



SMART CONTRACT SECURITY AUDIT OF



Ethernote

Summary

Audit Firm: Guardian Audits

Client Firm: Ethernote

Final Report Date - May 9, 2022

Audit Summary

After a line by line manual analysis and automated review, Guardian Audits has concluded that:

- Ethernote's smart contracts have a **LOW RISK SEVERITY**
- Ethernote's smart contracts have an **ACTIVE OWNERSHIP**
- Important owner privileges in the NFT contract – `createEdition`, `createNote`, `updateEdition`, `updateNote`, `ceaseNote`, `setWstEth`, `setPayout`, `withdrawFees`, `approveWstEth`
- Ethernote's smart contract owner has multiple "write" privileges. Centralization risk correlated to the active ownership is **MEDIUM**

Notice that the examined smart contracts are not resistant to internal exploit. For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

 Blockchain network: **Ethereum**

 Verify the authenticity of this report on Guardian's GitHub: <https://github.com/guardianaudits>

 Comprehensive code coverage + fuzzing test suite:

<https://github.com/GuardianAudits/Ethernote>

Table of Contents

Project Information

Project Overview 4

Audit Scope & Methodology 5

Smart Contract Risk Assessment

Inheritance Graph 6

Findings & Resolutions 7

Report Summary

Auditor’s Verdict 22

Addendum

Disclaimer 23

About Guardian Audits 24

Project Overview

Project Summary

Project Name	Ethernote
Language	Solidity
Codebase	https://github.com/abachi-io/ethernote
Commit	818a13bd4228326398a02cb4cd658db05cdcd437

Audit Summary

Delivery Date	May 8, 2022
Audit Methodology	Static Analysis, Manual Review, Full Test Suite, Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
● Critical	0	0	0	0	0	0
● High	1	0	0	0	0	1
● Medium	2	0	0	0	1	1
● Low	10	0	0	2	0	8

Audit Scope & Methodology

Scope

ID	File	SHA-1 Checksum
NTE	ethernote2.sol	1BC95F33381822A945193AF918161E381100B80A
PSL	ethernotePresale.sol	E8C7EDA92F5096AB832C46A999A152D57426FB2E

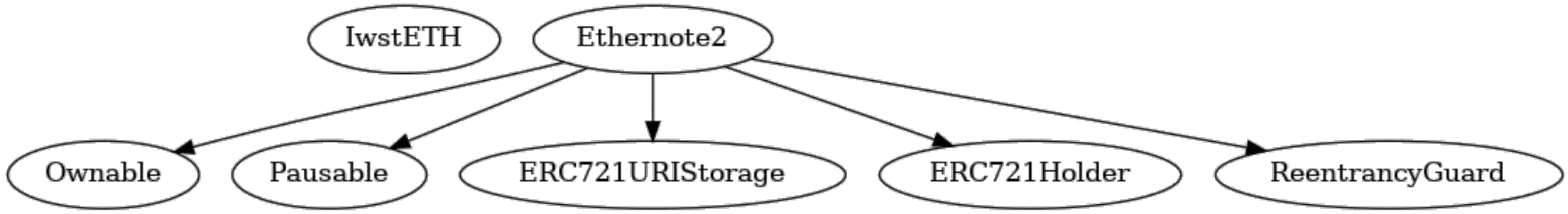
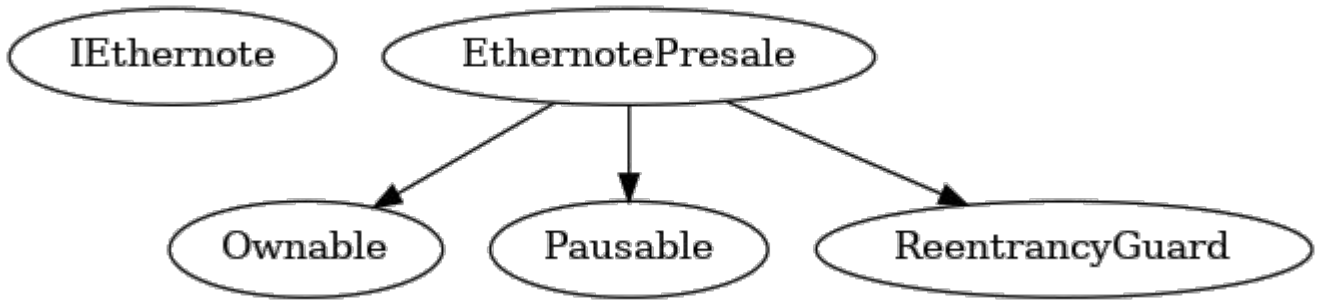
Methodology

The auditing process pays special attention to the following considerations:



- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.
- Comprehensive written tests as a part of a 100% code coverage testing suite.
- **Premium:** Contract fuzzing for increased attack resilience

Vulnerability Level	Classification
● Critical	Easily exploitable by anyone, causing loss/manipulation of assets or data.
● High	Arduously exploitable by a subset of addresses, causing loss/manipulation of assets or data.
● Medium	Inherent risk of future exploits that may or may not impact the smart contract execution.
● Low	Minor deviation from best practices.

Inheritance Graph



Findings & Resolutions

ID	Title	Category	Severity	Status
<u>GLOBAL-1</u>	Centralization Risk	Centralization / Privilege	 Medium	Partially Resolved
<u>NTE-1</u>	Incorrect Fee Structure	Logical error	 High	Resolved
<u>NTE-2</u>	Dangerous Edition Update	Logical Error / Privilege	 Medium	Resolved
<u>NTE-3</u>	Setting Default Values	Optimization	 Low	Resolved
<u>NTE-4</u>	Uncapped Values	Centralization / Trust	 Low	Resolved
<u>NTE-5</u>	Visibility Modifiers	Visibility	 Low	Resolved
<u>NTE-6</u>	Memory Modifiers	Optimization	 Low	Resolved
<u>PSL-1</u>	Pull Over Push Fees	Denial of Service	 Low	Acknowledged
<u>PSL-2</u>	Boolean Redundancy	Optimization	 Low	Resolved
<u>PSL-3</u>	Unchecked Return Value	Control Flow	 Low	Acknowledged
<u>PSL-4</u>	Immutability Modifiers	Mutability	 Low	Resolved
<u>PSL-5</u>	Visibility Modifiers	Visibility	 Low	Resolved
<u>PSL-6</u>	Comparison Inefficiency	Optimization	 Low	Resolved

GLOBAL-1 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● Medium	Global	Partially Resolved

Description

The owner for ethernote2 and ethernotePresale has authority over many functions that may be used to negatively disrupt the project.

Some possible attack vectors include:

- Ceasing the available notes to prevent minting
- Updating an edition’s validator status to prevent minting
- Calling approveWstEth and withdrawing all available wstETH
- Reentering on withdrawFees until the contract’s Ether balance is drained
- Updating a note’s redeemFee to 100% as it is not capped
- Updating a note’s edition index which would prevent minting due to an out-of-bounds error
- Setting the ethernote address in the EthernotePresale to siphon user funds

Recommendation

Ensure that the owner is a multi-sig and/or introduce a timelock for improved community oversight. Optionally introduce require statements to limit the scope of the exploits that can be carried out by the owner.

GLOBAL-1 | Centralization Risk

Resolution

Ethernote Team:

- Add reentrancy guard on withdrawFees function
- Remove denomination and edition being changeable from UpdateNote function
- Add constant MAX_FEE variable - Set currently to 501 (5%) , Add require check in updateNote and createNote - See NTE-3
- Updating index is not possible - the ID is just used to change that ID's attributes not the actual ID
- owner will also be a multi-sig after deployment with function transferOwnership function via OpenZeppelin's Ownable.sol contract
- Although the contract has an ACTIVE OWNERSHIP, once a note is minted, there is no ability for us to pause redemption, only new minting

NTE-1 | Incorrect Fee Structure

Category	Severity	Location	Status
Logical Error	● High	ethernote2.sol	Resolved

Description

The `fees` variable is incremented whenever `_redeemEth` or `_redeemWstEth` is called. However, the `withdrawFees` function treats `fees` as if it were entirely in Ether. Therefore, when `fees` is incremented in the `_redeemWstEth` function it is wrongly attributed as Ether, rather than `wstEth`.

Because of the misattribution, once `withdrawFees` is called the contract will no longer have enough Ether to back all of the regular Ether ethernotes, rendering some holders of regular Ether ethernotes unable to redeem and causing complete loss of funds for those users.

Notably, a malicious actor could front-run a call to `withdrawFees` to mint and redeem `wstEth` ethernotes, causing this imbalance and complete loss of funds for some regular Ether ethernote holders.

Recommendation

Track Ether fees and `wstEth` fees separately or find some fee structure that maintains the backing value of all ethernotes.

Resolution

Ethernote Team:

- Add a new variable `feesWstEth`
- Add a new function `withdrawWstEthFees`

NTE-2 | Dangerous Edition Update

Category	Severity	Location	Status
Logical Error / Privilege	● Medium	ethernote2.sol	Resolved

Description

The owner can update the `ethereum` field for an edition that has already been minted. This can lead to a state where users are shifted to a different redemption method but there are not enough funds for all users to redeem their ethernote, causing complete loss of funds for some ethernote holders.

Notably, a transaction to update the `ethereum` field can be sandwich attacked by a malicious actor to mint notes of the edition with one token before the `ethereum` value is updated and redeem their notes after for a different token. A malicious sandwich attack would leave some ethernetes unbacked by their respective token.

Recommendation

Only update the `ethereum` field when minting is paused and properly shift funds between Ether and `wstEth` to ensure all users can still redeem.

Resolution

Ethernote Team:

- Remove `_ethereum` Boolean in `updateEdition` to prevent changing type of note.

NTE-3 | Setting Default Values

Category	Severity	Location	Status
Optimization	● Low	ethernote2.sol: 60, 61	Resolved

Description

The variables circulatingNotes and fees are assigned to 0. This is unnecessary because the default value for the uint256 type is 0.

Recommendation

Remove the assignment.

Resolution

Ethernote Team:

- Remove the 0 assignment as uint256 defaults to 0

NTE-4 | Uncapped Values

Category	Severity	Location	Status
Centralization / Trust	● Low	ethernote2.sol: 93, 128	Resolved

Description

In the functions `createNote` and `updateNote`, the parameter `_redeemFee` is not capped. `_redeemFee` can be be arbitrarily set to siphon user funds and even exceed 100% to entirely prevent redemption.

Recommendation

A `require` statement can be added to cap the note's `redeemFee` at a reasonable rate.

Resolution

Ethernote Team:

- `createNote` added `require(_redeemFee < MAX_FEE, "[x] Max fee exceeded");`
- `updateNote` added `require(_redeemFee < MAX_FEE, "[x] Max fee exceeded");`
- `ceaseNote` has no effect on `_redeemFee`

NTE-5 | Visibility Modifiers

Category	Severity	Location	Status
Visibility	● Low	ethernote2.sol	Resolved

Description

The functions getTokenEditionName, getTokenEditionId, getTokenNotId, getTokenBalance, getEditionsLength, getNotesLength, totalPrinted, and notesCirculating are marked as public, but are never called from inside the contract.

Recommendation

Modify their visibility from public to external.

Resolution

Ethernote Team:

Modified visibility from public to external for the following functions:

- getTokenEditionName
- getTokenEditionId
- getTokenNotId
- getTokenBalance
- getEditionsLength
- getNotesLength
- totalPrinted
- notesCirculating

NTE-6 | Memory Modifiers

Category	Severity	Location	Status
Optimization	● Low	ethernote2.sol: 74, 114, 254	Resolved

Description

The functions `onERC721Received`, `updateEdition`, and `createEdition` have arguments marked as `memory` when they could be declared `calldata`.

Changing the memory modifiers from `memory` to `calldata` will demand more gas upon deployment but less gas for each call to `onERC721Received`, `updateEdition`, and `createEdition`.

Recommendation

If it is desired to reduce gas for these functions at the expense of deployment costs, change the memory modifiers from `memory` to `calldata`.

Resolution

Ethernote Team:

- Changed from `memory` to `calldata`

PSL-1 | Pull Over Push Fees

Category	Severity	Location	Status
Denial of Service	● Low	ethernotePresale.sol: 80, 91	Acknowledged

Description

In the `mintSilver` and `mintGold` functions a minting fee is charged and immediately sent to the `payout` address. In the event that the `payout` address was set to a malicious contract it could stop minting by reverting upon call, resulting in a denial of service for minting.

Recommendation

Adopt a pull-over-push pattern for charging fees and rely on the `withdrawAll` function to collect fees.

Resolution

Ethernote Team:

- No Change

PSL-2 | Boolean Redundancy

Category	Severity	Location	Status
Optimization	● Low	ethernotePresale.sol: 104, 111	Resolved

Description

Modifiers `hasWhitelistGold` and `hasWhitelistSilver` use `whitelistedGold[msg.sender] == true` and `whitelistedSilver[msg.sender] == true` respectively. This is redundant since the mapping values are booleans themselves.

Recommendation

Remove the unnecessary boolean assertion.

Resolution

Ethernote Team:

- Remove unnecessary Boolean assertion

PSL-3 | Unchecked Return Value

Category	Severity	Location	Status
Control Flow	● Low	ethernotePresale.sol:162	Acknowledged

Description

The `withdraw` function uses `transfer` which provides a return value that should be checked. Not all ERC20 implementations revert in case of failure, so it is important to have some logic in the event these executions fail.

Recommendation

Check the return value, or opt for a `safeTransfer` alternative.

Resolution

Ethernote Team:

- Not expecting any ERC20 tokens in this contract, only recovering ERC20 tokens to this address

PSL-4 | Immutability Modifiers

Category	Severity	Location	Status
Mutability	● Low	ethernotePresale.sol	Resolved

Description

The ETH_01, ETH_05, ETH_10, ETH_100, and ETH_FEE variables are never mutated, and should therefore be declared constant.

Recommendation

Declare them as constant.

Resolution

Ethernote Team:

- Changed to recommended
- No function exists to change these variables

PSL-5 | Visibility Modifiers

Category	Severity	Location	Status
Visibility	● Low	ethernotePresale.sol: 165	Resolved

Description

The withdrawAll function is marked as public, but is never called from inside the contract.

Recommendation

Modify the visibility of withdrawAll from public to external.

Resolution

Ethernote Team:

- Changed withdrawAll function to external

PSL-6 | Comparison Inefficiency

Category	Severity	Location	Status
Optimization	● Low	ethernotePresale.sol: 136, 151	Resolved

Description

The `require` statements in `addMultipleWhitelistSilver` and `addMultipleWhitelistGold` use a `<=` operator, while a `<` operator combined with incrementing the right hand side of the comparison is a more gas efficient alternative.

Recommendation

Utilize a `<` operator and increment the right hand side of the comparison.

Resolution

Ethernote Team:

- Change `<=` to `<`

Auditor's Verdict

After a line by line manual analysis and automated review, Guardian Audits has concluded that:

- Ethernote's smart contracts have a **LOW RISK SEVERITY**
- Ethernote's smart contracts have an **ACTIVE OWNERSHIP**
- Important owner privileges in the NFT contract – `createEdition`, `createNote`, `updateEdition`, `updateNote`, `ceaseNote`, `setWstEth`, `setPayout`, `withdrawFees`, `approveWstEth`
- Ethernote's smart contract owner has multiple "write" privileges. Centralization risk correlated to the active ownership is **MEDIUM**

Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Guardian’s position is that each company and individual are responsible for their own due diligence and continuous security. Guardian’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Guardian Audits

Founded in 2022 by DeFi experts, Guardian Audits is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian Audits upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit <https://guardianaudits.com>

To view our audit portfolio, visit <https://github.com/guardianaudits>

To book an audit, message <https://t.me/guardianaudits>