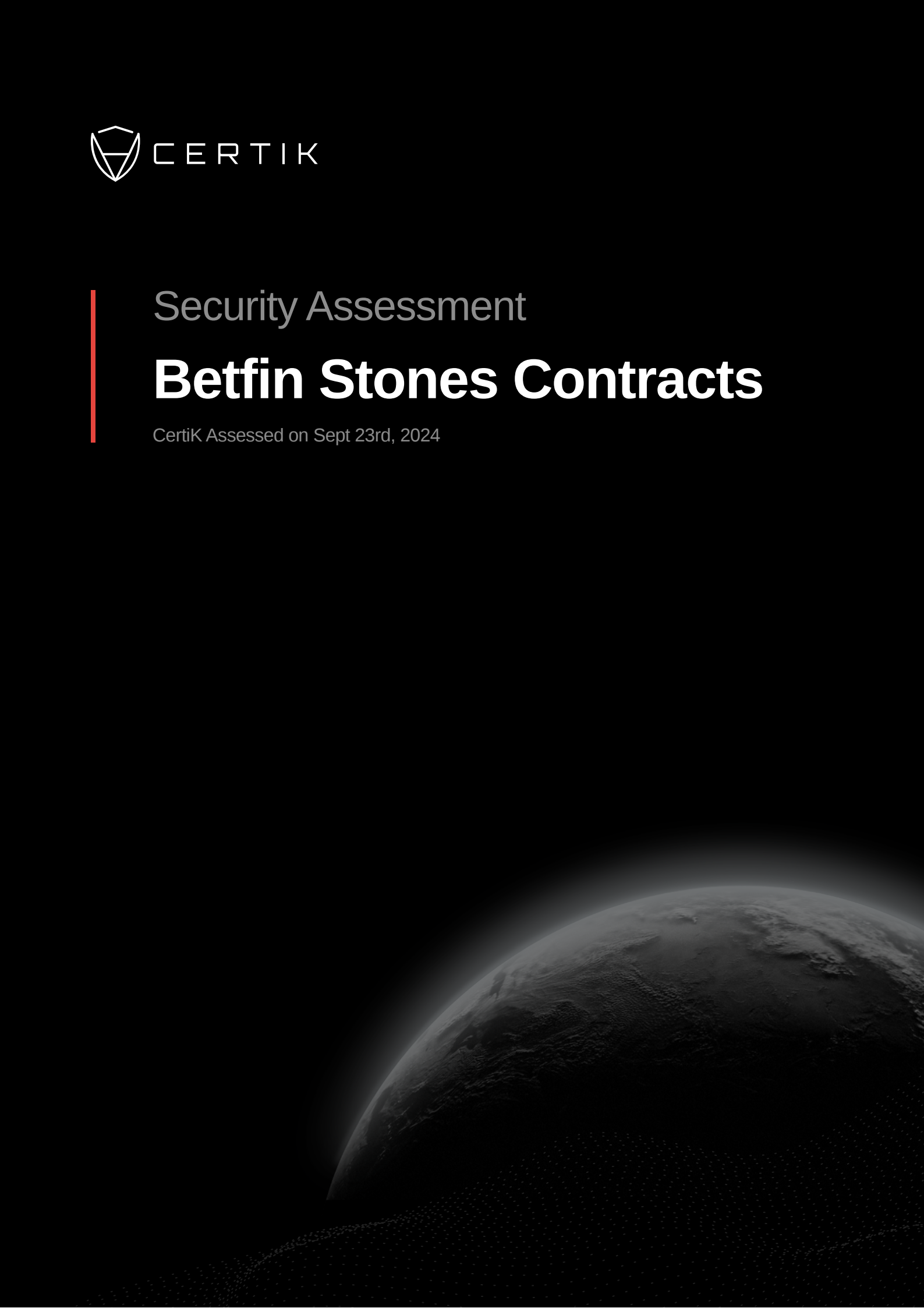
CERTIK

Security Assessment

Betfin Stones Contracts

CertiK Assessed on Sept 23rd, 2024



CertiK Assessed on Sept 23rd, 2024

Betfin Stones Contracts

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

Executive Summary

TYPES
Gaming

ECOSYSTEM
Ethereum (ETH)

METHODS
Formal Verification, Manual Review, Static Analysis

LANGUAGE
Solidity

TIMELINE
Delivered on 09/23/2024

KEY COMPONENTS
N/A

CODEBASE
stones-contract
View All in Codebase Page

COMMITS
- 14edb33c8ce4ab611d1108b1cb5c2c6c5a508d33
- 83e8acd0fbddb32013d25935ef348c756f06ae76
- f34c8aad90de30685af23c0293e9e0926be3d493

View All in Codebase Page

Vulnerability Summary

8	6	0	0	2	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.
0	Major		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.
1	Medium	1 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
6	Minor	5 Resolved, 1 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 | BETFIN STONES CONTRACTS

CODEBASE | BETFIN STONES CONTRACTS

Repository

stones-contract

Commit

- 14edb33c8ce4ab611d1108b1cb5c2c6c5a508d33
- 83e8acd0fbddb32013d25935ef348c756f06ae76
- f34c8aad90de30685af23c0293e9e0926be3d493

AUDIT SCOPE | BETFIN STONES CONTRACTS

6 files audited ● 6 files without findings

ID	Repo	File	SHA256 Checksum
STO	betfinio/stones-contract	Stones.sol	1ef30ac3de9605a0fcf6ef3e36b25813f94d286d62d67037312f5d3b13d0639d
SBB	betfinio/stones-contract	StonesBet.sol	ed6e26d7cdb1289a35203e5bc5993904fdc044a0da26ad30d811899ce467978f
STN	betfinio/stones-contract	Stones.sol	654cd506b16739f162af16fa03cf2b93a2c158f03eee2e78cef0a1b4a9e192e3
SBU	betfinio/stones-contract	StonesBet.sol	46f428f4d833521f2753052f05eb02cfcce7c26983ccd35d1ff964b13dbf2a9b
STE	betfinio/stones-contract	src/Stones.sol	0f8abccf0bf5310f96f45147c3f6970252f45d81bc4288fec50ab35e483aef8a
SBH	betfinio/stones-contract	src/StonesBet.sol	46f428f4d833521f2753052f05eb02cfcce7c26983ccd35d1ff964b13dbf2a9b

APPROACH & METHODS | BETFIN STONES CONTRACTS

This report has been prepared for Betfin to discover issues and vulnerabilities in the source code of the Betfin Stones Contracts 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 | BETFIN STONES CONTRACTS

Overview

The **Betfin Stones** project facilitates a betting game where players place bets on different sides in each round. After all bets are placed, VRF oracle service is used to determine a random winner. The contract manages bet placement, bank updates, winner determination, and payout distribution.

External Dependencies

In **Betfin Stones**, the project relies on a few external contracts or addresses to fulfill the needs of its business logic.

The following are third dependencies contracts used within the `Stones` and `StonesBet` contracts:

- `openzeppelin`: including `ReentrancyGuard`, `IERC20`, `SafeERC20` and `Ownable`;
- `chainlink`: including `VRFCoordinatorV2_5` and `VRFConsumerBaseV2Plus`.

It is assumed that these contracts or addresses are trusted and properly implemented within the entire project.

The team utilizes the subscription method of the Chainlink VRF service to generate random numbers. It is assumed that the `subscriptionId` in the project is always valid and maintains a sufficient balance to fund requests from consumer contracts.

FINDINGS | BETFIN STONES CONTRACTS

8	0	0	1	6	1
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Betfin Stones Contracts. Through this audit, we have uncovered 8 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
STO-02	Inconsistent Fee Status May Permanently Lock Fees In The Contract	Design Issue	Medium	Resolved
SBB-03	Missing Emit Events	Inconsistency	Minor	Resolved
SRC-01	Missing Zero Address Validation	Volatile Code	Minor	Resolved
STE-01	Check-Effects-Interactions Pattern Violation	Coding Style	Minor	Resolved
STO-03	External Call Inside Loop	Denial of Service	Minor	Resolved
STO-04	Third-Party Dependency Usage	Design Issue	Minor	Acknowledged
STO-05	Repeated And Inconsistent Error Message	Inconsistency	Minor	Resolved
STO-01	Unpredictable `block.timestamp` In `getCurrentRound()` Function	Design Issue	Informational	Acknowledged

STO-02 INCONSISTENT FEE STATUS MAY PERMANENTLY LOCK FEES IN THE CONTRACT

Category	Severity	Location	Status
Design Issue	Medium	Stones.sol (08/29-Stones-6ddc86): 253~255	Resolved

Description

Before `core` calls `placeBet()` function, it will determine whether to charge a fee based on `getFeeType()` in the `Stones` contract. In core, when `getFeeType()==0`, the fee is charged and totalAmount will be deducted from the fee before being transferred to the `Stones` contract. When `getFeeType==1`, totalAmount is fully transferred to the `Stones` contract.

In this contract, the result of the `getFeeType()` function is `1`, indicating no fee is charged. However, in the `executeResult()` function, `roundBank` (the total amount of tokens for a specified round) is deducted from the fee before calculating `winAmount` and `bonusAmount`. We would like to confirm with the team if this is the intended design. If so, the fee amount will be locked in the contract.

Proof of Concept

The POC shows that there would be 3.6% bet tokens locked in the game contract.

```
    function testFullfill_sameUser() public {
        uint256 round = stones.getCurrentRound();
        placeBet(alice, 1000, 1, round);
        placeBet(alice, 1000, 2, round);
        placeBet(alice, 1000, 3, round);
        placeBet(alice, 1000, 4, round);
        placeBet(alice, 1000, 5, round);
        assertEq(stones.roundStatus(round), 0);

        vm.warp(block.timestamp + 1 days);
        getRequest(5);
        stones.roll(round);
        assertEq(stones.roundStatus(round), 1);

        uint256[] memory result = new uint256[](1);
        result[0] = uint256(1);
        vm.startPrank(stones.vrfCoordinator());
        stones.rawFulfillRandomWords(5, result);

        assertEq(stones.roundWinnerSide(round), 1);
        assertEq(stones.roundStatus(round), 2);

        console2.log("balance in stones is %d", token.balanceOf(address(stones)));

        vm.expectEmit(address(token));
        emit Transfer(address(stones), alice, 4820 ether);
        assertEq(stones.roundStatus(round), 2);
        stones.executeResult(round);
        assertEq(stones.roundStatus(round), 3);
        vm.expectRevert(bytes("ST03"));
        stones.executeResult(round);

        console2.log("balance in stones is %d", token.balanceOf(address(stones)));
    }
```

Test resuluts:

```
% forge test --mt testFullfill_sameUser -vv
[⠒] Compiling...
[⠒] Compiling 3 files with 0.8.19
[⠒] Solc 0.8.19 finished in 4.17s
Compiler run successful!

Ran 1 test for test/Stones.t.sol:StonesTest
[PASS] testFullfill_sameUser() (gas: 3349797)
Logs:
  balance in stones is 5000000000000000000000
  balance in stones is 18000000000000000000

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 6.58ms (1.47ms CPU
time)

Ran 1 test suite in 146.94ms (6.58ms CPU time): 1 tests passed, 0 failed, 0 skipped
(1 total tests)
```

Recommendation

It is recommended to ensure the correct fee status is set or to transfer the appropriate amount of tokens to the contract.

Alleviation

[Betfin Team, 09/06/2024]:

Issue acknowledged. The team resolved this issue in the commit hash fdd5e3f70c293f833c2a4851330a0ecdf39c930a by changing the fee status to charging status and the fee will be deducted in the `core` contract.

SBB-03 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Inconsistency	Minor	StonesBet.sol (08/29-StonesBet-6ddc86): 71~73, 75~77	Resolved

Description

The smart contract contains one or more state changes that do not emit events to communicate the changes outside the blockchain, which can lead to difficulties in tracking or verifying these changes and may affect the contract's transparency and auditability.

```
71      function setStatus(uint256 _status) public onlyOwner {
72          status = _status;
73      }
```

```
75      function setResult(uint256 _result) public onlyOwner {
76          result = _result;
77      }
```

Recommendation

It is suggested to declare and emit corresponding events for all the essential state variables that are possible to be changed during runtime.

Alleviation

[Betfin Team, 09/06/2024]:

Issue acknowledged. The team resolved this issue in the commit hash 4f0603c3728013b2b14fc5692c3dde781a8e61e5 by adding corresponding events for the status variables setting functions.

SRC-01 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	Stones.sol (08/29-Stones-6ddc86): 81~82, 85; StonesBet.sol (08/29-StonesBet-6ddc86): 21, 22	Resolved

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.

In the `Stones` contract, the following provided addresses in `constructor` function are not zero-checked before being used.

```
constructor(
    uint256 _subscriptionId,
    address _core,
    address _staking,
    ...
    address _admin
) VRFConsumerBaseV2Plus(_vrfCoordinator) {
    ...
}
```

In the `StonesBet` contract, the following provided addresses in `constructor` function are not zero-checked before being used.

```
constructor(
    address _player,
    address _game,
      ...
) {
    ...
}
```

Recommendation

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

Alleviation

[Betfin Team, 09/06/2024]:

Issue acknowledged. The team resolved this issue in the commit hash 8da986afbb1bf165142fa46bf996e3e312e7e7e4 by adding the zero-check for input addresses in the constructor functions.

STE-01 | CHECK-EFFECTS-INTERACTIONS PATTERN VIOLATION

Category	Severity	Location	Status
Coding Style	Minor	Stones.sol (09/10-Stones-761475): 254	Resolved

Description

This Checks-Effects-Interactions Pattern is a best practice for writing secure smart contracts that involves performing all state changes before making any external function calls.

External call(s)

```
254             IERC20(token).safeTransfer(bet.getPlayer(), result);
```

State variables written after the call(s)

```
255             betSettled[address(bet)] = true;
256             distributedInRound[round] += result;
```

The `executeResult()` function is used to transfer tokens to winners, this pattern could help prevent a user from exploiting unintended flow patterns.

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

[Betfin Team, 09/13/2024]:
Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/betfinio/stones-contract/commit/1d6fdd0675c2f4d952a9604a79dca084f8403880.

STO-03 | EXTERNAL CALL INSIDE LOOP

Category	Severity	Location	Status
Denial of Service	Minor	Stones.sol (08/29-Stones-6ddc86): 193~243	Resolved

Description

External call `token.safetransfer()` is made inside a `for` loop. This may lead to a denial-of-service attack. If any of the calls fail, the entire loop will revert, causing all tokens deposited within this round to be locked in the contract.

```
    function executeResult(uint256 round) public {
        ...
        for (uint256 i = 0; i < betsCount; i++) {
            // related calculations
            ...
            IERC20(address(staking.getToken())).safeTransfer(
                bet.getPlayer(),
                winAmount + bonusAmount
            );
        }
        // get all other bets
        ...
    }
```

Recommendation

It is advised to refactor the code to move external calls outside the loop, or use alternative patterns such as the "Withdrawal Pattern" to minimize the risk of denial-of-service attacks and improve the overall security of the smart contract.

Alleviation

[Betfin Team, 09/06/2024]: The team added a check function to check balance of Stones contract if it has enough funds before sending. But even if it revert, all we do is send some tokens to `Stones` contract to resolve the problem.

[CertiK, 09/09/2024]
In the `for` loop, the contract distributes `winAmount` and `bonusAmount` to each winner, but the current `roundBank` reflects the deduction of `fee` and the `bonusBank`. To ensure correct implementation and guarantee that the contract has sufficient tokens to distribute to the winners during the loop, we recommend verifying that `IERC20(token).balanceOf(address(this)) >= roundBank + bonusBank`.

Additionally, it's noted that the `executeResult` function in the `Stones` contract possesses a vulnerability related to potential out-of-gas errors due to unlimited bet processing within a single transaction. This issue stems from the lack of

constraints on the number of bets per round, which can lead to excessive gas consumption when the function tries to process an extremely high number of bets.

```
        uint256 allBetsCount = roundBets[round].length;
        for (uint256 i = 0; i < allBetsCount; i++) {
            StonesBet bet = roundBets[round][i];
            // skip if winner side
            if (bet.getSide() == side) continue;
            // set bet status
            bet.setStatus(3);
            bet.renounceOwnership();
        }
```

Specifically, if `roundBets[round].length` is very large, the cumulative gas required to execute `executeResult` may exceed the block gas limit, causing the transaction to fail with an out-of-gas error. This not only prevents the distribution of winnings but also leaves the game's state in limbo if the round cannot be successfully concluded.

It's recommended to refactor the contract to consider the potential out-of-gas error.

[Betfin Team, 09/10/2024]:
Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/betfinio/stones-contract/commit/76147503d0ceb4c6aae6225f56ce99fa88accb4f.

STO-04 | THIRD-PARTY DEPENDENCY USAGE

Category	Severity	Location	Status
Design Issue	Minor	Stones.sol (08/29-Stones-6ddc86): 50, 51	Acknowledged

Description

The contract is serving as the underlying entity to interact with one or more third party protocols. 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.

```
50      StakingInterface public immutable staking;
```

- The contract `Stones` interacts with third party contract with `StakingInterface` interface via `staking`.

```
51      CoreInterface public immutable core;
```

- The contract `Stones` interacts with third party contract with `CoreInterface` interface via `core`.

```
    uint256 private immutable created;
    uint256 private immutable subscriptionId;
    address public immutable vrfCoordinator;
    bytes32 public immutable keyHash;
    uint32 private constant callbackGasLimit = 2_500_000;
    uint16 public constant requestConfirmations = 3;
    uint32 private constant numWords = 1;

    constructor(
        uint256 _subscriptionId,
        address _core,
        address _staking,
        address _vrfCoordinator,
        bytes32 _keyHash,
        address _admin
    ) VRFConsumerBaseV2Plus(_vrfCoordinator) {
        // validation for vrf coordinator is not needed because it is already
validated in the VRFConsumerBaseV2Plus contract
        vrfCoordinator = _vrfCoordinator;
        keyHash = _keyHash;
        subscriptionId = _subscriptionId;
        core = CoreInterface(_core);
        require(core.isStaking(_staking), "ST06");
        staking = StakingInterface(_staking);
        created = block.timestamp;
        _grantRole(DEFAULT_ADMIN_ROLE, _admin);
    }
```

Since they are immutable or constant, the project team needs to ensure that the `vrfCoordinator` is always callable, and that the `keyHash` and `subscriptionId` are always valid, as well as that the `callbackGasLimit` is sufficient to execute the `rawFulfillRandomWords()` callback.

In particular, the project team needs to prevent the risk of request failures caused by an invalid `subscriptionId`. It means the project team need to ensure that the request sent by calling `requestRandomWords()` in each round correctly triggers the callback.

Recommendation

The auditors understood that the business logic requires interaction with third parties. It is recommended for the team to constantly monitor the statuses of third parties to mitigate the side effects when unexpected activities are observed.

Alleviation

[Betfin Team, 09/06/2024]:

Issue Acknowledged. The team decided not to change the current codebase and will monitor the third parties.

STO-05 | REPEATED AND INCONSISTENT ERROR MESSAGE

Category	Severity	Location	Status
Inconsistency	Minor	Stones.sol (08/29-Stones-6ddc86): 23	Resolved

Description

Error codes `ST05` and `ST06` are repeated. Additionally, the `ST06` code is inconsistent with the corresponding code in the `constructor` function. The code in the `require` statement checks the status of the provided `_staking` instead of the transfer operation.

```
constructor(
    ...
) VRFConsumerBaseV2Plus(_vrfCoordinator) {
    ...
    require(core.isStaking(_staking), "ST06");
    ...
}
```

Recommendation

It is recommended to use appropriate error codes and update the error messages to accurately reflect the function's behavior for clarity.

Alleviation

[Betfin Team, 09/06/2024]:

Issue acknowledged. The team resolved this issue in the commit hash 8da986afbb1bf165142fa46bf996e3e312e7e7e4 by correcting the error messages in the contracts.

STO-01 UNPREDICTABLE `block.timestamp` IN `getCurrentRound()` FUNCTION

Category	Severity	Location	Status
Design Issue	Informational	Stones.sol (08/29-Stones-6ddc86): 112	Acknowledged

Description

The `getCurrentRound()` function calculates the current round based on the current `block.timestamp`, and its result is compared with the `_round` provided in the calldata. However, the `block.timestamp` at the time of transaction execution is unpredictable.

Recommendation

We would like to check with the team if there is a strategy to ensure that when a user places a bet, the current round matches the expected round, i.e., the `_round` specified in the calldata.

Alleviation

[Betfin Team, 09/06/2024]:
Issue Acknowledged. The team confirmed the current implementation is correct and users should place matched bet round.

OPTIMIZATIONS | BETFIN STONES CONTRACTS

ID	Title	Category	Severity	Status
SBB-01	Variables That Could Be Declared As Immutable	Gas Optimization	Optimization	Resolved
STO-07	Code Redundancy	Gas Optimization	Optimization	Resolved

SBB-01 | VARIABLES THAT COULD BE DECLARED AS IMMUTABLE

Category	Severity	Location	Status
Gas Optimization	Optimization	StonesBet.sol (08/29-StonesBet-6ddc86): 18	Resolved

Description

Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.

The `side` variable assigned in the constructor can be declared as `immutable`.

```
uint256 private side;

constructor(
    ...
    uint256 _side
) {
    ...
    side = _side;
}
```

Recommendation

We recommend declaring these variables as immutable. Please note that the `immutable` keyword only works in Solidity version `v0.6.5` and up.

Alleviation

[Betfin Team, 09/06/2024]:
Issue acknowledged. The team resolved this issue in the commit hash 14edb33c8ce4ab611d1108b1cb5c2c6c5a508d33 by declaring the specified variable as `immutable`.

STO-07 | CODE REDUNDANCY

Category	Severity	Location	Status
Gas Optimization	Optimization	Stones.sol (09/06-Stones-14edb3): 226, 241	Resolved

Description

Since the `Stones` contract is hard to compromise, so the owner of the `StonesBet` contract is protected, with state modifications in the `StonesBet` contract only occurring within specific functions of the `Stones` contract, the `bet.renounceOwnership()` function is not necessary to mitigate centralization risk in this case.

In addition, since the access control and timelock mechanisms are not used in this contract, removing the relevant code can reduce the gas cost of the contract.

Recommendation

It is recommended to remove the redundant codes to save gas.

Alleviation

[Betfin Team, 09/10/2024]:
Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/betfinio/stones-contract/commit/15afd8fe36c6fefb28e5cbfc6f685047a3d005dd

APPENDIX | BETFIN STONES CONTRACTS

Finding Categories

Categories	Description
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
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.
Denial of Service	Denial of Service findings indicate that an attacker may prevent the program from operating correctly or responding to legitimate requests.
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.
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.