soken

SMART CONTRACT SECURITY AUDIT

MetaShooter

Scan and check this report
was posted at Soken Github

June, 2022

Website: soken.io

Table of Contents

Disclaimer

This is a comprehensive report based on our automated and manual examination of cybersecurity vulnerabilities and framework flaws. We took into consideration smart contract based algorithms, as well. Reading the full analysis report is essential to build your understanding of project's security level. It is crucial to take note, though we have done our best to perform this analysis and report, that you should not rely on the our research and cannot claim what it states or how we created it. Before making any judgments, you have to conduct your own independent research. We will discuss this in more depth in the following disclaimer - please read it fully.

DISCLAIMER: You agree to the terms of this disclaimer by reading this report or any portion thereof. Please stop reading this report and remove and delete any copies of this report that you download and/or print if you do not agree to these conditions. This report is for non-reliability information only and does not represent investment advice. No one shall be entitled to depend on the report or its contents, and Soken and its affiliates shall not be held responsible to you or anyone else, nor shall Soken provide any guarantee or representation to any person with regard to the accuracy or integrity of the report. Without any terms, warranties or other conditions other than as set forth in that exclusion and Soken excludes hereby all representations, warrants, conditions and other terms (including, without limitation, guarantees implied by the law of satisfactory quality, fitness for purposes and the use of reasonable care and skills). The report is provided as "as is" and does not contain any terms and conditions. Except as legally banned, Soken disclaims all responsibility and responsibilities and no claim against Soken is made to any amount or type of loss or damages (without limitation, direct, indirect, special, punitive, consequential or pure economic loses or losses) that may be caused by you or any other person, or any damages or damages, including without limitations (whether innocent or negligent).

Security analysis is based only on the smart contracts. No applications or operations were reviewed for security. No product code has been reviewed.

Procedure

Our analysis contains following steps:

1. Project Analysis;

2. Manual analysis of smart contracts:
- Deploying smart contracts on any of the network(Ropsten/Rinkeby) using Remix IDE
- Hashes of all transaction will be recorded
- Behaviour of functions and gas consumption is noted, as well.

3. Unit Testing:
- Smart contract functions will be unit tested on multiple parameters and under multiple conditions to ensure that all paths of functions are functioning as intended.
- In this phase intended behaviour of smart contract is verified.
- In this phase, we would also ensure that smart contract functions are not consuming unnecessary gas.
- Gas limits of functions will be verified in this stage.

4. Automated Testing:
- Mythril
- Oyente
- Manticore
- Solgraph

Terminology

We categorize the finding into 4 categories based on their vulnerability:

- Low-severity issue — less important, must be analyzed
- Medium-severity issue — important, needs to be analyzed and fixed
- High-severity issue —important, might cause vulnerabilities, must be analyzed and fixed
- Critical-severity issue —serious bug causes, must be analyzed and fixed.

Limitations

The security audit of Smart Contract cannot cover all vulnerabilities. Even if no vulnerabilities are detected in the audit, there is no guarantee that future smart contracts are safe. Smart contracts are in most cases safeguarded against specific sorts of attacks. In order to find as many flaws as possible, we carried out a comprehensive smart contract audit. Audit is a document that is not legally binding and guarantees nothing.

Token Contract Details for 21.06.2022

Contract Name: **MetaShooterNFT**

Deployed address: **0x8F7A57125E23E7e4E6724D20dc8907a9E18D94A5**

Total Supply: **1**

Token Tracker: **MHUNT NFT**

Token holders: **1**

Transactions count: **1**

Top 100 holders dominance: **100%**

Audit Details

Project Name: **MetaShooter**

Language: **Solidity**

Compiler Version: **v0.8.9**

Blockchain: **BSC**

Social Profiles

Project Website: **https://metashooter.gg/**

Project Twitter: **https://twitter.com/MetaShooter_gg**

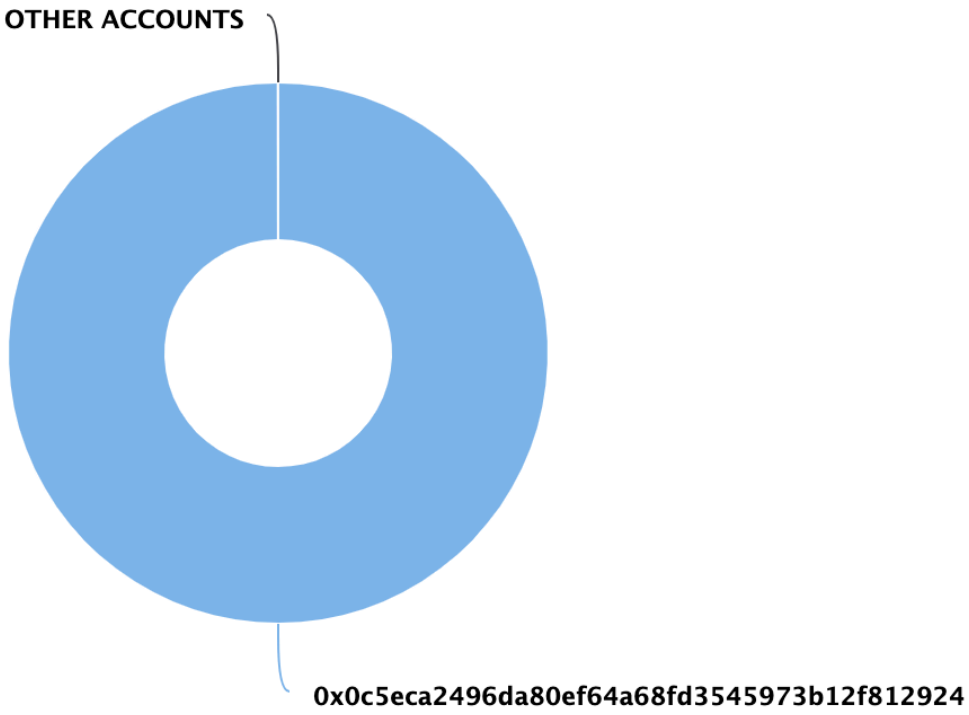
Project Telegram: **https://t.me/METASHOOTER_GG**

Project Linkedin: **https://www.linkedin.com/company/metashooter/**

Project Facebook: **https://www.facebook.com/MetaShooter.gg/**

Project Youtube: **https://www.youtube.com/channel/UC82FpbcY_C05XqQYaJobFyQ**

MHUNT NFT Token Distribution



MHUNT NFT Top Holders

Rank	Address	Quantity (Token)	Percentage
1	0x0c5eca2496da80ef64a68fd3545973b12f812924	1	100.0000%

Vulnerabilities checking

Issue Description	Checking Status
Compiler Errors	Completed
Delays in Data Delivery	Completed
Re-entrancy	Completed
Transaction-Ordering Dependence	Completed
Timestamp Dependence	Completed
Shadowing State Variables	Completed
DoS with Failed Call	Completed
DoS with Block Gas Limit	Completed
Outdated Complier Version	Completed
Assert Violation	Completed
Use of Deprecated Solidity Functions	Completed
Integer Overflow and Underflow	Completed
Function Default Visibility	Completed
Malicious Event Log	Completed
Math Accuracy	Completed
Design Logic	Completed
Fallback Function Security	Completed
Cross-function Race Conditions	Completed
Safe Zeppelin Module	Completed

Security Issues

1) Loop consuming excessive gas: Low-severity

```
177     function reservedItemsOfOwner(address _owner) public view returns (uint32[] memory) {
178         uint32 itemsCount = 0;
179         for (uint32 i = 0; i < items.length; i++) {
180             if (_reservedTokens[_owner][i] > 0){
181                 itemsCount++;
182             }
183         }
184
185         uint32[] memory ownedItemIds = new uint32[](itemsCount);
186         uint32 j = 0;
187         for (uint32 i = 0; i < items.length; i++) {
188             if (_reservedTokens[_owner][i] > 0){
189                 ownedItemIds[j] = i;
190                 j++;
191             }
```

If items.length is large enough, the function exceeds the block gas limit, and transactions calling it will never be confirmed.

Recommendation:

Either explicitly or just due to normal operation, the number of iterations in a loop can grow beyond the block gas limit, which can cause the complete contract to be stalled at a certain point. Therefore, loops with a bigger or unknown number of steps should always be avoided.

2) Use of Floating Pragma: Low-severity

Solidity source files indicate the versions of the compiler they can be compiled with using a pragma directive at the top of the solidity file. This can either be a floating pragma or a specific compiler version. The contract was found to be using a floating pragma which is not considered safe as it can be compiled with all the versions described.

Recommendation:

It is recommended to follow the latter example, as future compiler versions may handle certain language constructions in a way the developer did not foresee. The developers should always use the exact Solidity compiler version when designing their contracts as it may break the changes in the future.

Conclusion

Low-severity issues exist within smart contracts. Smart contracts are free from any critical or high-severity issues.

NOTE: Please check the disclaimer above and note, that audit makes no statements or warranties on business model, investment attractiveness or code sustainability.

Soken Contact Info

Website: www.soken.io

Mob: (+1)416-875-4174

32 Britain Street, Toronto, Ontario, Canada

Telegram: @team_soken

GitHub: sokenteam

Twitter: @soken_team