DeHacker

Code Security Assessment

THE SANDBOX

July 13th, 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	THE SANDBOX
Platform	ETHEREUM
website	https://www.sandbox.game/en
Type	LAND Bridge
Language	solidity
Codebase	https://github.com/thesandboxgame/sandbox-smart-contracts-private

Vulnerability Summary

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

Audit scope

ID	File	SHA256 Checksum
LBT	Land/erc721/LandBaseToken.sol	3f970ad1e573cf55bffedf14575e317f6125ebb64a0c31f7d6b8f1b7e5fe8090

Findings

ID	Issue	Severity	Status
GLOBAL-01	Centralization Related Risks	Major	Acknowledged
LBT-01	Inconsistent NatSpec For Minting Function	Informational	Resolved
LBT-02	Potential Denial-Of-Service Attack	Informational	Resolved
LBT-03	Unable To Mint Burnt Tokens	Informational	Resolved

MAJOR

GLOBAL-01|Centralization Related Risks

Issue	Severity	Location	Status
Centralization /Privilege	Major		Acknowledged

Description

In the contract LandBaseToken , the role _admin has authority over the following functions:

- setMinter() : Decide if an address is a minter or not;

In addition, the minter role has authority over the following function:

- mintQuad() : mints quads to an address.

Also, the roles superOperator and _metaTransactionContracts have authority over the following functions:

- transferQuad() : Transfer any user’s quads to an address;
- batchTransferQuad() : Transfer any user’s quads to an address.

The contract LandBaseToken inherits the contract ERC721BaseToken , where _admin has authority over the following functions:

- setMetaTransactionProcessor() : Give or remove the _metaTransactionContracts role to or from an address;

Description

- setSuperOperator() : Give or remove the superOperator role to or from an address;
- changeAdmin() : Change the address of the role _admin .

In addition, the superOperator role has authority over the following functions:

- approveFor() : Decide the allowance of any token;
- approve() : Decide the allowance of any token;
- transferFrom() : Transfer any user's tokens to an address;
- safeTransferFrom() : Transfer any user's tokens to an address;
- batchTransferFrom() : Transfer several of a user's tokens to an address;
- safeBatchTransferFrom() : Transfer several of a user's tokens to an address;
- setApprovalForAllFor() : Set the approval for an address to manage all of a user's tokens;
- burnFrom() : Burn any user's tokens.

Furthermore, the _metaTransactionContracts role has authority over the following functions:

- approveFor() : Decide the allowance of any token;
- transferFrom() : Transfer any user's tokens to an address;
- safeTransferFrom() : Transfer any user's tokens to an address;

Description

- batchTransferFrom() : Transfer several of a user’s tokens to an address;
- safeBatchTransferFrom() : Transfer several of a user’s tokens to an address;
- setApprovalForAllFor() : Set the approval for an address to manage all of a user’s tokens;
- burnFrom() : Burn any user’s tokens.

Any compromise to the aforementioned privileged accounts may allow a hacker to take advantage of this authority and manipulate the reward system.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level ofdecentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefullymanage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommendcentralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accountswith enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in 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 a single point of keymanagement 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 the private keycompromised; AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the publicaudience.

Recommendation

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 information with the publicaudience.

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.

Noted: Recommend considering the long-term solution or the permanent solution. The project team shall make a decisionbased on the current state of their project, timeline, and project resources

INFORMATIONAL

LBT-01|Inconsistent NatSpec For Minting Function

Issue	Severity	Location	Status
Inconsistency	Informational	Land/erc721/LandBaseToken.sol: 68	Resolved

Description

The NatSpec of the `mintQuad()` function states that mints a quad of size 3, 6, 12, or 24 only. However, the function allows mints of size 1.

```
    function mintQuad(address to, uint256 size, uint256 x, uint256 y, bytes calldata data) external {
        require(to != address(0), "to is zero address");
        require(
            isMinter(msg.sender),
            "Only a minter can mint"
        );
        require(x % size == 0 && y % size == 0, "Invalid coordinates");
        require(x <= GRID_SIZE - size && y <= GRID_SIZE - size, "Out of bounds");

        uint256 quadId;
        uint256 id = x + y * GRID_SIZE;

        if (size == 1) {
            quadId = id;
```

Recommendation

We recommend changing either the NatSpec or the code of `mintQuad()` so that both are consistent with each other.

INFORMATIONAL

LBT-02|Potential denial-Of-Service Attack

Issue	Severity	Location	Status
Logical Issue	Informational	Land/erc721/Land BaseToken.sol: 75	Resolved

Description

When the contract tries to mint a quad via _mintQuad() , there is a check to ensure that no quads containing the quad tomint and no quads (or LANDS) within the quad to mint have already been minted by calling the function exists() .This leads to a possible denial-of-service attack where the attacker mints 1x1 LANDS at specific locations to prevent theminting of larger quads. For example, out of the possible 166,464 LAND placements, only 289 LANDS need to be minted toprevent 24x24 quads from occurring.

Recommendation

We recommend only allowing mints of langer quads if this is not intended.

INFORMATIONAL

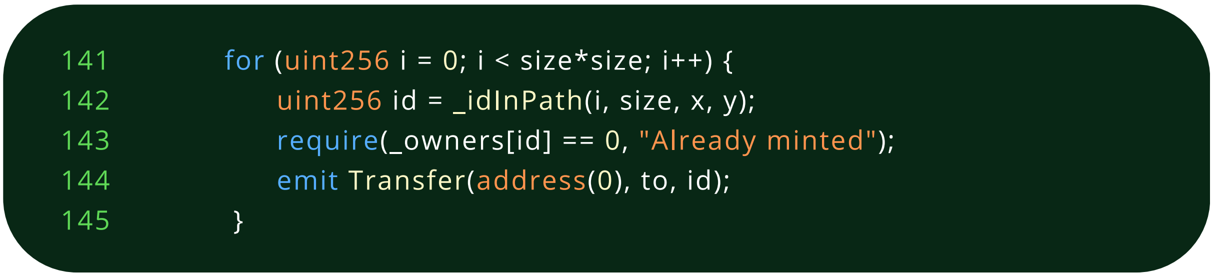
LBT-03|Unable To Mint Burnt Tokens

Issue	Severity	Location	Status
Logical Issues	Informational	Land/erc721/LandBaseToken.sol: 143	Resolved

Description

Minting of tokens on L1 is done by the function `mintQuad()` , which checks whether the tokens already exist or not. In detail, the `mintQuad()` function checks the owner of each token ID to decide whether or not the token has already been minted.

For example, for 1x1 LAND tokens, it checks to see if the owner is non-zero



```
for (uint256 i = 0; i < size*size; i++) {
     uint256 id = _idInPath(i, size, x, y);
     require(_owners[id] == 0, "Already minted");
     emit Transfer(address(0), to, id);
 }
```

However, for burnt tokens, their value in the `_owners` mapping is non-zero due to the burning flag. This means that burnttokens cannot be minted again.

Description

Proof of Concept

```
it('Burnt land cannot be minted again', async function () {
  const {
     landContract,
     getNamedAccounts,
     ethers,
     mintQuad,
   } = await setupLand();
   const {deployer, landAdmin} = await getNamedAccounts();
   const contract = landContract.connect(ethers.provider.getSigner(deployer));
   const x = 0;
   const y = 0;
   await mintQuad(deployer, 3, x, y);
   const tokenId = x + y * GRID_SIZE;
   await contract.burn(tokenId);
   await expect(mintQuad(deployer, 1, x, y)).to.be.revertedWith('Already minted as
3x3');
 });
```

The test passed:

```
✓ Burnt land cannot be minted again (64ms)
```

From the result, the burnt token is unable to be minted again.

Recommendation

We recommend allowing burnt tokens to be minted again if this is unintentional.

OPTIMIZATION

LBT-04|Rebundant Code

Issue	Severity	Location	Status
Gas Optimization	Optimization	Land/erc721/LandBaseToken.sol: 327	Resolved

Description

The internal function `_ownerOfQuad()` is used to find the owner or parent owner of a quad and can only be called by itself orone of the regroup functions. As none of these will call `_ownerOfQuad()` with a value of 1 for the input variable `size` , the `if` code branch `size == 1` will never be reached.

Recommendation

We recommend removing the branch `size == 1` .

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
- Eos
- Cosmos
- Substrate

TECH STACK

- Python
- Rust
- Solidity
- c++

CONTACTS

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

DeHacker

July 13th 2024