() public {
        token = new MockERC20("Test", "Test");
        game = new UpVsDownGameV8(gameController, address(token));

        vm.startPrank(gameController);
        game.changeGameFeeAddress(feeAddr);
        game.changeGameFeeJackpotAddress(feeAddr);
        game.startGame();
        vm.stopPrank();
    }

    function testBetsTakenFromOtherPools() public {
        // make fees zero to easily see token flow
        vm.startPrank(gameController);
        game.changeGameFeePercentage(0);
        game.changeGameFeeJackpotPercentage(0);
        vm.stopPrank();

        // create two pools
        vm.startPrank(gameController);
        game.createPool(poolOne, 0, 1 ether, 10);
        game.createPool(poolTwo, 0, 1 ether, 10);
        game.trigger(
            poolOne,
            0,
            block.timestamp,
            block.timestamp + 1 days,
            0,
            10,
            0
        );
        game.trigger(
            poolTwo,
            0,
            block.timestamp,
            block.timestamp + 1 days,
            0,
            10,
            0
        );
        vm.stopPrank();

        // bet on both pools
        uint256 startingBalance = 500;
        token.mint(bettor, startingBalance);
        vm.startPrank(bettor);
        token.approve(address(game), startingBalance);

        UpVsDownGameV8.makeTradeStruct memory poolOneBetUp =
            UpVsDownGameV8.makeTradeStruct({
                poolId: poolOne,
                avatarUrl: "",
                countryCode: "",
                upOrDown: true,
                whiteLabelId: "",
                amount: 100
            });

        UpVsDownGameV8.makeTradeStruct memory poolOneBetDown =
            UpVsDownGameV8.makeTradeStruct({
                poolId: poolOne,
                avatarUrl: "",
                countryCode: "",
                upOrDown: false,
                whiteLabelId: "",
                amount: 150
            });

        UpVsDownGameV8.makeTradeStruct memory poolTwoBetUp =
            UpVsDownGameV8.makeTradeStruct({
                poolId: poolTwo,
                avatarUrl: "",
                countryCode: "",
                upOrDown: true,
                whiteLabelId: "",
                amount: 250
            });

        game.makeTrade(poolOneBetUp);
        game.makeTrade(poolOneBetDown);
        game.makeTrade(poolTwoBetUp);
        vm.stopPrank();
        assert(token.balanceOf(address(game)) == startingBalance);

        // complete poolOne
        vm.startPrank(gameController);
        game.trigger(
            poolOne,
            0,
            block.timestamp,
            block.timestamp + 1 days,
            100,
            10,
            0
        );
        game.trigger(
            poolOne,
            0,
            block.timestamp,
            block.timestamp + 1 days,
            200,
            10,
            0
        );

        // check balances
        // bets for poolTwo are gone
        assert(token.balanceOf(address(game)) == 0);

        // bettor acquires the 250 winnings from poolOne as expected
        assert(token.balanceOf(bettor) == 250);

        // fee address has a balance even though there should be no fees
        assert(token.balanceOf(feeAddr) == 250);
    }
}

contract MockERC20 is ERC20 {
    constructor(string memory _name, string memory _symbol) ERC20(_name, _symbol) {}

    function mint(address to, uint256 amount) external {
        _mint(to, amount);
    }
}
```

Recommendation

If only one pool is to ever exist, it is recommended to add checks to ensure only one pool can ever be running. Otherwise, it is recommended to not allow such a transfer of tokens.

Alleviation

[Playnance Team, June 19, 2024]: Issue acknowledged. I won't make any changes for the current version. It's not possible because we are running only one pool on each smart contract instance.

[CertiK, June 19, 2024]: While the issue will not exist if only one pool is run for each smart contract, there are no guarantees of this, meaning the risk still exists.

GTP-03 | NOT COMPLIANT WITH THE EIP20 STANDARD

Category	Severity	Location	Status
Design Issue	● Minor	usd.sol (005bd6b): 48	● Acknowledged

Description

When transferring tokens, a possible tax is taken on the amount transferred:

```
  46          if (!_whitelist[from] && !_whitelist[to] && _taxRate > 0) {
  47              taxAmount = (amount * _taxRate) / 10000;
  48              require(taxAmount > 0, "Tax amount too low");
  49              super._update(from, _taxWallet, taxAmount);
//Send the tax amount to tax wallet
  50          }
```

If the tax amount is 0, then the transfer fails. In particular, when transferring 0 tokens, the transfer will always fail as the tax amount will be 0 in this situation.

This is not compliant with the EIP20 standard, as transfers of 0 tokens are expected to succeed: https://eips.ethereum.org/EIPS/eip-20.

In addition, this creates an issue with interoperability. Smart contracts wishing to interact with the USDP contract would need to handle the situation when transferring 0 tokens, which is not the standard.

Recommendation

It is recommended to allow transfers of 0 tokens.

Alleviation

[Playnance Team, June 19, 2024]: Issue acknowledged. I won't make any changes for the current version.

GTP-04 | MINTS AND BURNS ARE TAXED

Category	Severity	Location	Status
Logical Issue	● Minor	usd.sol (005bd6b): 46	● Acknowledged

Description

The ERC20 `_update` function has been overwritten to apply taxes if the tax rate is positive.

```
 46         if (!_whitelist[from] && !_whitelist[to] && _taxRate > 0) {
 47             taxAmount = (amount * _taxRate) / 10000;
 48             require(taxAmount > 0, "Tax amount too low");
 49             super._update(from, _taxWallet, taxAmount);
//Send the tax amount to tax wallet
 50         }
```

In particular, mints and burns are by default taxed as the `_mint` and `_burn` functions use `_update`.

```
    function _mint(address account, uint256 value) internal {
        if (account == address(0)) {
            revert ERC20InvalidReceiver(address(0));
        }
        _update(address(0), account, value);
    }
```

```
    function _burn(address account, uint256 value) internal {
        if (account == address(0)) {
            revert ERC20InvalidSender(address(0));
        }
        _update(account, address(0), value);
    }
```

However, it does not make sense for these operations to be taxed.

Recommendation

It is recommended to include the zero address in the whitelist in the constructor to ensure that mints and burns are not taxed. The zero address should also not be allowed to be removed in `removeFromNoTaxWhitelist`.

Alleviation

[Playnance Team, June 19, 2024]: Issue acknowledged. I will fix the issue in the future, which will not be included in this audit engagement.

GTP-05 | INCORRECT UPPER BOUND ON TAX RATE

Category	Severity	Location	Status
Inconsistency	● Minor	usd.sol (005bd6b): 65	● Acknowledged

Description

The documentation and comments state that the maximum tax rate is 1%.

```
65          require(newTaxRate <= 1000, "Tax rate cannot exceed 1%");
```

However, the tax rate is out of 10000, meaning the maximum tax rate is 1000/10000 = 10%.

```
47              taxAmount = (amount * _taxRate) / 10000;
```

Recommendation

It is recommended to ensure the documentation aligns with the code implementation, either by changing the documentation to state the maximum tax rate is 10% or changing the `require` check to ensure that `newTaxRate` cannot exceed 100.

Alleviation

[Playnance Team, June 19, 2024]: Issue acknowledged. I won't make any changes for the current version.

PBB-02 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	● Minor	usd.sol (005bd6b): 23; UpVsDownGameV8.sol (d54cce2): 154	● Acknowledged

Description

Addresses are not validated before assignment or external calls, potentially allowing the use of zero addresses and leading to unexpected behavior or vulnerabilities. For example, transferring tokens to a zero address can result in a permanent loss of those tokens.

```
23          _taxWallet = taxWallet;
```

- `taxWallet` is not zero-checked before being used.

```
154          gameController = newGameController;
```

- `newGameController` is not zero-checked before being used.

Recommendation

It is recommended to add a zero-check for the passed-in address value to prevent unexpected errors.

Alleviation

[Playnance Team, June 19, 2024]: Issue acknowledged. I won't make any changes for the current version.

UVD-02 | UNCHECKED ERC-20 `transfer()` / `transferFrom()` CALL

Category	Severity	Location	Status
Volatile Code	● Minor	UpVsDownGameV8.sol (d54cce2): 561	● Acknowledged

Description

The return values of the `transfer()` and `transferFrom()` calls in the smart contract are not checked. Some ERC-20 tokens' transfer functions return no values, while others return a bool value, they should be handled with care. If a function returns `false` instead of reverting upon failure, an unchecked failed transfer could be mistakenly considered successful in the contract.

```
561         token.transferFrom(_msgSender(), address(this), userTrade.amount);
```

Recommendation

It is advised to use the OpenZeppelin's `SafeERC20.sol` implementation to interact with the `transfer()` and `transferFrom()` functions of external ERC-20 tokens. The OpenZeppelin implementation checks for the existence of a return value and reverts if false is returned, making it compatible with all ERC-20 token implementations.

Alleviation

[Playnance Team, June 19, 2024]: Issue acknowledged. I won't make any changes for the current version.

UVD-03 | POTENTIAL REENTRANCY ATTACK (SENDING TOKENS)

Category	Severity	Location	Status
Concurrency	● Minor	UpVsDownGameV8.sol (d54cce2): 298, 330, 331, 363, 397, 433, 561	● Acknowledged

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

External call(s)

```
298                 sendToken(group.addresses[i], group.bets[i]);
```

- This function call executes the following external call(s).
- In `UpVsDownGameV7.sendToken`,
    - `sent = token.transfer(to,amount)`

State variables written after the call(s)

```
307             group.distributedCount = to;
```

External call(s)

```
330                     returnBets(poolId, returnGroupUp, batchSize);
```

- This function call executes the following external call(s).
- In `UpVsDownGameV7.sendToken`,
    - `sent = token.transfer(to,amount)`

```
331                     returnBets(poolId, returnGroupDown, batchSize);
```

State variables written after the call(s)

```
                        clearPool(poolId);
```

- This function call executes the following assignment(s).
- In `UpVsDownGameV7.clearPool` ,
    - `delete pools[poolId].upBetGroup`
- In `UpVsDownGameV7.clearPool` ,
    - `delete pools[poolId].downBetGroup`
- In `UpVsDownGameV7.clearPool` ,
    - `delete pools[poolId].startPrice`
- In `UpVsDownGameV7.clearPool` ,
    - `delete pools[poolId].endPrice`

```
                    returnBets(poolId, returnGroupDown, batchSize);
```

- This function call executes the following assignment(s).
- In `UpVsDownGameV7.returnBets` ,
    - `group.distributedCount = to`

External call(s)

```
                    returnBets(poolId, returnGroup, batchSize);
```

- This function call executes the following external call(s).
- In `UpVsDownGameV7.sendToken` ,
    - `sent = token.transfer(to,amount)`

State variables written after the call(s)

```
                        clearPool(poolId);
```

- This function call executes the following assignment(s).
- In `UpVsDownGameV7.clearPool` ,
    - `delete pools[poolId].upBetGroup`
- In `UpVsDownGameV7.clearPool` ,
    - `delete pools[poolId].downBetGroup`
- In `UpVsDownGameV7.clearPool` ,
    - `delete pools[poolId].startPrice`
- In `UpVsDownGameV7.clearPool` ,
    - `delete pools[poolId].endPrice`

External call(s)

```
                sendToken(winners.addresses[i], winnings + winners.bets[i]);
```

- This function call executes the following external call(s).
- In `UpVsDownGameV7.sendToken` ,
    - `sent = token.transfer(to,amount)`

State variables written after the call(s)

```
                winners.totalDistributed = winners.totalDistributed + winnings;
```

```
            winners.distributedCount = to;
```

```
            for (uint i = winners.distributedCount; i < to; i++) {
```

External call(s)

```
                sendToken(winners.addresses[i], winnings + winners.bets[i]);
```

- This function call executes the following external call(s).

- In `UpVsDownGameV7.sendToken`,
    - `sent = token.transfer(to,amount)`

```
            sendToken(
                feeAddress,
                ((dist.fee + dist.totalMinusFee) * feePercentage) / 100
            );
```

```
            sendToken(
                feeJackpotAddress,
                ((dist.feeJackpot + dist.totalMinusJackpotFee) *
                    feeJackpotPercentage) / 100
            );
```

```
            sendToken(feeAddress, getContractBalance());
```

State variables written after the call(s)

```
            clearPool(poolId);
```

- This function call executes the following assignment(s).
- In `UpVsDownGameV7.clearPool`,
    - `delete pools[poolId].upBetGroup`
- In `UpVsDownGameV7.clearPool`,
    - `delete pools[poolId].downBetGroup`
- In `UpVsDownGameV7.clearPool`,
    - `delete pools[poolId].startPrice`
- In `UpVsDownGameV7.clearPool`,
    - `delete pools[poolId].endPrice`

External call(s)

```
        token.transferFrom(_msgSender(), address(this), userTrade.amount);
```

State variables written after the call(s)

```
        newTotal = addBet(
            betGroup,
            _msgSender(),
            userTrade.amount,
            userTrade.avatarUrl,
            userTrade.countryCode,
            userTrade.whiteLabelId
        );
```

- This function call executes the following assignment(s).
- In `UpVsDownGameV7.addBet` ,
    - `betGroup.bets.push(amount)`
- In `UpVsDownGameV7.addBet` ,
    - `betGroup.addresses.push(signer)`
- In `UpVsDownGameV7.addBet` ,
    - `betGroup.avatars.push(avatar)`
- In `UpVsDownGameV7.addBet` ,
    - `betGroup.countries.push(countryCode)`
- In `UpVsDownGameV7.addBet` ,
    - `betGroup.whiteLabelIds.push(whiteLabelId)`
- In `UpVsDownGameV7.addBet` ,
    - `betGroup.total += amount`

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

[Playnance Team, June 19, 2024]: Issue acknowledged. I won't make any changes for the current version.

UVD-04 | POTENTIAL DIVIDE BY ZERO

Category	Severity	Location	Status
Logical Issue	● Minor	UpVsDownGameV8.sol (d54cce2): 393~394	● Acknowledged

Description

Performing division by zero would raise an error and revert the transaction.

```
393            uint256 winnings = ((winners.bets[i] * dist.totalFees * 100) /
394                winners.total /
```

The expression `(winners.bets[i] * dist.totalFees * 100) / winners.total` may divide by zero.

Note that this is only possible if the minimum bet is 0.

Recommendation

It is recommended to either reformulate the divisor expression, or to use conditionals or require statements to rule out the possibility of a divide-by-zero.

Another option is to enforce the minimum bet to always be positive.

Alleviation

[Playnance Team, June 19, 2024]: Issue acknowledged. I won't make any changes for the current version. The minimum bet will never be 0. There is a lower and upper limit to the bet so zero bet will not be accepted.

[CertiK, June 19, 2024]: As there are no restrictions on the minimum bet, it is possible for the minimum bet to be zero.

UVD-05 | LOCKED BLOCKCHAIN NATIVE TOKENS

Category	Severity	Location	Status
Inconsistency	● Minor	UpVsDownGameV8.sol (d54cce2): 517	● Resolved

Description

The `makeTrade` function is marked as `payable` , making it able to receive native tokens. However, there is no reason for this as the contract deals with ERC20 tokens. This can lead to permanently locked native tokens within the contract.

Recommendation

It is suggested to remove the `payable` attribute.

Alleviation

[Playnance Team, June 19, 2024]: The team heeded the advice and resolved the issue in commit d24b9fe0043ea6bacb5d943e047690742bfe5153 by removing the `payable` attribute.

UVD-06 | USE OF OLD COMPILER VERSIONS

Category	Severity	Location	Status
Language Version	● Minor	UpVsDownGameV8.sol (d54cce2): 3	● Resolved

Description

The contract `UpVsDownGameV8` allows compilation using versions of solidity older than 0.8.0. Such older versions do not include overflow checks when conducting mathematical operations, such as when calculating the winnings after a game.

Recommendation

If an older version of solidity is necessary, it is recommended to use the `SafeMath` library to ensure overflows do not occur. Otherwise, it is recommended to always use a newer version of solidity.

Alleviation

[Playnance Team, June 19, 2024]: The team heeded the advice and resolved the issue in commit a5d6316b90215ef98a32b94302f0017dffd353d1 by requiring the solidity compiler to be at least version 0.8.0.

UVD-07 | FEES CAN EXCEED 100%

Category	Severity	Location	Status
Logical Issue	● Minor	UpVsDownGameV8.sol (d54cce2): 170, 179	● Resolved

Description

Both fee and jackpot fee have a maximum limit of 100%:

```
169     function changeGameFeePercentage(uint8 newFeePercentage) public onlyOwner {
170         require(newFeePercentage <= 100, "Wrong fee percentage value");
```

```
175     function changeGameFeeJackpotPercentage(
176         uint8 newFeeJackpotPercentage
177     ) public onlyOwner {
178         require(
179             newFeeJackpotPercentage <= 100,
180             "Wrong jackpot fee percentage value"
```

However, both fees are applied to winnings, meaning the maximum fee is 200%. Instead of ensuring each fee is at most 100%, the fees should be constrained so that their sum does not exceed 100%.

Recommendation

It is recommended to change the `require` statement to something such as `feePercentage + feeJackpotPercentage <= 100`.

Alleviation

[Playnance Team, June 19, 2024]: The team heeded the advice and resolved the issue in commit 3ed4e26b0fc3b13119bf0de420b2c9ff9d3c070e by ensuring the total of all fees does not exceed 100%.

UVD-08 | POTENTIAL DENIAL-OF-SERVICE ATTACK

Category	Severity	Location	Status
Design Issue	● Minor	UpVsDownGameV8.sol (d54cce2): 298, 397	● Acknowledged

Description

When returning bets or distributing winnings, a loop occurs where tokens are sent to the bettors for each iteration of the loop.

```
297         for (uint i = group.distributedCount; i < to; i++) {
298             sendToken(group.addresses[i], group.bets[i]);
```

Depending on the token used, such as tokens with hooks or callbacks, it may be possible for a malicious better to cause a transfer of tokens to fail, resulting in the entire loop reverting. As there are no other ways for bettors to receive their funds, their tokens will be locked in the contract.

Recommendation

It is recommended to handle the situation when a token transfer fails. An alternative is to create functionality where bettors call a function to receive their funds instead of directly transferring funds to the bettors.

Alleviation

[Playnance Team, June 19, 2024]: Issue acknowledged. I won't make any changes for the current version. We are not taking bets from Smart contracts so we will not send tokens to smart contract that may not have receive functionality and can stuck the distribution process.

[CertiK, June 19, 2024]: This issue is more about the tokens used and the bettor does not need to be a smart contract for a revert to occur. For example, if a token has a blacklist and a bettor is added to the list after betting, they may not be able to receive their winnings.

UVD-09 | INSUFFICIENT CHECK FOR EOA

Category	Severity	Location	Status
Logical Issue	● Minor	UpVsDownGameV8.sol (d54cce2): 628	● Resolved

Description

The function `isEOA` is used to check if an address is an EOA or contract by checking if the address has any code.

```
626      function isEOA(address _addr) private view returns (bool) {
627          // Checks if the caller is an EOA
628          return _addr.code.length == 0;
```

However, this is an insufficient check and can be bypassed, for instance if a function call was made from a contract's constructor, then the contract's code length will be 0 at that time.

Recommendation

It is recommended to consider adding an additional check to ensure the address is an EOA. Since `isEOA` is only used with `msg.sender` as an input, one option is requiring `msg.sender == tx.origin`.

Alleviation

[Playnance Team, June 19, 2024]: The team heeded the advice and resolved the issue in commit 9f83d5f05d8b120e28113ab74296e74a3d1d125a.

UVG-11 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	● Informational	UpVsDownGameV8.sol (d54cce2): 158~167, 169~173, 175~184, 186~193, 195~204, 218~228, 595~600, 602~616	● Acknowledged

Description

It is important to emit events for sensitive actions, particularly those that can be executed by centralized roles or administrators. This ensures transparency and enables tracking of critical changes, which is essential for security and trust in the system. Missing event logs can indeed result in a lack of visibility and potential information loss.

Recommendation

It is recommended to emit events in sensitive functions that are controlled by centralization roles.

Alleviation

[Playnance Team, June 19, 2024]: Issue acknowledged. I will fix the issue in the future, which will not be included in this audit engagement.

APPENDIX | PLAYNANCE - AUDIT 2

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.
Language Version	Language Version findings indicate that the code uses certain compiler versions or language features with known security issues.
Concurrency	Concurrency findings are about issues that cause unexpected or unsafe interleaving of code executions.
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.

DISCLAIMER | CERTIK

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

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

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

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

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

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

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

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

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

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

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

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

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

CertiK | Securing the Web3 World

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