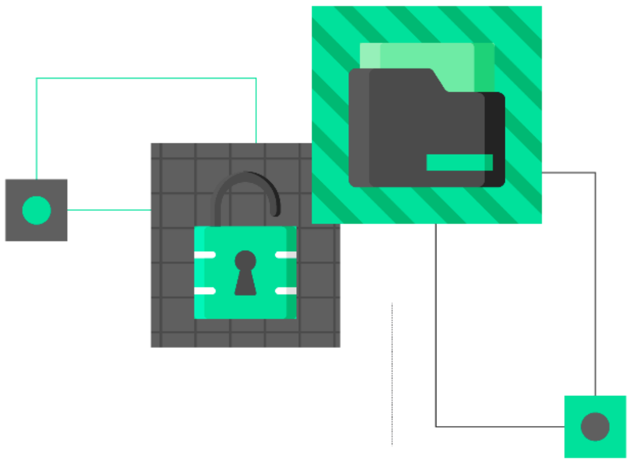
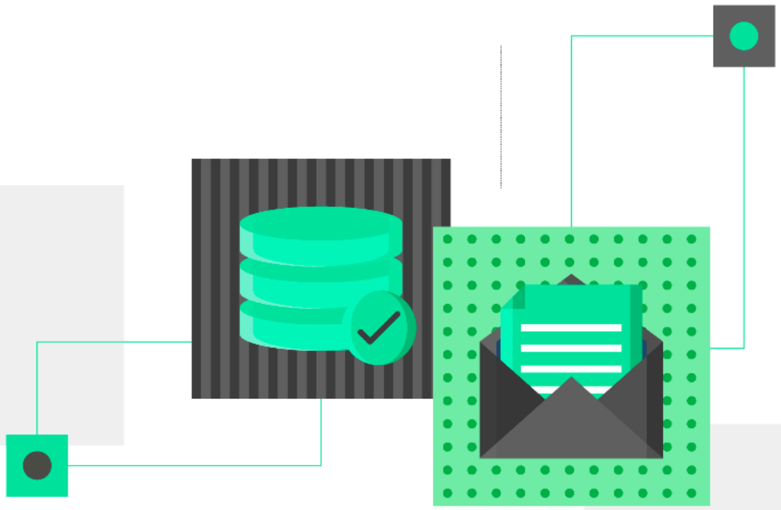
Mantisec Labs

Smart Contract Audit

Fusor-Contracts





Dec 2024

Mantisec Labs

Contents

Disclaimer

Audit Process & Methodology

The Mantisec Labs team carried out a thorough audit for the project, starting with an in-depth analysis of code design patterns. This initial step ensured the smart contract's architecture was well-structured and securely integrated with third-party smart contracts and libraries.Also, our team conducted a thorough line-by-line inspection of the smart contract,seeking out potential issues such as Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, among others.
During the Unit testing phase, we assessed the functions authored by the developer to ascertain their precise functionality. Our Automated Testing procedures leveraged proprietary tools designed in-house to spot vulnerabilities and security flaws within the Smart Contract. The code was subjected to an in-depth audit administered by an independent team of auditors, encompassing the following critical aspects:

- Scrutiny of the smart contract's structural analysis to verify its integrity.
- Extensive automated testing of the contract
- A manual line-by-line Code review, undertaken with the aim of evaluating, analyzing, and identifying potential security risks.
- An evaluation of the contract's intended behavior, encompassing a review of provided documentation to ensure the contract conformed to expectations.
- Rigorous verification of storage layout in upgradeable contracts.
- An integral component of the audit procedure involved the identification and recommendation of enhanced gas optimization techniques for the contract

Audit Purpose

Mantisec Labs was hired by the Ethernity team to review their smart contract.
This audit was conducted in **Dec 2024**.

The main reasons for this review were:

- To find any possible security issues in the smart contract.
- To carefully check the logic behind the given smart contract.

This report provides valuable information for assessing the level of risk associated with this smart contract and offers suggestions on how to improve its security by addressing any identified issues.

Contract Details

Project Name	Ethernity
Contract links	https://github.com/ethernitychain/epictoken/blob/governor/contracts/EpicToken.sol
Language	Solidity
Type	ERC20

Security Level Reference

Each problem identified in this report has been categorized into one of the following severity levels:

- **Critical**: Vulnerabilities that present an immediate and serious threat to system or data integrity, demanding urgent action.
- **High**: Significant risks that have the potential to cause major security breaches or loss of functionality.
- **Medium**: Issues that moderately affect system performance or security and require timely resolution.
- **Low**: Low-risk concerns primarily related to optimization and code quality, with minimal direct impact on system security.
- **Informational**: Observations or recommendations that do not pose any direct risk but provide insights for potential improvements or best practices.

Severity	Score
Critical	4-5
High	3-4
Medium	2-3
Low	1-2
Informational	0-1

Mantisec Labs

Findings Overview

Contract Name:

- **SaleFixedPrice.sol**
- **Claim.sol**

Critical	**High**	**Medium**	**Low**	**Informational**
0	4	1	1	0

Identified Issues and Resolutions

Issue	**Severity**	**Fix Date**
L01- Unused Events in Contract	**Low** (1.2)	
M01- Visibility Mismatch for Internal Function: claimableTokens Should Be internal	**Medium** (2.5)	
H01- Incorrect Execution Order: Token Transfer Occurs Before _listingDuration Validation in ListToken Function	**High** (4.0)	
H02- Incorrect Handling of `max` Value in Listing	**High** (3.2)	
H03- Lack of Offer Deletion in cancelOffer Function	**High** (3.6)	
H04- Missing _cliffPeriod Validation in registerPurchase Function	**High** (3.5)	

Detailed Findings

Contract: SaleFixedPrice

L01- Unused Events in Contract

Description:

OfferCancelled and OfferRenewed events are declared but never emitted in the contract.

Suggested Improvement:

Either implement functionality to emit these events where relevant or remove their declarations if not needed.

H01- Incorrect Execution Order: Token Transfer Occurs Before _listingDuration Validation in ListToken Function

Description:

In the ListToken function of the Fusor OTC contract, tokens are transferred to the contract with _token.safeTransferFrom before validating the _listingDuration parameter. This sequence can lead to unnecessary token transfers if _listingDuration is invalid, as the transfer occurs regardless of any subsequent validation failure. Specifically, if _listingDuration is less than or equal to zero, the transfer operation will complete but the function will revert due to the failed duration check, resulting in a wasted gas cost for the user and potentially locking tokens in the contract.

Suggested Improvement:

Reorder operations: Place _listingDuration validation (require(_listingDuration > 0, "Invalid listing duration");) before the token transfer (safeTransferFrom).

H02- Incorrect Handling of `max` Value in Listing

Description:
When _listingData.balance > _listingData.min, the max value is being set to _listingData.balance. This behavior may unintentionally override the intended upper limit (max) for purchases, leading to potential inconsistencies in listing constraints.

Impact:

- Buyers may be allowed to purchase more tokens than initially intended.
- Disrupts the balance between min and max constraints for listings.

Recommendation:
Introduce a conditional check to ensure max is updated only if it aligns with the listing's intended logic and constraints. Confirm the requirement to dynamically adjust max based on balance and implement safeguards as needed.

H03- Lack of Offer Deletion in cancelOffer Function

Description:
The current implementation of the cancelOffer function marks the offer as inactive but does not remove it from the offers array. As a result, canceled offers remain in the array, leading to unnecessary data retention and potential inefficiency.

Impact:

- Increased storage costs and gas fees due to the persistence of canceled offers in the array.
- Potential confusion, as canceled offers still exist in the list despite being inactive.

Recommendation:
Implement the delete operation on the offer in the offers array after marking it as inactive. This will properly remove canceled offers from storage, improving gas efficiency and data integrity.

Mantisec Labs

Contract: Claim

H04- Missing _cliffPeriod Validation in registerPurchase Function

Description:

The registerPurchase function lacks validation for the _cliffPeriod parameter, which could allow unintended or nonsensical values to be set for cliff duration.

Suggested Improvement:

Add a requirement check on _cliffPeriod to ensure it is within acceptable limits (e.g., require(_cliffPeriod >= MIN_CLIFF_PERIOD, "Invalid cliff period");).

M01- Visibility Mismatch for Internal Function: claimableTokens Should Be internal

Description:

The claimableTokens function is public but is used only within the contract (internally).

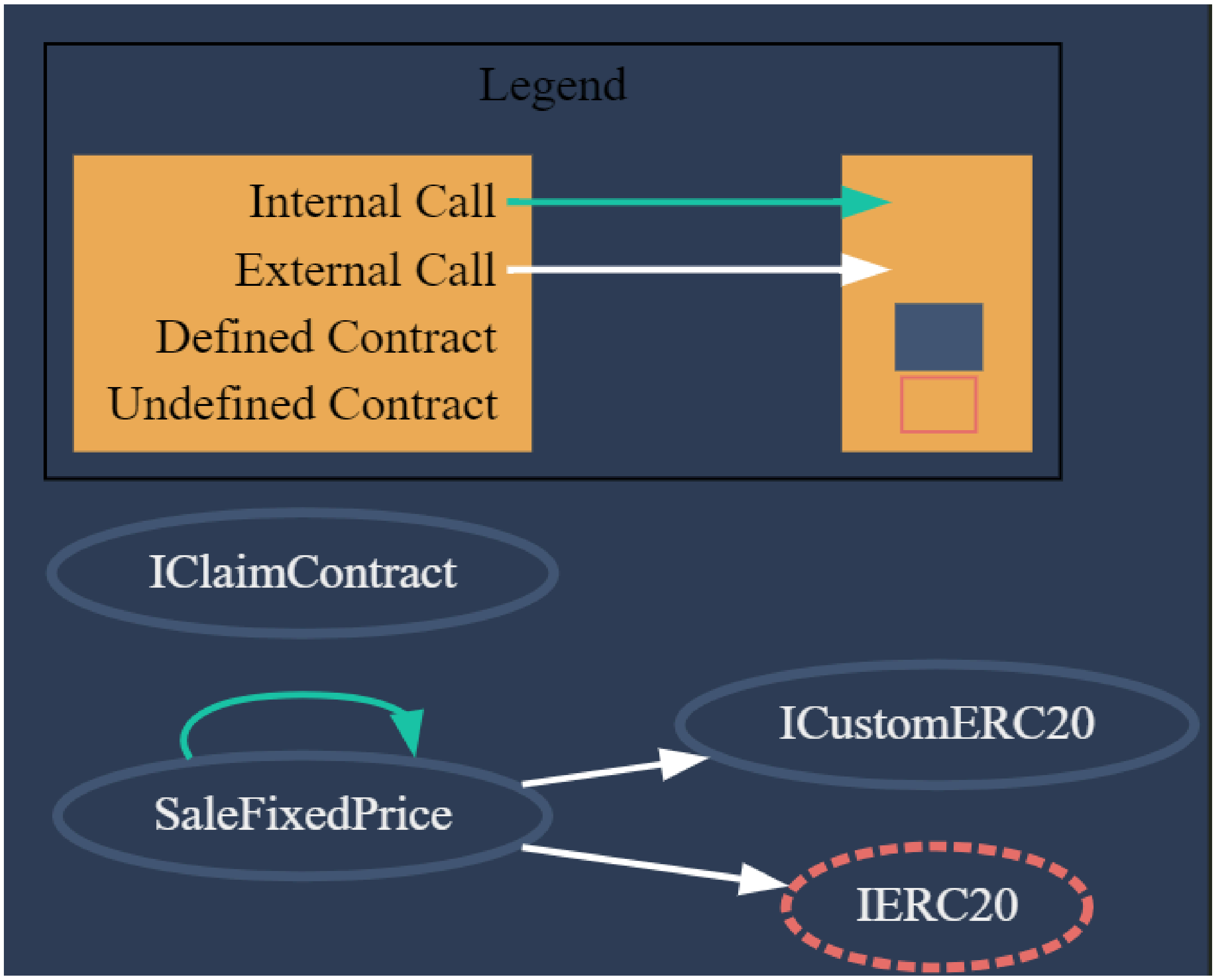
Suggested Improvement:

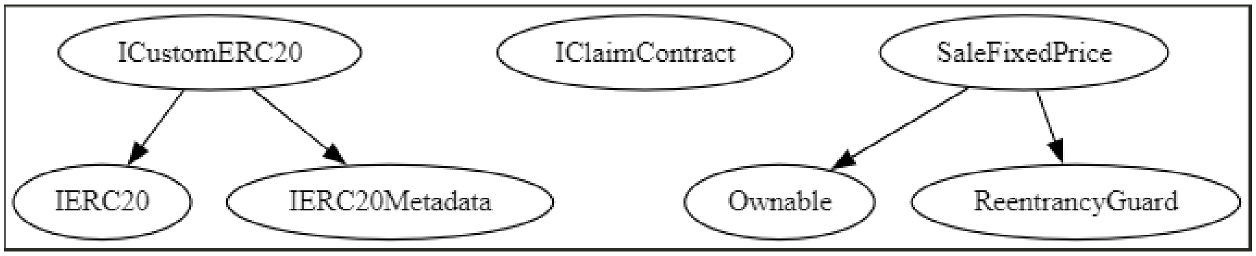
Change to internal: Update the function visibility to internal to limit access and reduce unnecessary exposure, enhancing security and efficiency.

Mantisec Labs

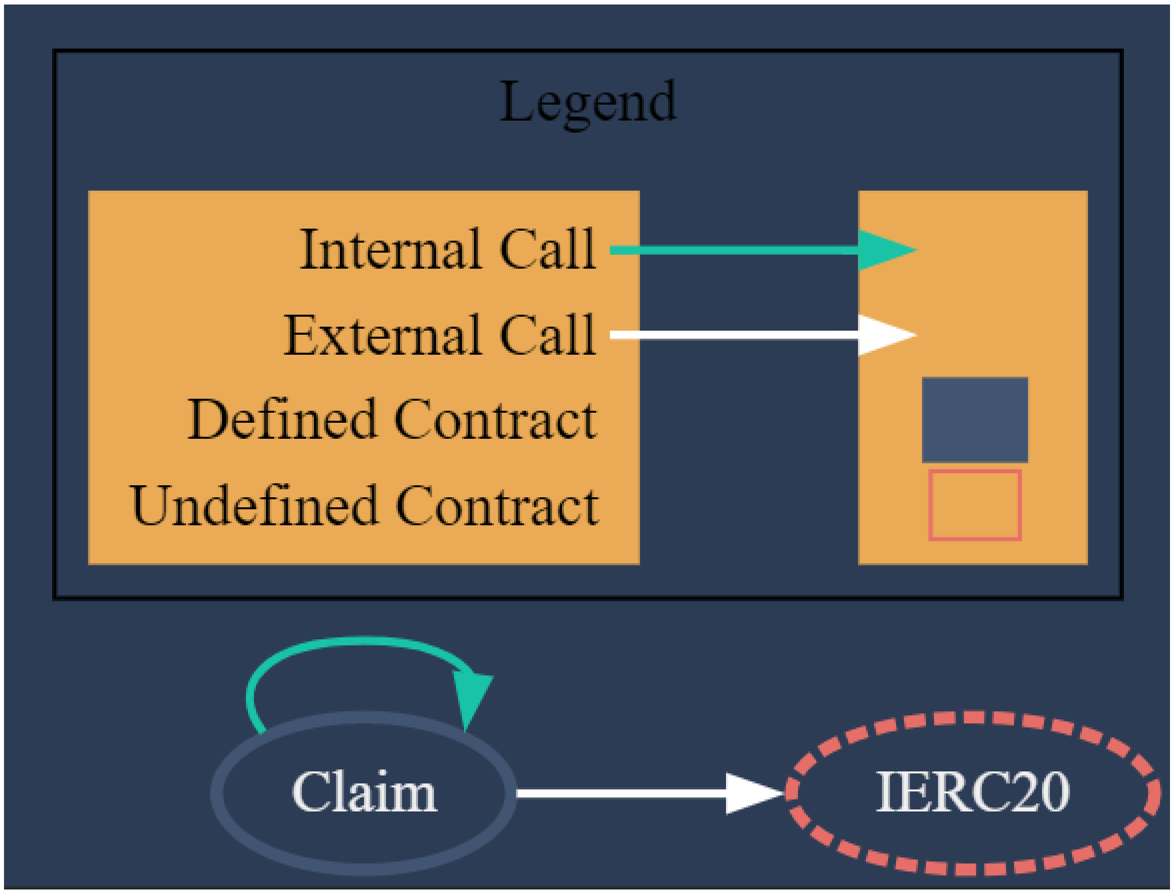
Additional Details:

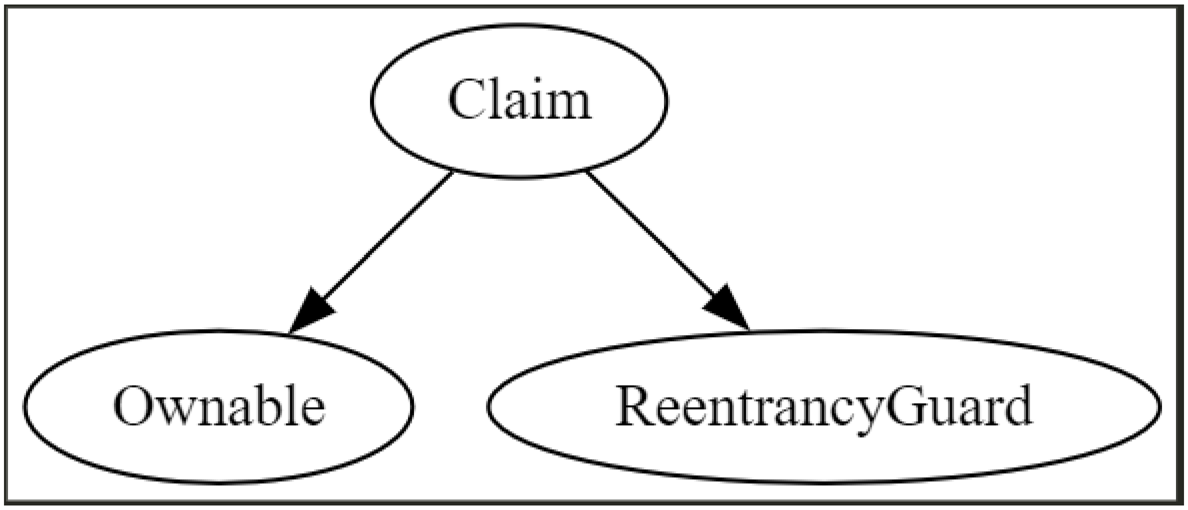
Contract: SaleFixedPrice





Contract: **Claim**





Concluding Remarks

To wrap it up, this audit has given us a good look at the contract's security and functionality.
Our auditors confirmed that all the issues are now resolved by the developers.