QuillAudits

Audit Report
July, 2024

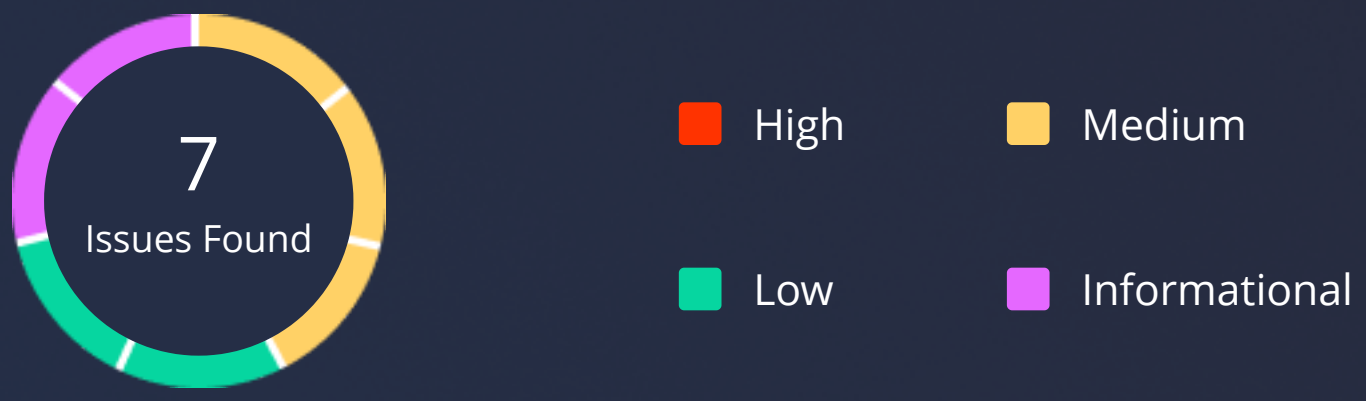
For

Table of Content

Executive Summary

Project Name	AntiGravity
Overview	• DarkX - ERC20 Token with open transfer • MiningRig - Smart contract to mine DarkX token • Dark - ERC20 token • DarkXlaims - Claim Dark Tokens using Merkle Proofs of user mining activity of DarkX token and not their current DarkX token balance.
Timeline	19th June 2024 - 17th July 2024
Update Code Received	9th July
Second Review	11th July 2024 - 17th July 2024
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Audit Scope	The scope of this audit was to analyze the Antigravity contracts for quality, security, and correctness.
Source Code	https://github.com/chain-labs/antigravity-core
Branch	Main
Commit Hash	c8d96b50bfb33d990c0cc6ded099003b8d750a6
Fixed In	https://github.com/chain-labs/antigravity-core/pull/68

Number of Security Issues per Severity



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	1	1	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	2	1	2

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC-20's standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Hardhat, Foundry.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four severity levels, each of which has been explained below.

High Severity Issues

A high severity issue or vulnerability means your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impacts and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These issues were identified in the initial audit and successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Medium Severity Issues

1. brutalized function does not do anything and can be removed

Path

DarkX.sol

Function

_brutalized()

Description

The brutalized function according to inline comments manipulated the input address using current values.

It takes an address and modifies it using low level calls and modifies it by performing a bitwise OR operation with a value derived from the current remaining gas in the transaction, shifted left by 160 bits.

However, this function serves no purpose since the final address returned is exactly similar to the input.

This can be confirmed by testing this function independently and comparing input and output address.

Recommendation

It is recommended to remove this function as it serves no purpose but to increase computation.

Status

Acknowledged

AntiGravity Team's Comment

This was added to give more personality to DarkX. that anyone can brutalize an address. We would like to keep the function while understanding brutalized function adds extra gas and computation.

2. Possibility of DOS due to overflow

Path

DarkClaims.sol

Function

claim()

Description

The function claim takes and array of addresses, amounts , nonces and their merkle proofs and an input parameter and mints calculated amount to those addresses as a result.It does so by traversing the entire array of addresses and minting tokens for each one individually.Since there is no upper bound in the for loop, there is a chance that this loop exceeds the block gas limit, resulting in a state of Denial of Service.
Note that the arrays can be artificially inflated by putting null entries into them.

Recommendation

Ensure an upper bound over the iteration and add an address(0) check before the loop.

Status

Resolved

Fixed In

https://github.com/chain-labs/antigravity-core/pull/68/commits/765cee3dd5169f9f3033a76584ecce62fbb2c4b1

3. claim function does not work as intended

Path

DarkClaims.sol

Function

claim

Description

According to the documentation(specs_token.md), if a user claims N amount of dark tokens, 10% of it should go to Mystery box address while the remaining 90% should go to the user.

However, the function fails to do so where the user amount is 10%, followed by the mystery account, which is 25%, and the remaining amount goes to ‘eviladdress’ specified in the constructor.

```
// Calculate the amount to be sent to the mystery address.
uint256 userAmount = (amountToBeMinted * 10) / 100;
uint256 mysteryAmount = (amountToBeMinted * 25) / 100;
uint256 evilAmount = amountToBeMinted - userAmount - mysteryAmount;
dark.mint(addresses[i], userAmount);
dark.mint(mysteryBox, mysteryAmount);
dark.mint(evilAddress, evilAmount);
```

Recommendation

Remove these inconsistencies by syncing the docs and the code.

Status

Resolved

Fixed In

https://github.com/chain-labs/antigravity-core/pull/68/commits/eae632785d056817c49fd93318f6e8490250828a

Low Severity Issues

1. Transfer token without user approval

Path

DarkX.sol

Description

The DarkX Contract is an extension of ERC20 contract that is used by the miningRig contract to mint new tokens. However, it has some overridden functions like directTransfer and directSpendAllowance that allow any user to transfer the tokens from any user acount to their account without the token holder approval or consent.

```
function directTransfer(address from, address to, uint256 amount) public virtual {
    _transfer(_brutalized(from), _brutalized(to), amount);
    emit GalacticHeistAlert(from, to, amount);
}
```

Note

Upon talking to the team, it was mentioned that the functionality is intended and will be used for marketing purpose. Additionally, it was pointed out that they won't be providing liquidity to the token and it will be made clear to the users that the only way to get tokens is to mine them.

Users should be made aware that tokens can be transferred from one account to another without their approval. Although there may be multiple reasons behind this, such as marketing strategies or game mechanics, it is important for users interacting with the contract to understand this behavior.

Status

Acknowledged

AntiGravity Team's Comment

It's for marketing purposes. We're hoping there will be a lot of tweeting and uproar about the darkx being stolen from peoples wallets we hope to make a game out of it.

2. Inconsistent maxsupply of tokens

Path

Dark.sol

Description

The documentation (specs_token.md) states Dark Token supply to be 1 billion.
But in reality this is only 1 Million in the contract.

uint256 public constant MAX_SUPPLY = 1_000_000 ether; //@audit

This poses a threat where the minting of tokens will be stopped way before it is expected and can have cascading effects on the protocols tokenomics

Recommendation

Remove the mismatch in code if total supply is supposed to be 1 billion or change documentation if 1 million tokens is considered as max supply.

Status

Resolved

QuillAudits' Team Comment

The documentation (specs_token.md) has been updated to 1 million, however the contract.Dark.md still mentions 1000000000.

Fixed In

https://github.com/chain-labs/antigravity-core/pull/79/commits/2e69555714c3d46412a63ee70c76161c980d97c0

Informational Issues

1. Remove redundant import statements

Path

MiningRig.sol

Description

The contract import DarkX.sol twice, which is useless

Recommendation

Remove one import statement from two

Status

Resolved

Fixed In

https://github.com/chain-labs/antigravity-core/pull/68/commits/05d368bb1ea8dc88a780dc0718751193df523384

2. Mark Constants as immutable

Description

Variables only set in the constructor and never edited afterwards should be marked as immutable, as it would avoid the expensive storage-writing operation in the constructor (around 20 000 gas per variable) and replace the expensive storage-reading operations (around 2100 gas per reading) to a less expensive value reading (3 gas).

- durationOfPhase
- address public miningRig

Recommendation

Mark these variables as immutable

Status

Resolved

Fixed In

https://github.com/chain-labs/antigravity-core/pull/68/commits/40eee6cac7d926a5d9a6000d5c85b1d3888b1834

Functional Tests Cases

Some of the tests performed are mentioned below:

- Merke root validation
- Should remove from allowlist only by Trusted entity
- Should not safeMint more than MAX SUPPLY
- Should perform endMinitng by claims contract
- Should mine correctly after the phase change
- Should set timer as 3weeks

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the AntiGravity contract. We performed our audit according to the procedure described above.

Some issues of High, Medium, Low severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.In the End, AntiGravity Team Has Resolved Most of the Issues.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in AntiGravity smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of AntiGravity smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the AntiGravity to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over $30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

1000+
Audits Completed

$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
July, 2024

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com