



CONTRACT
CHECKER

Blockchain Solutions



<https://t.me/contractchecker>

contact@contractchecker.app

contractchecker.app

Anywhere on the Blockchain

Date: 13.10.2022

Smart Contract Security Audit

SPORT FANTASY PREDICTION



Harry K

Harry Kedelman
General Manager

Audit Result



 SPORT FANTASY PREDICTION has successfully **PASSED** the smart contract source code audit with below listed privileges

(Other unknown security vulnerabilities are not included in the audit responsibility scope)

Audit Result:	PASSED
Ownership:	Not renounced yet
KYC Verification:	NA At the date of report edition
Audit Date:	October 13, 2022
Audit Team:	CONTRACTCHECKER

Findings

Privileges of Ownership

-  Prediction game rewards supposed to be distributed from an external account which can be changed at any time by owner
-  Owner can change Reward token contract at any time

Important Notice for Investors

As Contract Checker team we are mainly auditing the contract code to find out how it will be functioning, and risks which are hidden in the code if any.

There are many factors must be taken into consideration before investing to a project, like: ownership status, project team approach, marketing, general market condition, liquidity, token holdings etc.

Investors must always do their own research and manage their risk considering different factors which can affect the success of a project.

Table of Contents

Audit Result	1
Findings	1
Privileges of Ownership	1
Important Notice for Investors	1
SUMMARY	3
Project Summary	3
OVERVIEW	4
Auditing Approach and Applied Methodologies	4
Security	4
Sound Architecture	4
Code Correctness and Quality	4
Risk Classification	5
High level vulnerability	5
Medium level vulnerability	5
Low level vulnerability	5
Manual Audit:	5
Smart Contract SWC Attack Test	6
➤ SWC-108: State variable visibility is not set	7
➤ SWC-116: A control flow decision is made based on The block.timestamp environment variable	7
➤ SWC-123: Requirement violation	7
Automated Audit	11
Remix Compiler Warnings	11
Disclaimer	12

SUMMARY

CONTRACTCHECKER received an application for smart contract security audit of SPORT FANTASY PREDICTION on October 12, 2022, from the project team to discover if any vulnerability in the source codes of the SPORT FANTASY PREDICTION as well as any contract dependencies. Detailed test has been performed using Static Analysis and Manual Review techniques.

The auditing process focuses to the following considerations with collaboration of an expert team

- Functionality test of the Smart Contract to determine if proper logic has been followed throughout the whole process.
- Manually detailed examination of the code line by line by experts.
- Live test by multiple clients using Testnet.
- Analysing failure preparations to check how the Smart Contract performs in case of any bugs and vulnerabilities.
- Checking whether all the libraries used in the code are on the latest version.
- Analysing the security of the on-chain data.

Project Summary

Token Name	SPORT FANTASY PREDICTION
Web Site	https://sportfantasy.finance
Twitter	https://twitter.com/Sportfts
Telegram	https://t.me/sportfantasyp2e
Platform	Binance Smart Chain
Token Type	BEP20
Language	Solidity
Platforms & Tools	Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Mythril, Contract Library
Contract	0x3aa8e3638985b1039abda929301771692b8a8116
Contract Link	https://bscscan.com/address/0x3aa8e3638985b1039abda929301771692b8a8116
Test Link	https://testnet.bscscan.com/address/0x8ba22a643bf113456c738eb1481c80b0b883f667

OVERVIEW

This Audit Report mainly focuses on overall security of SPORT FANTASY PREDICTION smart contract. Contract Checker team scanned the contract and assessed overall system architecture and the smart contract codebase against vulnerabilities, exploitations, hacks, and back-doors to ensure its reliability and correctness.

Auditing Approach and Applied Methodologies

Contract Checker team has performed rigorous test procedures of the project

- Code design patterns analysis in which smart contract architecture is reviewed to ensure it is structured according to industry standards and safe use of third-party smart contracts and libraries.
- Line-by-line inspection of the Smart Contract to find any potential vulnerability like race conditions, transaction-ordering dependence, timestamp dependence, and denial of service attacks.
- Unit testing Phase, we coded/conducted custom unit tests written for each function in the contract to verify that each function works as expected.
- Automated Test performed with our in-house developed tools to identify vulnerabilities and security flaws of the Smart Contract.

The focus of the audit was to verify that the Smart Contract System is secure, resilient, and working according to the specifications. The audit activities can be grouped in the following three categories:

Security

Identifying security related issues within each contract and the system of contract.

Sound Architecture

Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

Code Correctness and Quality

A full review of the contract source code. The primary areas of focus include:

- Accuracy
- Readability
- Sections of code with high complexity
- Quantity and quality of test coverage

Risk Classification

Vulnerabilities are classified in 3 main levels as below based on possible effect to the contract.

High level vulnerability

Vulnerabilities on this level must be fixed immediately as they might lead to fund and data loss and open to manipulation. Any High-level finding will be highlighted with **RED** text

Medium level vulnerability

Vulnerabilities on this level also important to fix as they have potential risk of future exploit and manipulation. Any Medium-level finding will be highlighted with **ORANGE** text

Low level vulnerability

Vulnerabilities on this level are minor and may not affect the smart contract execution. Any Low-level finding will be highlighted with **BLUE** text

Manual Audit:

For this section the code was tested/read line by line by our developers. Additionally, Remix IDE's JavaScript VM and Kovan networks used to test the contract functionality.

Smart Contract SWC Attack Test

SWC ID	Description	Test Result
SWC-100	Function Visibility	Passed
SWC-101	Integer Overflow and Underflow	Passed
SWC-102	Outdated Compiler Version	Passed
SWC-103	Floating Pragma	Passed
SWC-104	Unchecked Call Return Value	Passed
SWC-105	Unprotected Ether Withdrawal	Passed
SWC-106	Unprotected SELFDESTRUCT Instruction	Passed
SWC-107	Re-entrancy	Passed
SWC-108	State Variable Default Visibility	LOW
SWC-109	Uninitialized Storage Pointer	Passed
SWC-110	Assert Violation	Passed
SWC-111	Use of Deprecated Solidity Functions	Passed
SWC-112	Delegate Call to Untrusted Callee	Passed
SWC-113	DoS with Failed Call	Passed
SWC-114	Transaction Order Dependence	Passed
SWC-115	Authorization through tx.origin	Passed
SWC-116	Block values as a proxy for time	LOW
SWC-117	Signature Malleability	Passed
SWC-118	Incorrect Constructor Name	Passed
SWC-119	Shadowing State Variables	Passed
SWC-120	Weak Sources of Randomness from Chain Attributes	Passed
SWC-121	Missing Protection against Signature Replay Attacks	Passed
SWC-122	Lack of Proper Signature Verification	Passed
SWC-123	Requirement Violation	LOW
SWC-124	Write to Arbitrary Storage Location	Passed
SWC-125	Incorrect Inheritance Order	Passed
SWC-126	Insufficient Gas Griefing	Passed
SWC-127	Arbitrary Jump with Function Type Variable	Passed
SWC-128	DoS With Block Gas Limit	Passed
SWC-129	Typographical Error	Passed
SWC-130	Right-To-Left-Override control character (U+202E)	Passed
SWC-131	Presence of unused variables	Passed
SWC-132	Unexpected Ether balance	Passed
SWC-133	Hash Collisions with Multiple Variable Length Arguments	Passed
SWC-134	Message call with hardcoded gas amount	Passed
SWC-135	Code With No Effects (Irrelevant/Dead Code)	Passed
SWC-136	Unencrypted Private Data On-Chain	Passed

➤ SWC-108: State variable visibility is not set

It is best practice to set the visibility of state variables explicitly. The default visibility for "userPredictions" is internal. Other possible visibility settings are public and private.

```
194
195 // 1 is first team win, 2 is sencond team win, 3 is draw
196 mapping(address => mapping(uint256 => uint256)) userPredictions;
197 mapping(address => mapping(uint256 => bool)) userMatchClaimed;
198
```

The default visibility for "userMatchClaimed" is internal. Other possible visibility settings are public and private.

```
195 // 1 is first team win, 2 is sencond team win, 3 is draw
196 mapping(address => mapping(uint256 => uint256)) userPredictions;
197 mapping(address => mapping(uint256 => bool)) userMatchClaimed;
198
199 address public SFS = 0x57D1b20840e34Ed8F9b142282099f34164375F1E;
```

➤ SWC-116: A control flow decision is made based on The block.timestamp environment variable

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coin base, gas limit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

```
215 function addMatch(uint256 _startTime,string memory _stTeam,string memory _ndTeam) public onlyUpdater{
216     require(_startTime > block.timestamp, "time error");
217
218     uint256 matchId = matchCount;
```

➤ SWC-123: Requirement violation

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

```
305 function getUserBalance(address user) external view returns (uint256) {
306     return IERC20[SFS].balanceOf(user);
307 }
```

```
182 contract SportFantasyPrediction is Ownable, Updatable {
183     using SafeMath for uint256;
184
185     event MatchCreated(uint256 matchId, string stTeam, string ndTeam, uint256 startTime);
186     event MatchResulted(uint256 matchId, string stTeam, string ndTeam, uint256 startTime, bool isEnded, uint256 result);
187     event Predicted(uint256 matchId, address indexed user, uint256 result);
188     event RewardClaimed(uint256 matchId, address indexed user, uint256 rewardValue);
189
190     uint256 public matchCount;
191     uint256 public endedMatchCount;
192     uint256 public currentMatchCount;
193     address public rewardPoolAddress = 0x68AcB7D164C1F0a49da6D6e0E3b2795B1Eddf0d;
194
195     // 1 is first team win, 2 is sencond team win, 3 is draw
196     mapping(address => mapping(uint256 => uint256)) userPredictions;
197     mapping(address => mapping(uint256 => bool)) userMatchClaimed;
198
199     address public SFS = 0x57D1b20840e34Ed8F9b142282099f34164375F1E;
200     uint256 public minPredictRequire = 600 * 10**8;
```


CONTRACT CHECKER

```
202 struct Match {
203     uint256 matchId;
204     string stTeam;
205     string ndTeam;
206     bool isEnded;
207     uint256 startTime;
208     uint256 result;
209 }
210
211 Match[] private _matches;
212
213 constructor() {
214
215     function addMatch(uint256 _startTime, string memory _stTeam, string memory _ndTeam) public onlyUpdater {
216         require(_startTime > block.timestamp, "time error");
217
218         uint256 matchId = matchCount;
219
220         _matches.push(Match(matchId, _stTeam, _ndTeam, false, _startTime, 0));
221
222         matchCount = matchCount + 1;
223         currentMatchCount = currentMatchCount + 1;
224     }
225 }
```

```
225 emit MatchCreated(matchId, _stTeam, _ndTeam, _startTime);
226 }
227
228 function isPredicted(uint256 _matchId, address user) external view returns(bool) {
229     require(_matchId >= 0 && _matchId < matchCount, "not found matchId");
230
231     return userPredictions[user][_matchId] > 0;
232 }
233
234 function isClaimable(uint256 _matchId, address user) external view returns(bool) {
235     require(_matchId >= 0 && _matchId < matchCount, "not found matchId");
236
237     Match memory _match = _matches[_matchId];
238
239     if(_match.isEnded == true)
240         && userMatchClaimed[user][_matchId] == false
241         && userPredictions[user][_matchId] == _match.result
242     {
243         return true;
244     }
245
246     return false;
247 }
248
249 function claimReward(uint256 _matchId) public {
250     require(_matchId >= 0 && _matchId < matchCount, "not found matchId");
251
252     Match memory _match = _matches[_matchId];
253     require(!_match.isEnded == true, "match has not ended!");
254
255     uint256 predictedResult = userPredictions[msg.sender][_matchId];
256     require(predictedResult > 0, "not predicted yet!");
257     require(predictedResult == _match.result, "predicted wrong result!");
258     require(userMatchClaimed[msg.sender][_matchId] == false, "has claimed yet!");
259 }
```

```

254 uint256 totalHold -= IERC20[SFS].balanceOf(msg.sender);
255 uint256 totalSupply -= IERC20[SFS].totalSupply();
256 uint256 totalReward -= IERC20[SFS].balanceOf(rewardPoolAddress);
257 uint256 rewardValue = totalReward.div(totalSupply.div(totalHold)+1) + totalReward.div(totalSupply.mod(totalHold)+1);
258 IERC20[SFS].transferFrom(rewardPoolAddress, msg.sender, rewardValue);
259 userMatchClaimed[msg.sender][_matchId] = true;
260
261 emit RewardClaimed(_matchId, msg.sender, rewardValue);
262
263
264 function predict(uint256 _matchId, uint256 _result) public
265 require(_matchId >= 0 && _matchId < matchCount, "not found matchId")
266 require(_result > 0 && _result <= 5, "result error")
267 require(IERC20[SFS].balanceOf(msg.sender) >= minPredictRequire, "not enough SFS to predict")
268
269 Match memory _match = _matches[_matchId];
270
271 require(_match.isEnded == false, "match has ended!");
272 require(block.timestamp < _match.startTime, "match has started!");
273
274 userPredictions[msg.sender][_matchId] = _result;
275
276 userMatchClaimed[msg.sender][_matchId] = false;
277
278 emit Predicted(_matchId, msg.sender, _result);
279
280
281
282 function updateResult(uint256 _matchId, uint256 _stResult, uint256 _ndResult) public onlyUpdater
283 require(_matchId >= 0 && _matchId < matchCount, "not found matchId")
284
285
286 require(_matches[_matchId].isEnded == false, "match has ended!");
287
288
289 if(_stResult == _ndResult)
290     _matches[_matchId].result = 3;
291 else if(_stResult > _ndResult)
292     _matches[_matchId].result = 1;
293 else
294     _matches[_matchId].result = 2;
295
296
297 _matches[_matchId].isEnded = true;
298
299 currentMatchCount = currentMatchCount - 1;
300 endedMatchCount = endedMatchCount + 1;
301
302 emit MatchResulted(_matchId, _matches[_matchId].stTeam, _matches[_matchId].ndTeam, _matches[_matchId].startTime, _matches[_matchId].isEnded, _matches[_matchId].result);
303
304
305 function getCurrentMatches() external view returns (Match[] memory) {
306     Match[] memory currentMatches = new Match[](currentMatchCount);
307     uint256 count = 0;
308
309     for(uint256 i = 0; i < matchCount; i++)
310         if(_matches[i].isEnded == false)
311             currentMatches[count] = _matches[i];
312     count++;

```

CONTRACT CHECKER

```
313 }
314 }
315 return currentMatches;
316 }
317
318 function getEndedMatches() external view returns (Match[] memory) {
319     Match[] memory endedMatches = new Match[](endedMatchCount);
320     uint256 count = 0;
321     for (uint256 i = 0; i < matchCount; i++) {
322         if (_matches[i].isEnded == true) {
323             endedMatches[count] = _matches[i];
324             count++;
325         }
326     }
327 }
328
329 return endedMatches;
330 }
331
332 function getAllMatches() external view returns (Match[] memory) {
333     Match[] memory allMatches = new Match[](matchCount);
334     for (uint256 i = 0; i < matchCount; i++) {
335         allMatches[i] = _matches[i];
336     }
337     return allMatches;
338 }
```

```
339
340 function isMatchStarted(uint256 _matchId) external view returns (bool) {
341     require(_matchId >= 0 && _matchId < matchCount, "not found matchId");
342     Match memory _match = _matches[_matchId];
343     return block.timestamp > _match.startTime;
344 }
345
346 function isMatchEnded(uint256 _matchId) external view returns (bool) {
347     require(_matchId >= 0 && _matchId < matchCount, "not found matchId");
348     Match memory _match = _matches[_matchId];
349     return _match.isEnded;
350 }
```

```
353 function setRewardPoolAddress(address account) public onlyOwner {
354     rewardPoolAddress = account;
355 }
356
357 function setSFS(address addr) public onlyOwner {
358     SFS = addr;
359 }
360
361 function setMinPredictRequire(uint256 amount) public onlyOwner {
362     minPredictRequire = amount;
363 }
364
365 function getUserBalance(address user) external view returns (uint256) {
366     return IERC20(SFS).balanceOf(user);
367 }
368
369 function getTotalReward() external view returns (uint256) {
370     return IERC20(SFS).balanceOf(rewardPoolAddress);
371 }
```

```
373 function getCurrentUserReward(address user) external view returns (uint256) {
374     uint256 totalHold = IERC20[SFS].balanceOf(user);
375     uint256 totalSupply = IERC20[SFS].totalSupply();
376     uint256 totalReward = IERC20[SFS].balanceOf(rewardPoolAddress);
377
378     return totalReward.div(totalSupply.div(totalHold)+1) + totalReward.div(totalSupply.mod(totalHold)+1);
379 }
380
381 function isEnoughSFS(address user) external view returns(bool) {
382     return IERC20[SFS].balanceOf(user) > minPredictRequire;
383 }
384
385 }
```

Automated Audit

Remix Compiler Warnings

It throws warnings by Solidity's compiler. No issues found.

Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. To get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us based on what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the below disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and ContractChecker and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (ContractChecker) owe no duty of care towards you or any other person, nor does ContractChecker make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and ContractChecker hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, ContractChecker hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against ContractChecker, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report.

The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. If you have any doubt about the Genuity for this document, please check QR code:

