DeHacker

Code Security Assessment

SONET TOKEN

September 3rd, 2024

Contents

Summary

DeHacker's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best prac tices. 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	GAMEE
Platform	Ethereum
Website	sonet.one/
Type	GameFi
Language	Solidity
Codebase	https://github.com/platwin/sonet-token/tree/59722b08ee1051274404ce55f52d50867f19873d

Vulnerability Summary

Vulnerability Level	Total	Mitigated	Declined	Acknowledged	Partially Resolved	Resolved
Critical	0	0	0	0	0	0
Major	2	0	0	2	0	0
Medium	2	0	0	1	0	1
Minor	4	0	0	4	0	0
Informational	2	0	0	2	0	0
Discussion	0	0	0	0	0	0

Audit scope

ID	File	SHA256 Checksum
MIN	contracts/mining	
IRR	contracts/mining/IRewardRelease.so	2374c385ac7606025fc645547b901747c01c5e422cb891ff877635ecf7e779af
LMB	contracts/mining/LiquidityMining.sol	9ab335a460984013506cc26e4449d70675ed3e6e98f733913c482c4623272ee1
ORB	contracts/mining/OwnableRound.sol	98bcbab1c19b8723795a253c7a0169082348e945b839fdcf77c6da5c0db3ec04
SON	contracts/mining/SONRelease.sol	ecfe6f596a148ef4c1c0818ef14ad78f1b817e685a725e9aaa2a401b880ce6b8
ASS	contracts/assets	
ISN	contracts/assets/ISoNetAsset.sol	499ea4a7e45ca835571a39ecb5315f3d929f810d78a7011ba6d9d125af1c51c0
SOA	contracts/assets/SON.sol	0833f3d50bcbcba3aaa215a5f5541bb2e435993a591ef82402498d5ef26a0409
EXP	contracts/utils/Exponential.sol	51eb23f4bab4e72d658b0ab5e30df73c4a2436ab43419ccfaa1beaa16cd531f7
RES	contracts/Reserves.sol	0cbc89f2f947cb625ace169ddd744b512a7f9d6c59b00bbfd521c102694493a3

Findings

ID	Title	Severity	Status
SON-01	Lack Of Storage Gap In Upgradeable Contract	Medium	Acknowledged
SON-02	Third Party Dependencies And Potential Reentrancy	Medium	Resolved
CON-01	Unused Return Value	Minor	Acknowledged
CON-02	Missing Emit Events	Informational	Acknowledged
LMB-01	Centralization Risk	Major	Acknowledged
LMB-02	Incompatibility With Deflationary Tokens	Minor	Acknowledged
LMB-03	No Check On Duplicate StakingAsset In Pools	Minor	Acknowledged
LMB-04	External Call Inside Loop	Informational	Acknowledged
SOA-01	Initial Token Distribution	Major	Acknowledged
SOR-01	Missing Zero Address Validation	Minor	Acknowledged

MEDIUM

SON-01| Lack Of Storage Gap In Upgradeable Contract

Issue	Severity	Location	Status
Volatile Code	Medium		Acknowledged

Description

For upgradeable contracts, there must be storage gap to "allow developers to freely add new state variablesin the future without compromising the storage compatibility with existing deployments". Otherwise it may bevery difficult to write new implementation code. Without storage gap, the variable in child contract might beoverwritten by the upgraded base contract if new variables are added to the base contract.

Refer to https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable

Recommendation

We advise the client to add appropriate storage gap at the end of upgradeable contracts such as:

```
uint256[[5050]] private __gap
```

MEDIUM

SON-02| Third Party Dependencies And Potential Reentrancy

Issue	Severity	Location	Status
Volatile Code	Medium		Resolved

Description

The contract is serving as the underlying entity to interact with different ERC20 tokens both for staking andfor reward payment. Each ERC20 token could have its own implementation of important functions such as transferFrom() . The scope of the audit treats 3rd party entities as black boxes and assume their functionalcorrectness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolenassets. In addition, upgrades of 3rd parties can possibly create severe impacts.

As an example, the distributeReward function in the LiquidityMining contract transfers rewardAsset toexternal address before updating state.rewardAccrued . A reentrancy from the external user address ispossible if the safeTransfer function in line 125 is not implemented properly, causing rewardAccrued to besent to user multiple times before the state variable is updated to 0 in line 126.

```
125 releaser.rewardAssetrewardAsset()..safeTransfer(user, state.rewardAccrued);
126 state.rewardAccrued = 0;
```

Additionally, the code base contains the implementation contract of several upgradeable contracts. Theproxy contract behind the implementation contracts is not in scope for the audit, and the role that controlsthe upgrade functionality cannot be assessed. Inadequate access control or incorrect contract upgradecould have serious consequences to the project overall.

Recommendation

We understand that the business logic of this project requires interaction with different ERC20 tokens etc.We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects whenunexpected activities are observed. Additionally, we recommend adding the OpenZeppelin nonReentrantmodifier for functions that interact with external addresses, where the check-effect-interaction pattern is notstrictly followed.

MINOR

CON-01 | Unused Return Value

Issue	Severity	Location	Status
Volatile Code	Minor	$/github/platwin/sonet-token/59722b08ee1051274404ce55f52d50867f19873d/contracts/Reserves.sol (Client GitHub): 43, 58; $/github/platwin/sonet-token/59722b08ee1051274404ce55f52d50867f19873d/contracts/tests/MockRewardRelease.sol (Client GitHub): 24	Acknowledged

Description

The return value of an external call is not stored in a local or state variable, or checked for success in casewhere the return value is a boolean variable.

File: contracts/Reserves.sol (Line 43, Function Reserves.buyback)

```
IERC20(path[00]).approve(address(univ2Router),amountIn);
```

File: contracts/Reserves.sol (Line 58, Function Reserves.recycle)

```
IERC20(path[00]).approve(address(univ2Router),amountIn);
```

File:contracts/tests/MockRewardRelease.sol (Line 24, Function MockRewardRelease.getToken

```
IMintableSoNetAsset(address(rewardAsset)).mint(msg.sender, mintAmountPerInvoke);
```

Recommendation

We recommend checking or using the return values of all external function calls.

INFORMATIONAL

CON-02| Missing Emit Events

Issue	Severity	Location	Status
Coding Style	Informational	$/github/platwin/sonet-token/59722b08ee1051274404ce55f52d50867f19873d/contracts/Reserves.sol (Client GitHub): 64; $/github/platwin/sonet-token/59722b08ee1051274404ce55f52d50867f19873d/contracts/mining/LiquidityMining.sol (Client GitHub): 52, 61; $/github/platwin/sonet-token/59722b08ee1051274404ce55f52d50867f19873d/contracts/mining/SONRelease.sol (Client GitHub): 49, 61	Acknowledged

Description

There should always be events emitted in the sensitive functions that are controlled by centralization roles.

Recommendation

It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

MAJOR

LMB-02| Centralization Risk

Issue	Severity	Location	Status
Centralization / Privilege	Major	$/github/platwin/sonet-token/59722b08ee1051274404ce55f52d50867f19873d/contracts/mining/LiquidityMining.sol (Client GitHub): 52, 61	Acknowledged

Description

In the contract LiquidityMining the role _owner has authority over the functions. In the contract Reserves the role _owner has authority over the functions. In the contract SONRelease the role _owner has authority over the functions. In the contract SONRelease the role sonMining has authority over the functions.

Any compromise to the _owner or sonMining account may allow the hacker to take advantage of thisauthority and alter critical parameters of the project, such as adjusting pool weight, or changing the

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the securityoperation and level of decentralization, which in most cases cannot be resolved entirely at the presentstage. We advise the client to carefully manage the privileged account's private key to avoid any potentialrisks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol beimproved via a decentralized mechanism or smart-contract-based accounts with enhanced securitypractices, e.g., multisignature wallets.

Recommendation

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different levelin 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 asingle 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 theprivate 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 informationwith 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

MINOR

LMB-02| Incompatibility With Deflationary Tokens

Issue	Severity	Location	Status
Logical Issue	Minor	contracts/mining/LiquidityMining.sol (Client GitHub): 95~101, 103~109	Acknowledged

Description

When transferring standard ERC20 deflationary tokens, the input amount may not be equal to the receivedamount due to the charged transaction fee.

For example, if a user deposits 100 deflationary tokens (with a 10% transaction fee) into the LiquidityMiningcontract, only 90 tokens actually arrive. However, the contract still uses 100 tokens when calculating state.balance and pool.totalAmount

```
function deposit(uint poolId, uint amount) public poolExisted(poolId) {
(Pool storage pool, UserState storage state) = distributeReward(poolId,
msg.sender);
state.balance += amount;
pool.totalAmount += amount; /
pool.stakingAsset.safeTransferFrom(msg.sender, address(this), amount); }
function withdraw(uint poolId, uint amount) public poolExisted(poolId) {
(Pool storage pool, UserState storage state) = distributeReward(poolId,
msg.sender);
state.balance -= amount;
pool.totalAmount -= amount;
pool.stakingAsset.safeTransfer(msg.sender, amount);
}
```

Recommendation

We advise the client to regulate the set of stakingAsset tokens supported and add necessary mitigationmechanisms to keep track of accurate balances if there is a need to support deflationary tokens.

MINOR

LMB-03| No Check On Duplicate StakingAsset In Pools

Issue	Severity	Location	Status
Logical Issue	Minor	contracts/mining/LiquidityMining.sol (Client GitHub): 52~58	Acknowledged

Description

The addPool function does not check if the stakingAsset being added already exists in the pool. Thus it ispossible that the same stakingAsset gets added more than once to the pools, with potentially differentweights.

Recommendation

We recommend including a check that the same stakingAsset can only have one unique poolId

INFORMATIONAL

LMB-04| External Call Inside Loop

Issue	Severity	Location	Status
Control Flow	Informational	$/github/platwin/sonet-token/59722b08ee1051274404ce55f52d50867f19873d/contracts/mining/LiquidityMining.sol (Client GitHub): 76, 112~114, 124, 125	Acknowledged

Description

External calls are made inside a for loop in the claimAll() function. This might lead to a denial-of-serviceattack. If any of the calls fail, it will cause the entire loop to revert.

Recommendation

We recommend using the pull-over-push strategy for external calls.

MAJOR

SOA-01| Initial Token Distribution

Issue	Severity	Location	Status
Centralization /Privilage	Major	contracts/assets/SON.sol (Client GitHub): 12~13	Acknowledged

Description

All of the SON tokens are sent to deployer designated addresses (0xD2C7133BD40bBdAf7805fDF6d5865c4Cd51ed41d) when deploying the SocialNetworkToken contract.This could be a centralization risk as the deployer designated addresses can distribute SON tokens withoutobtaining the consensus of the community

```
1 _mint(0xD2C7133BD40bBdAf7805fDF6d5865c4Cd51ed41d,2_000_000_000 * (10 **18));
```

Recommendation

We recommend the team to be transparent regarding the initial token distribution process, and the teamshall make enough efforts to restrict the access of the private key.

MINOR

SOR-01| Missing Zero Address Validation

Issue	Severity	Location	Status
Volatile Code	Minor	$/github/platwin/sonet-token/59722b08ee1051274404ce55f52d50867f19873d/contracts/mining/SONRelease.sol (Client GitHub): 35, 36	Acknowledged

Description

Addresses should be checked before assignment or external call to make sure they are not zero addresses.
File: contracts/mining/SONRelease.sol (Line 35, Function SONRelease.initialize)

```
reserve =_reserve;
```

- _reserve is not zero-checked before being used.

File: contracts/mining/SONRelease.sol (Line 36, Function SONRelease.initialize)

```
sonMining =_sonMining;
```

- _sonMining is not zero-checked before being used

Recommendation

We advise adding a zero-check for the passed-in address value to prevent unexpected errors.

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/

September 3rd 2024