DeHacker

Code Security Assessment

Polkamarkets

Aug 10th, 2023

Contents

Summary

DeHacker's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.
Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow/underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service/logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Issue Categories

Every issue in this report was assigned a severity level from the following:

Critical severity issues

A vulnerability that can disrupt the contract functioning in a number of scenarios or creates a risk that the contract may be broken.

Major severity issues

A vulnerability that affects the desired outcome when using a contract or provides the opportunity to use a contract in an unintended way.

Medium severity issues

A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.

Minor severity issues

A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.

Informational

A vulnerability that has informational character but is not affecting any of the code.

Overview

Project Summary

Project Name	Polkamarkets
Platform	Ethereum
Website	https://www.polkamarkets.com/
Type	Defi
Language	Solidity

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	0	0	0	0	0	0
Major	1	0	0	1	0	1
Medium	0	0	0	0	0	0
Minor	1	0	0	0	0	1
Informational	3	0	0	2	0	1
Discussion	0	0	0	0	0	0

Audit scope

ID	File	SHA256 Checksum
PMK	PredictionMarket.sol	75eb20df81ad4a9b9f4ecad66e4e83556d4fa03856ceffc91be9fedc3fbe12d5

Findings

ID	Category	Severity	Status
Polkamarket	Data Flow	Informational	Acknowledged
PMK-01	Volatile Code	Informational	Resolved
PMK-02	Coding Style	Informational	Acknowledged
PMK-03	Volatile Code	Major	Acknowledged
PMK-04	Logical Issue	Minor	Resolved

INFORMATIONAL

Polkamarkets-01 | Financial Models

Category	Severity	Location	Status
Data Flow	Informational	Global	Acknowledged

Description

The main functions of the Prediction protocol can be listed as follows:

1. User can pay some eth tokens to create Market . This market is used to predict the outcome of some events.
2. Users can provide liquidity to the market. The liquidity provider can get fees when users buy or sellthe shares of the outcome they believe.
3. There are only two outcomes of an event in the Market . Users can buy or sell the shares of theoutcome in the Market .
4. Once the deadline has passed and the definite outcome of the question has come out, the marketwill close.
5. After the deadline, the winner can get their principal and rewards. And the loser will lost everything.The liquidity provider also can get their liquidity fund and rewards if the market has.

However, there is a question:
1. In the function removeLiquidity() , liquidity providers' rewards are from the fewer one. However,the fewer shares may be less than the amount staked by liquidity providers. It may cause a loss tothe liquidity provider. Does this match the design intention?

Recommendation

Financial models of blockchain protocols need to be resilient to attacks. They need to pass simulationsand verifications to guarantee the security of the overall protocol.The financial model of this protocol is not in the scope of this audit.

INFORMATIONAL

CKIK-02 | Redundant Code

Category	Severity	Location	Status
Volatile Code	Informational	PredictionMarket.sol: 171	Resolved

Description

In the constructor, the input variables realitioAddress should not be zero address.

Recommendation

We advise the client to check that the aforementioned variable is not zero address.

```
161 constructor(
162     uint256 _fee,
163     IERC20 _token,
164     uint256 _requiredBalance,
165     address _realitioAddress,
166     uint256 _realitioTimeout
167 ) public {
168     require(_realitioAddress != 0, "_realitioAddress is address 0");
169     fee = _fee;
170     token = _token;
171     requiredBalance = _requiredBalance;
172     realitioAddress = _realitioAddress;
173     realitioTimeout = _realitioTimeout;
174 }
```

INFORMATIONAL

PMK-02 | Unlocked Compiler Version

Category	Severity	Location	Status
Coding Style	Informational	PredictionMarket.sol: 1	Acknowledged

Description

We do not recommend using unlocked versions of solidity for deployment.

Recommendation

Deploy with any of the following Solidity versions:
0.5.16 - 0.5.17
0.6.11 - 0.6.12
0.7.5 - 0.7.6
Use a simple pragma version that allows any of these versions. Consider using the latest version of Solidityfor testing.

MAJOR

PMK-03 | Third Party Dependencies

Category	Severity	Location	Status
Volatile Code	Major	Prediction Market. sol: 499	Acknowledged

Description

The contract is serving as the underlying entity to interact with the third-party RealitioERC20 contract.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. Inaddition, upgrades of third parties can possibly create severe impacts, such as increasing fees of thirdparties, migrating to new LP pools, etc. And the result of the function resultFor() decides the winner ofthe question of the market . It will affect the benefits of all buyers.

Recommendation

We understand that the business logic of PredictionMarket requires interaction with RealitioERC20 . Weencourage the team to constantly monitor the statuses of third parties to mitigate the side effects whenunexpected activities are observed.

MINOR

PMK-04 | Possible Incorrect Calculation

Category	Severity	Location	Status
Logical Issue	Minor	PredictionMarket. sol: 417~418, 865	Resolved

Description

Function getMarketLiquidityPrice is used to obtain earnings by liquidity per share.The output result of line 875 is conflicting with the line 871 if statement result. And this result isincorrectly used in line 418.

```
417 uint256 liquidityPrice = getMarketLiquidityPrice(marketId);
418 uint256 liquidityValue = liquidityPrice.mul(liquidityAmount) / ONE;
```

```
865 function getMarketLiquidityPrice(uint256 marketId) public view returns (uint256) {
866   Market storage market = markets[marketId];
867
868   if (market.state == MarketState.resolved && !isMarketVoided(marketId)) {
869     // resolved market, price is either 0 or 1
870     // final liquidity price = outcome shares / liquidity shares
871     return
market.outcomes[market.resolution.outcomeId].shares.available.mul(ONE).div(market.liquidi
ty);
872   }
873
874   // liquidity price = # liquidity shares / # outcome shares * # outcomes
875   return market.liquidity.mul(ONE *
market.outcomeIds.length).div(market.sharesAvailable);
876 }
```

Recommendation

Please review the design intention and do more tests on the related functions.

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Coding Style

Coding Style findings usually do not affect the generated bytecode but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block. timestamp works.

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.

About

DeHacker is a team of auditors and white hat hackers who perform security audits and assessments. With decades of experience in security and distributed systems, our experts focus on the ins and outs of system security. Our services follow clear and prudent industry standards. Whether it's reviewing the smallest modifications or a new platform, we'll provide an in-depth security survey at every stage of your company's project. We provide comprehensive vulnerability reports and identify structural inefficiencies in smart contract code, combining high-end security research with a real-world attacker mindset to reduce risk and harden code.

BLOCKCHAIINS

- Ethereum
- Cosmos
- Eos
- Substrate

TECH STACK

- Python
- Solidity
- Rust
- c++

CONTACTS

- https://dehacker.io
- https://twitter.com/dehackerio
- https://github.com/dehacker/audits_public
- https://t.me/dehackerio
- https://blog.dehacker.io/

Aug 2023