

SMART CONTRACT

Security Audit Report

Project: Enfineo
Website: enfineo.com
Platform: Binance Smart Contract
Language: Solidity
Date: September 9th, 2024

Table of contents

Introduction	4
Project Background	4
Audit Scope	5
Claimed Smart Contract Features	6
Audit Summary	7
Technical Quick Stats	8
Business Risk Analysis	9
Code Quality	10
Documentation	10
Use of Dependencies	10
AS-IS overview	11
Severity Definitions	14
Audit Findings	15
Conclusion	21
Our Methodology	22
Disclaimers	24
Appendix	
• Code Flow Diagram	25
• Slither Results Log	28
• Solidity static analysis	32
• Solhint Linter	35

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Introduction

EtherAuthority was contracted by the Enfineo team to perform the Security audit of the Enfineo smart contracts code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on September 9th, 2024.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

- The Enfineo Contracts handle multiple contracts, and all contracts have different functions.
 - **enfineoToken:** This contract has clear role-based access, making it more secure and manageable, with controlled minting and burning operations.
 - **enfineoStaking:** This contract allows ENF token holders to stake their tokens and earn rewards over time based on predefined deposit types. The contract supports early withdrawals with penalties, and the reward pool is managed by trusted roles. Security features such as pausing, role-based access, and reentrancy protection are integrated for robust operation.
 - **enfineoVesting:** This Ethereum-based smart contract for a vesting system using Solidity. The contract implements functionality related to creating, managing, and claiming tokens within different vesting schedules. The code involves handling vesting schedules for various groups such as "SEED," "EARLY_ADOPTERS," "PRIVATE," and others, each with defined periods and percentages for token distribution. It effectively combines vesting and staking features while ensuring token distribution happens gradually based on predefined schedules.
- This audit scope has included 3 smart contract files, 2 interface files, and 2 struct files.
- The Enfineo contracts inherit the AccessControl, ReentrancyGuard, Pausable, Strings, IERC20, ERC20 standard smart contracts from the OpenZeppelin library.

- These OpenZeppelin contracts are considered community-audited and time-tested and hence are not part of the audit scope.

Audit scope

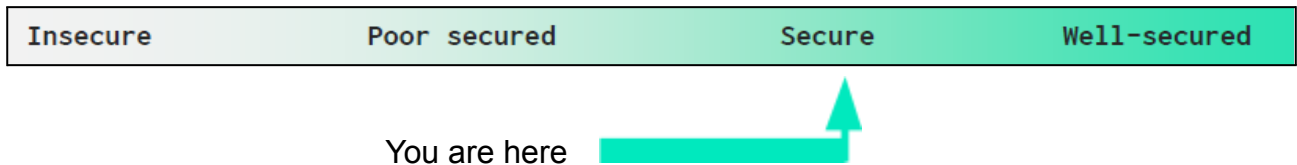
Name	Code Review and Security Analysis Report for Enfineo Smart Contracts
Platform	Binance Smart Contract
Language	Solidity
File 1	enfineoToken.sol
File 1 MD5 hash	1DB5A4E3642A2BE632E912F5F4D31494
File 2	enfineoStaking.sol
File 2 MD5 hash	102D957218F9E1D6034CCAC05F606427
Updated File 2 MD5 hash	7AAFD2E9D5B309B70DB003E1840C8DB5
File 3	enfineoVesting.sol
File 3 MD5 hash	7EC9B95ACDC49E14623E98AF7FBE5D06
Updated File 3 MD5 hash	AF5F7E8FBFB9D8ED56DFF42AB158838F
File 4	IENF.sol
File 4 MD5 hash	9F69E9592A5C4555405C5B11B2C2DE74
Updated File 4 MD5 hash	CD7D0DDF39415E2E5486996C846F66D6
File 5	IENFVesting.sol
File 5 MD5 hash	EFA1A34906C7C8D05E2F639E874613E9
File 6	StakingStructs.sol
File 6 MD5 hash	3E45206D869F3A3548F9D695F214D4AD
File 7	VestingStructs.sol
File 7 MD5 hash	5A1F5F732C2A9BAEF820823F87513CD5
Audit Date	September 9th, 2024
Revised Audit Date	September 11th, 2024

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
<p>File 1: enfineoToken.sol</p> <p>Tokenomics:</p> <ul style="list-style-type: none">• Name: enfineo• Symbol: ENF• Decimals: 18• Maximum Supply: 110 Million <p><u>Other Control:</u></p> <ul style="list-style-type: none">• Mint new tokens by the minter.	<p>YES, This is valid.</p>
<p>File 2: enfineoStaking.sol</p> <p><u>Other Control:</u></p> <ul style="list-style-type: none">• The Operational role can add/delete the reward pool.• The Operational role can pause/unpause stake.• The setters can set Enf Token.• The setters can set the vesting contract address.• Update/Add a deposit type by the Deposit type role.	<p>YES, This is valid.</p>
<p>File 3: enfineoVesting.sol</p> <p><u>Admin Control:</u></p> <ul style="list-style-type: none">• Set the start time of the claim and stake.• Set the start time of the vesting contract.• Set the token address.• Set the stake contract address.	<p>YES, This is valid.</p>

Audit Summary

According to the standard audit assessment, the Customer's solidity-based smart contracts are "**Secured**". Also, these contracts contain owner control, which does not make them fully decentralized.



We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.

All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit Overview section. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 1 high, 0 medium, 3 low, and 6 very low-level issues.

We confirm that all issues are fixed/Acknowledged in the revised smart contract code.

Investor Advice: A technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner-controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	The solidity version is not specified	Passed
	The solidity version is too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	"Out of Gas" Issue	Passed
	High consumption 'for/while' loop	Passed
	High consumption 'storage' storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage is not set	Passed
	"Short Address" Attack	Passed
	"Double Spend" Attack	Passed

Overall Audit Result: **PASSED**

Business Risk Analysis

Category	Result
● Buy Tax	0%
● Sell Tax	0%
● Cannot Buy	No
● Cannot Sell	No
● Max Tax	0%
● Modify Tax	Not Detected
● Fee Check	No
● Is Honeypot	Not Detected
● Trading Cooldown	Not Detected
● Can Pause Trade?	No
● Pause Transfer?	Not Detected
● Max Tax?	No
● Is it Anti-whale?	Not Detected
● Is Anti-bot?	Not Detected
● Is it a Blacklist?	Not Detected
● Blacklist Check	No
● Can Mint?	Yes
● Is it a Proxy?	No
● Can Take Ownership?	No
● Hidden Owner?	Not Detected
● Self Destruction?	Not Detected
● Auditor Confidence	High

Overall Audit Result: PASSED

Code Quality

This audit scope has included 3 smart contract files, 2 interface files, and 2 struct files. Smart contracts contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in Enfineo are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties/methods can be reused many times by other contracts in the Enfineo.

The Enfineo team has **not** provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are **well** commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given an Enfineo smart contract code in the form of a file. The hash of that code is mentioned in the table above.

As mentioned above, code parts are **well**-commented. and the logic is straightforward. So it is easy to quickly understand the programming flow as well as the complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries used in this smart contracts infrastructure are based on well-known industry-standard open-source projects.

Apart from libraries, its functions are used in external smart contract calls.

AS-IS overview

enfineoToken.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	mint	external	Role of Mint	No Issue
3	burn	external	Role of BURNER	Acknowledged
4	name	read	Passed	No Issue
5	symbol	read	Passed	No Issue
6	decimals	read	Passed	No Issue
7	totalSupply	read	Passed	No Issue
8	balanceOf	read	Passed	No Issue
9	transfer	write	Passed	No Issue
10	allowance	read	Passed	No Issue
13	approve	write	Passed	No Issue
14	transferFrom	write	Passed	No Issue
15	transfer	internal	Passed	No Issue
16	update	internal	Passed	No Issue
17	mint	internal	Passed	No Issue
18	burn	internal	Passed	No Issue
19	approve	internal	Passed	No Issue
20	approve	internal	Passed	No Issue
21	spendAllowance	internal	Passed	No Issue
22	onlyRole	modifier	Passed	No Issue
23	supportsInterface	read	Passed	No Issue
24	hasRole	read	Passed	No Issue
25	checkRole	internal	Passed	No Issue
26	checkRole	internal	Passed	No Issue
27	getRoleAdmin	read	Passed	No Issue
28	grantRole	write	Role of Admin	No Issue
29	revokeRole	write	Passed	No Issue
30	renounceRole	write	Role of Admin	No Issue
31	setRoleAdmin	internal	Passed	No Issue
32	grantRole	internal	Passed	No Issue
33	revokeRole	internal	Passed	No Issue

enfineoStaking.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	stake	external	Passed	No Issue
3	unstake	write	Passed	No Issue
4	updateRewardPool	write	Passed	No Issue

5	addRewardToPool	external	Role of Operational	No Issue
6	removeRewardFromPool	external	Role of Operational	No Issue
7	pauseStake	external	Role of Operational	No Issue
8	unpauseStake	external	Role of Operational	No Issue
9	setEnfToken	external	Role of Setter	No Issue
10	setVestingContractAddress	external	Role of Setter	No Issue
13	updateDepositsType	external	Passed	Fixed
14	getEnfToken	external	Passed	No Issue
15	getVestingContractAddress	external	Passed	No Issue
16	getDepositTypes	external	Passed	No Issue
17	getDepositsNumberPerOwner	external	Passed	No Issue
18	getDepositsByOwner	external	Passed	No Issue
19	getRewardPoolAmount	external	Passed	No Issue
20	getRewardOnADepositType	read	Passed	No Issue
21	nonReentrant	modifier	Passed	No Issue
22	_nonReentrantBefore	write	Passed	No Issue
23	_nonReentrantAfter	write	Passed	No Issue
24	_reentrancyGuardEntered	internal	Passed	No Issue
25	onlyRole	modifier	Passed	No Issue
26	supportsInterface	read	Passed	No Issue
27	hasRole	read	Passed	No Issue
28	checkRole	internal	Passed	No Issue
29	checkRole	internal	Passed	No Issue
30	getRoleAdmin	read	Passed	No Issue
31	grantRole	write	Role of Admin	No Issue
32	revokeRole	write	Passed	No Issue
33	renounceRole	write	Role of Admin	No Issue
34	_setRoleAdmin	internal	Passed	No Issue
35	grantRole	internal	Passed	No Issue
36	revokeRole	internal	Passed	No Issue
37	when not paused	modifier	Passed	No Issue
38	whenPaused	modifier	Passed	No Issue
39	paused	read	Passed	No Issue
40	_requireNotPaused	internal	Passed	No Issue
41	_requirePaused	internal	Passed	No Issue
42	_pause	internal	Passed	No Issue
43	_unpause	internal	Passed	No Issue

enfineoVesting.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	Fixed
2	createVestingSchedule	write	Passed	No Issue
3	createVestingSchedules	external	Passed	Acknowledged
4	claimTokens	external	Passed	No Issue

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

5	claimAndStakeTokens	external	Passed	No Issue
6	calculateTgeAmount	read	Passed	No Issue
7	setVestingContractClaimAndStakeStartingDate	external	Role of Admin	No Issue
8	getVestingContractClaimAndStakeStartingDate	external	Passed	No Issue
9	setVestingContractStartingDate	external	Role of Admin	No Issue
10	getVestingContractStartingDate	external	Passed	No Issue
13	computeVestingIdForAddressAndVestingName	write	Passed	No Issue
14	getAddressVestingSchedulesIds	read	Passed	No Issue
15	getVestingById	external	Passed	No Issue
16	getAddressVestingsCount	external	Passed	No Issue
17	getEnfTokenAddress	external	Passed	No Issue
18	getStakingContractAddress	external	Passed	No Issue
19	setEnfToken	external	Role of Admin	No Issue
20	setStakeContractAddress	external	Role of Admin	No Issue
21	getVestingPeriodsDetailsByVestingId	external	Passed	No Issue
22	getEndOfVestingByVestingId	external	Passed	No Issue
23	setVestingScheduleEnableStatus	external	Passed	Acknowledged
24	onlyRole	modifier	Passed	No Issue
25	supportsInterface	read	Passed	No Issue
26	hasRole	read	Passed	No Issue
27	checkRole	internal	Passed	No Issue
28	checkRole	internal	Passed	No Issue
29	getRoleAdmin	read	Passed	No Issue
30	grantRole	write	Role of Admin	No Issue
31	revokeRole	write	Passed	No Issue
32	renounceRole	write	Role of Admin	No Issue
33	setRoleAdmin	internal	Passed	No Issue
34	grantRole	internal	Passed	No Issue
35	revokeRole	internal	Passed	No Issue
36	nonReentrant	modifier	Passed	No Issue
37	nonReentrantBefore	write	Passed	No Issue
38	nonReentrantAfter	write	Passed	No Issue
39	reentrancyGuardEntered	internal	Passed	No Issue
40	getVestingPeriodCurrentAllocatedAmount	external	Passed	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets, that can't have a significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

(1) BURNER can burn anyone's token: [enfineoToken.sol](#)

BURNER can burn any user's tokens.

Resolution: We suggest changing the code so only token holders can burn their own tokens and not anyone else. Not even a contract creator.

Status: **Fixed.**

Medium

No Medium-severity vulnerabilities were found.

Low

(1) Infinite loops possibility: [enfineoVesting.sol](#)

As the number of array elements increases, it will cost more and more gas. And eventually, it will stop all the functionality. After several hundreds of transactions, all those functions depending on it will stop. We suggest avoiding loops. For example, use mapping to store the array index. And query that data directly, instead of looping through all the elements to find an element.

Resolution: Adjust logic to replace loops with mapping or other code structures.

- setVestingScheduleEnableStatus() - vestingScheduleIds.length
- createVestingSchedules() - beneficiaries.length

Status: **Acknowledged.** The client has confirmed that the length will not be greater than 100.

(2) Staking Contract Burner Role Setup: [enfineoToken.sol](#)

To enable the staking contract to burn tokens, it needs to be assigned the BURNER_ROLE in the token contract. Without this role, unstake will revert.

Resolution: The token contract owner should grant the BURNER_ROLE to the staking contract. This can be done using the token contract's role management functions to ensure not reverting the unstaking process.

Status: **Acknowledged.**

(3) Undeclared identifier: [enfineoToken.sol](#)

```
30
31     function burn(uint256 amount) external onlyRole(BURNER_ROLE) {
32         _burn(msg.sender, amount);
33         emit Burn(_account, amount);
34     }
```


The `_account` variable and the `_amount` variables are used to log a burn event but not defined in the function.

Resolution: We suggest changing the variable names and using the correct variables in the event log.

Status: **Fixed.**

Very Low / Informational / Best practices:

(1) Variable Initialization: [enfineoVesting.sol](#)

```
function createVestingSchedules(  infinite gas
    address[] calldata beneficiaries,
    string[] calldata vestingNames,
    uint256[] memory amounts
    external onlyRole(CREATE_VESTING_SCHEDULE_ROLE) {
    uint256 i;
    if (beneficiaries.length != vestingNames.length || beneficiaries.length != amounts.length) {
        revert WrongParam();
    }
}
```

Always initialize variables to avoid unexpected behavior or logical errors. Uninitialized variables may lead to unpredictable results and security issues.

Resolution: Initialize variables with a default value, e.g., `uint256 i = 0;`. This practice ensures variables start with a known value, improving code reliability and security.

Status: **Fixed.**

(2) Unused error and event:

[enfineoVesting.sol](#)

Errors are defined but not used in code.

- InvalidReleasedAmount
- InvalidDepositType

Events are defined but not used in code.

- StakeContractAccess

[enfineoStaking.sol](#)

Errors are defined but not used in code.

- DepositsNotMature
- Unauthorized
- DepositAlreadyUstaked
- CoolingDeposit
- NotEnoughTokens
- InvalidAmountToUpdateRewardPool

Events are defined but not used in code.


- Claim
- SetPenalty

Resolution: We suggest removing unused errors and events.

Status: **Fixed.**

(3) Spelling mistake:

[enfineoVesting.sol](#)

```
constructor() {  infinite gas 3693600 gas
  _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
  _grantRole(CREATE_VESTING_SCHEDULE_ROLE, msg.sender);
  _grantRole(ENABLE_VESTING_SCHEDULE_ROLE, msg.sender);

  // @dev we define how much time an interval should last (in days)
  _vestingDefinitions["SEED"].vestingPeriods = [0 days, 500, 1 seconds, 500, 1 seconds, 500, 1 seconds, 500, 1 seconds, 12*500];

  /** @dev we define how much percentage an interval has alocated. value is equal to desired percentage * SCALING_FACTOR.
   * E.g.: If SCALING_FACTOR = 100 and desired percentage is 5.65, the value inserted in the array should be 565
   */
  _vestingDefinitions["SEED"].vestingPeriodsPercentages = [500, 0, 125, 0, 125, 0, 125, 0, 125, 5000];

  // @dev we define total amount of token alocated to this vesting
  _vestingTokens["SEED"] = VestingTokensStruct(8325000 * 10 ** 18, 0);
```

Spelling mistake in constructor comments.

“alocated” should be **“allocated”**.

```

/**
 * @notice Create a vesting schedules for an array of addresses with predefined types of vesting plans.
 * @notice this function is accessible only to addresses with special role: CREATE_VESTING_SCHEDULE_ROLE
 * @param beneficiaries array of addresses of the vesting schedule that can claim the tokens
 * @param vestingNames array of vesting names
 * @param amounts array of amounts (amount in Gwei-already multiplied by 10**18) that are locked accordi
 */
function createVestingSchedules( infinite gas
    address[] calldata beneficiaries,
    string[] calldata vestingNames,
    uint256[] memory amounts
) external onlyRole(CREATE_VESTING_SCHEDULE_ROLE) {

```

“accessible” should be “**accessible**”.

enfineStaking.sol

```

/**
 * @notice Update/Add a deposit type
 * @notice if is updated with empty/0 fields, it will be considered deleted
 * @notice the check for
 * @param index the index of the deposit type if an update is made,
 * @param index or if it is greater than current length than a new deposit is created
 * @param apr annual percentage rate|
 * @param penalty the percent of the staked amount that will be substracted
 * @param duration duration in seconds until maturity
 * @param name the name of the deposit
 * @param canUnstakePriorMaturation bool, if the deposit can be unstaed prior maturity
 */
function updateDepositsType( infinite gas
    uint256 index,

```

“unstaed” should be “**unstaked**”.

Resolution: Correct the spelling.

Status: **Fixed.**

(4) Magic Numbers defined: [enfineoVesting.sol](#)

```

uint256 currentTime = block.timestamp;
uint256 stakeTypeId = 999;
// @dev claim and stake is allowed only between staking start time and vesting start time + 500
if(currentTime < _vestingContractClaimAndStakeStartingDate || currentTime >= _vestingContractStartingDate + 500){
    revert StakeFailed();
}
// @dev if the method is claimed before we start the vesting contract, but after we start the claimand stake per
if(currentTime >= _vestingContractClaimAndStakeStartingDate && currentTime < _vestingContractStartingDate){
    stakeTypeId = 0;
}
// @dev if the method is called after the vesting contract start date, we send the tokens to pool 2
if(currentTime >= _vestingContractStartingDate && currentTime < _vestingContractStartingDate + 500){
    stakeTypeId = 1;
}
uint256 tgeAmount = calculateTgeAmount(vestingScheduleId);
if(vest.stakedAmount == 0
&& vest.releasedAmount == 0
&& (stakeTypeId == 0 || stakeTypeId == 1)){

```

Magic numbers like 999, 0, and 1 reduce code readability and increase the risk of errors. They make the code harder to understand and maintain.

Resolution: Define these values as constants or enumerations with descriptive names, e.g., uint256 constant STAKE_TYPE_POOL1 = 0; This improves code clarity and maintainability, making it easier to understand and modify.

Status: Fixed.

(5) Correct Grammar in Condition Description: [enfineoStaking.sol](#)

```
/**
 * @notice Update/Add a deposit type
 * @notice if is updated with empty/0 fields, it will be considered deleted
 * @notice the check for
 * @param index the index of the deposit type if an update is made,
 * @param index or if it is greater than current length than a new deposit is created
 * @param apr anual percentage rate
 * @param penalty the percent of the staked amount that will be substracted
 * @param duration duration in seconds until maturity
 * @param name the name of the deposit
 * @param canUnstakePriorMaturation bool, if the deposit can be unstaed prior maturity
 */
function updateDepositsType( infinite gas
```

The current text incorrectly uses "than" instead of "then" in the condition description. The intended meaning is to describe a conditional action.

Resolution: Change "than" to "then" to accurately reflect the conditional nature of the statement. The corrected line should read: "Index or, if it is greater than the current length, then a new deposit is created."

Status: Fixed.

(6) Typographical Error in Parameter Description: [enfineoStaking.sol](#)

```
/**
 * @notice Update/Add a deposit type
 * @notice if is updated with empty/0 fields, it will be considered deleted
 * @notice the check for
 * @param index the index of the deposit type if an update is made,
 * @param index or if it is greater than current length than a new deposit is created
 * @param apr anual percentage rate
 * @param penalty the percent of the staked amount that will be substracted
 * @param duration duration in seconds until maturity
 * @param name the name of the deposit
 * @param canUnstakePriorMaturation bool, if the deposit can be unstaed prior maturity
 */
function updateDepositsType( infinite gas
    uint256 index,
```

The current text contains a typographical error with the misspelling of "annual" as "anual." This affects the clarity of the parameter description.

Resolution: Correct the spelling of "anual" to "annual" to accurately describe the parameter. The revised line should read @param apr Annual percentage rate.

Status: **Fixed.**

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet private key would be compromised, then it would create trouble. Following are Admin functions:

enfineoVesting.sol

- setVestingContractStartingDate: Sets the start time of the vesting contract by the admin.
- setEnfToken: Sets the token address by the admin.
- setStakeContractAddress: Sets the stake contract address by the admin.

AccessControl.sol

- grantRole: Grants `role` to `account` can be set by the admin.
- revokeRole: Revokes `role` from `account` by the admin.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

Conclusion

We were given a contract code in the form of a file. And we have used all possible tests based on given objects as files. We observed 1 high, 3 low, and 6 Informational issues in the smart contracts. We confirm that all issues are fixed/Acknowledged in the revised smart contract code. **So, the smart contracts are ready for the mainnet deployment.**

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover the maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

The security state of the reviewed smart contract, based on standard audit procedure scope, is **“Secured”**.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and white box penetration testing. We look at the project's website to get a high-level understanding of the functionality of the software under review. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, and then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status or any other statements of the contract. While we have done our best to conduct the analysis and produce this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

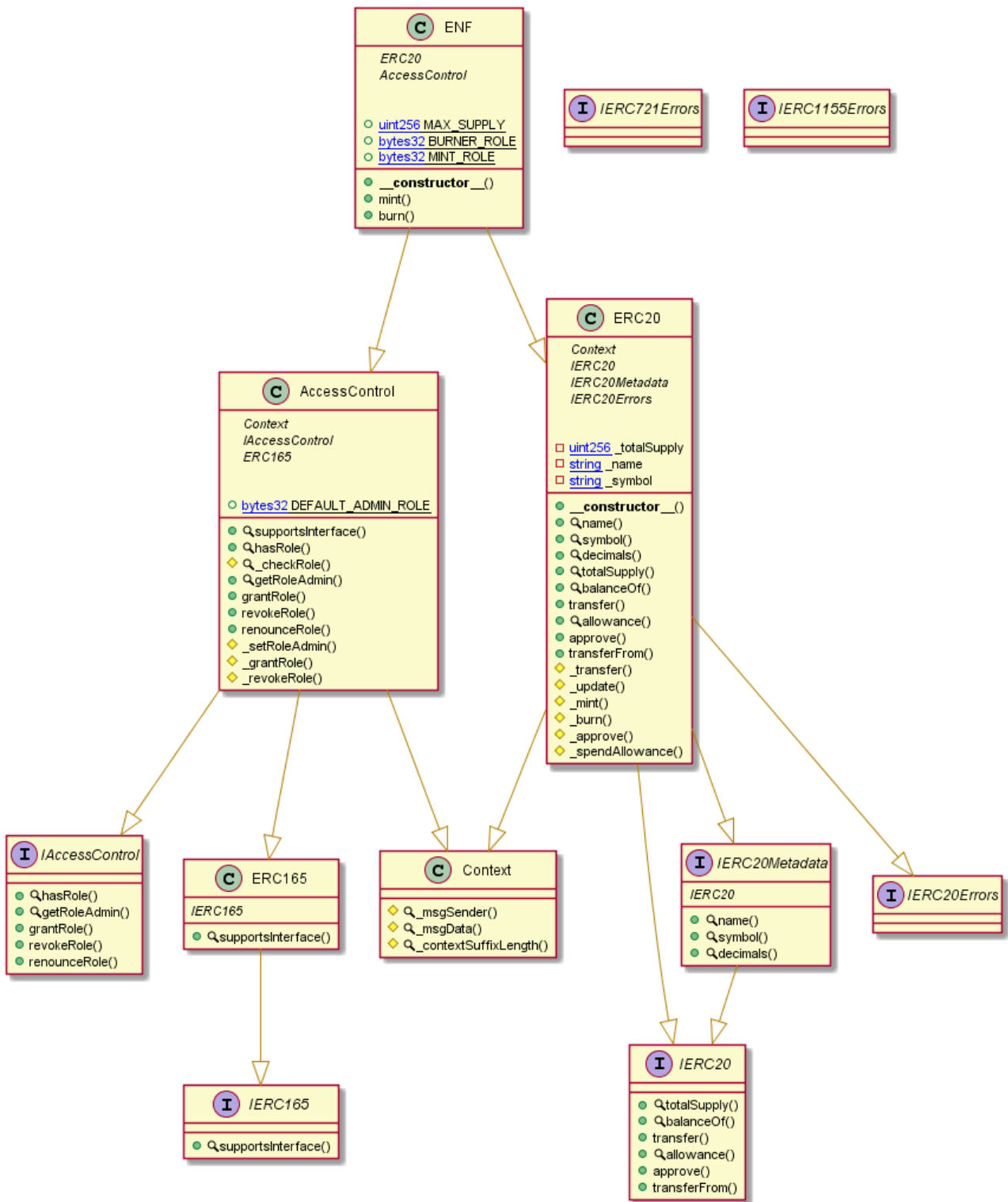
Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.

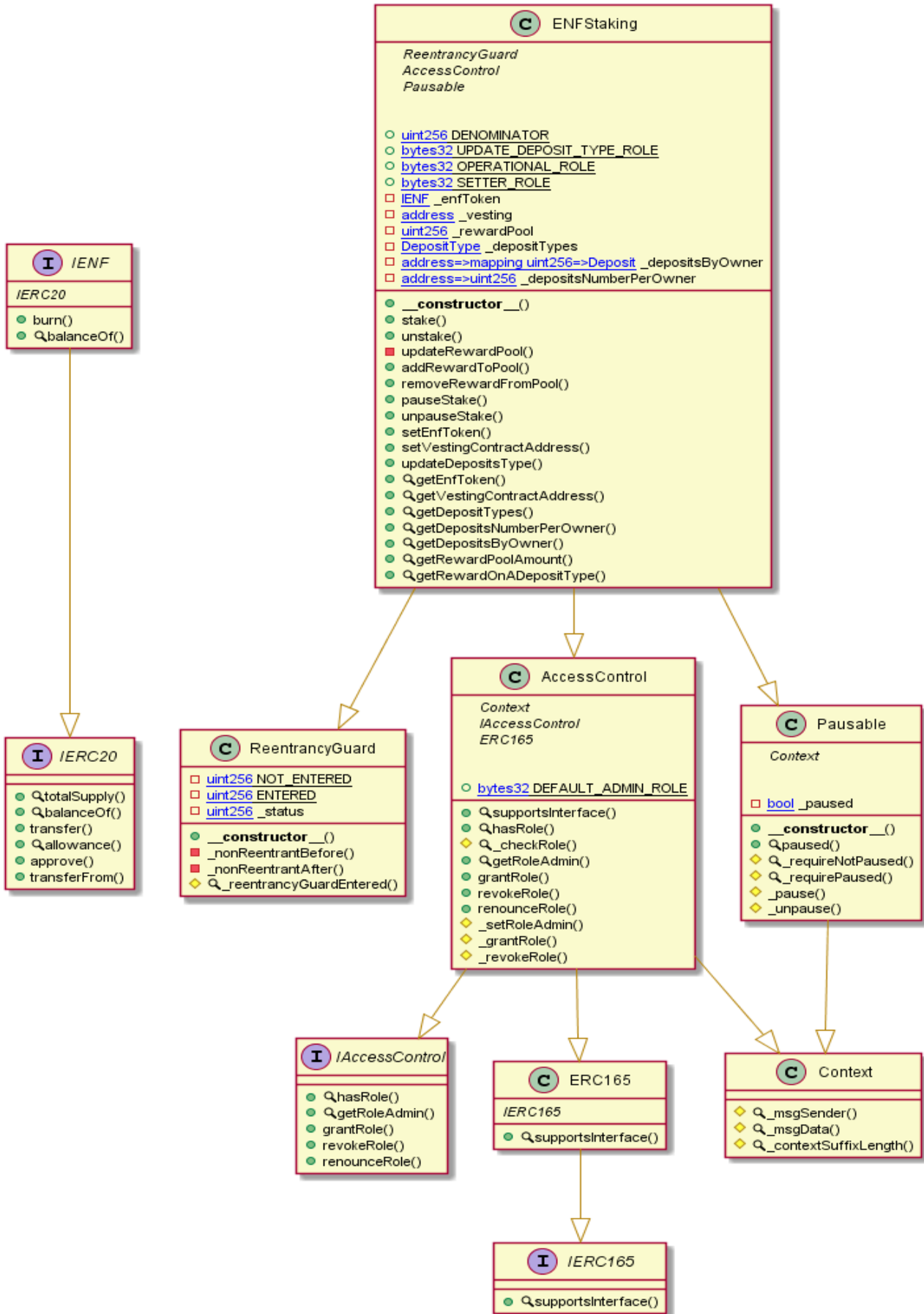
Appendix

Code Flow Diagram - Enfineo

enfineoToken Diagram



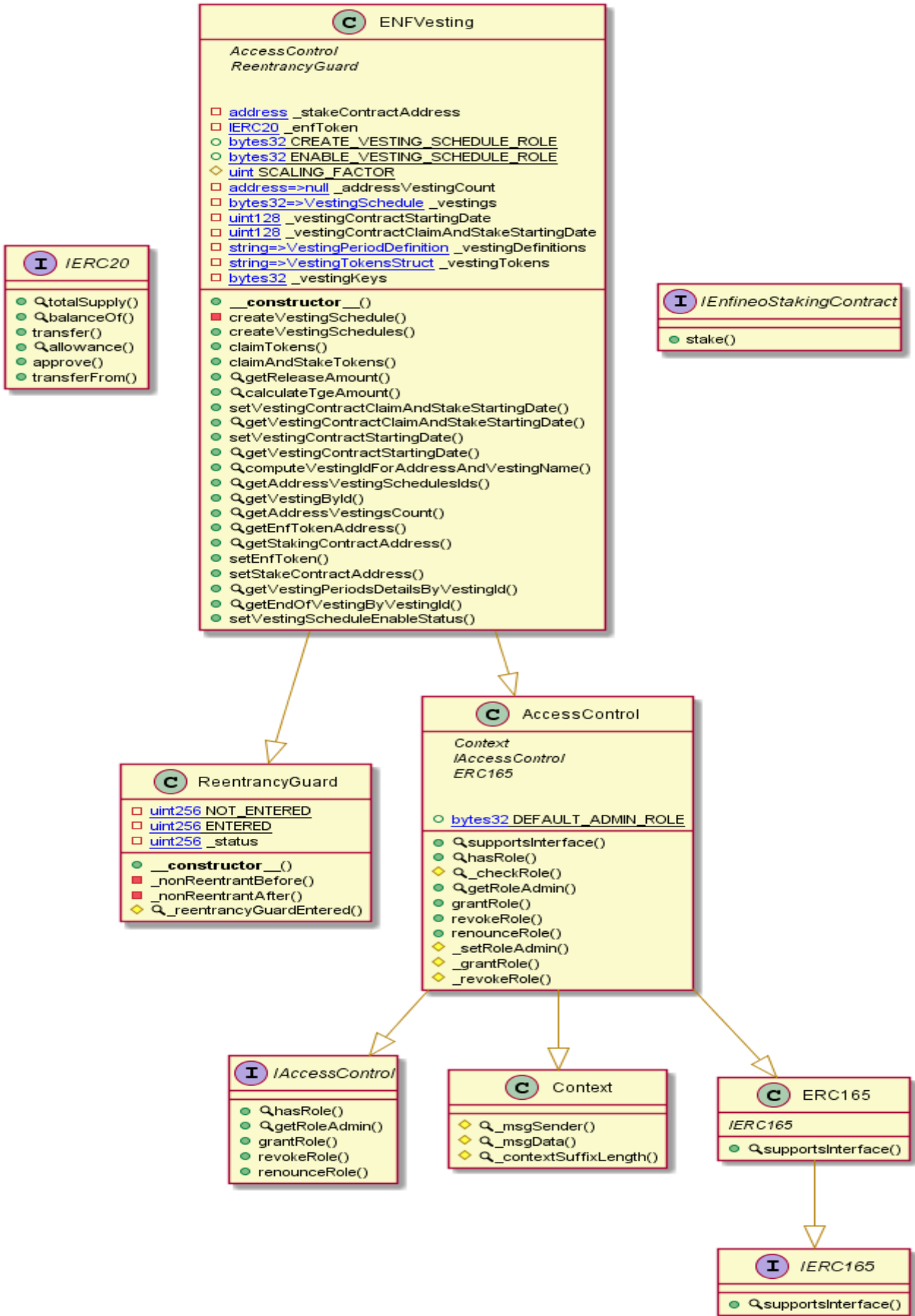
enfineoStaking Diagram



This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

enfineoVesting Diagram



This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

Slither Results Log

Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We did the analysis of the project altogether. Below are the results.

Slither Log >> enfineoToken.sol

```
INFO:Detectors:
Pragma version^0.8.25 (enfineoToken.sol#2) necessitates a version too recent to be trusted.
Consider deploying with 0.8.18.
solc-0.8.25 is not recommended for deployment
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter ENF.burn(address,uint256)._account (enfineoToken.sol#684) is not in mixedCase
Parameter ENF.burn(address,uint256)._amount (enfineoToken.sol#684) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Slither:enfineoToken.sol analyzed (12 contracts with 93 detectors), 7 result(s) found
```

Slither Log >> enfineoStaking.sol

```
INFO:Detectors:
ENFStaking.removeRewardFromPool(address,uint256) (enfineoStaking.sol#728-735) ignores
return value by _enfToken.transfer(to,amount) (enfineoStaking.sol#733)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
ENFStaking.getDepositsByOwner(address,uint256,uint256).step (enfineoStaking.sol#863) is a
local variable never initialized
ENFStaking.getDepositsByOwner(address,uint256,uint256).depositNumber
(enfineoStaking.sol#862) is a local variable never initialized
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
Reentrancy in ENFStaking.removeRewardFromPool(address,uint256)
(enfineoStaking.sol#728-735):
```

External calls:

- `_enfToken.transfer(to,amount)` (enfineoStaking.sol#733)

Event emitted after the call(s):

- `RewardRemovedFromPool(address(_enfToken),to,amount)` (enfineoStaking.sol#734)

Reference:

<https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3>

INFO:Detectors:

`ENFStaking.stake(uint256,uint256,address)` (enfineoStaking.sol#574-656) uses timestamp for comparisons

Dangerous comparisons:

- `block.timestamp + selectedDepositType.duration > type()(uint40).max`

(enfineoStaking.sol#614)

`ENFStaking.unstake(uint256)` (enfineoStaking.sol#662-704) uses timestamp for comparisons

Dangerous comparisons:

- `currentDeposit.maturityTimestamp <= block.timestamp` (enfineoStaking.sol#670)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp>

INFO:Detectors:

`Pragma version^0.8.20` (enfineoStaking.sol#2) necessitates a version too recent to be trusted.

Consider deploying with 0.8.18.

`solc-0.8.25` is not recommended for deployment

Reference:

<https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity>

INFO:Detectors:

Redundant expression "`startIndex` (enfineoStaking.sol#866)" in `ENFStaking`

(enfineoStaking.sol#542-927)

Redundant expression "`step` (enfineoStaking.sol#876)" in `ENFStaking`

(enfineoStaking.sol#542-927)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements>

INFO:Slither:enfineoStaking.sol analyzed (10 contracts with 93 detectors), 20 result(s) found

Slither Log >> enfineoVesting.sol

INFO:Detectors:

Reentrancy in `ENFVesting.claimAndStakeTokens(bytes32)` (enfineoVesting.sol#640-705):

External calls:

- `success = _enfToken.transfer(_stakeContractAddress,tgeAmount)` (enfineoVesting.sol#680)

- `externalContract.stake(tgeAmount,stakingType,vest.beneficiary)`

(enfineoVesting.sol#688-699)

State variables written after the call(s):

- `vest.stakedAmount = vest.stakedAmount - tgeAmount` (enfineoVesting.sol#691)

`ENFVesting._vestings` (enfineoVesting.sol#467) can be used in cross function reentrancies:

- `ENFVesting.calculateTgeAmount(bytes32)` (enfineoVesting.sol#760-765)

- `ENFVesting.createVestingSchedule(address,string,uint256)` (enfineoVesting.sol#537-568)

- `ENFVesting.getEndOfVestingByVestingId(bytes32)` (enfineoVesting.sol#880-890)

- `ENFVesting.getReleaseAmount(bytes32)` (enfineoVesting.sol#711-751)

- `ENFVesting.getVestingById(bytes32)` (enfineoVesting.sol#836-838)

- `ENFVesting.getVestingPeriodsDetailsByVestingId(bytes32)` (enfineoVesting.sol#874-878)

```
- ENFVesting.setVestingScheduleEnableStatus(bytes32[],bool[])
(enfineoVesting.sol#897-912)
- vest.stakedAmount = vest.stakedAmount - tgeAmount (enfineoVesting.sol#695)
ENFVesting._vestings (enfineoVesting.sol#467) can be used in cross function reentrancies:
- ENFVesting.calculateTgeAmount(bytes32) (enfineoVesting.sol#760-765)
- ENFVesting.createVestingSchedule(address,string,uint256) (enfineoVesting.sol#537-568)
- ENFVesting.getEndOfVestingByVestingId(bytes32) (enfineoVesting.sol#880-890)
- ENFVesting.getReleaseAmount(bytes32) (enfineoVesting.sol#711-751)
- ENFVesting.getVestingById(bytes32) (enfineoVesting.sol#836-838)
- ENFVesting.getVestingPeriodsDetailsByVestingId(bytes32) (enfineoVesting.sol#874-878)
- ENFVesting.setVestingScheduleEnableStatus(bytes32[],bool[])
(enfineoVesting.sol#897-912)
```

Reference:

<https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1>

INFO:Detectors:

ENFVesting.setVestingScheduleEnableStatus(bytes32[],bool[]).i (enfineoVesting.sol#901) is a local variable never initialized

Reference:

<https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables>

INFO:Detectors:

ENFVesting.setStakeContractAddress(address).stakeAddress (enfineoVesting.sol#868) lacks a zero-check on :

```
- _stakeContractAddress = stakeAddress (enfineoVesting.sol#869)
```

Reference:

<https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation>

INFO:Detectors:

ENFVesting.claimTokens(bytes32) (enfineoVesting.sol#596-634) uses timestamp for comparisons

Dangerous comparisons:

```
- vest.releasedAmount > vest.vestedAmount (enfineoVesting.sol#618)
- releaseToSend > 0 (enfineoVesting.sol#623)
```

ENFVesting.claimAndStakeTokens(bytes32) (enfineoVesting.sol#640-705) uses timestamp for comparisons

Dangerous comparisons:

```
- currentTime < _vestingContractClaimAndStakeStartingDate || currentTime >=
_vestingContractStartingDate + 2592000 (enfineoVesting.sol#661)
- currentTime >= _vestingContractClaimAndStakeStartingDate && currentTime <
_vestingContractStartingDate (enfineoVesting.sol#665)
- currentTime >= _vestingContractStartingDate && currentTime <
_vestingContractStartingDate + 2592000 (enfineoVesting.sol#669)
```

ENFVesting.getReleaseAmount(bytes32) (enfineoVesting.sol#711-751) uses timestamp for comparisons

Dangerous comparisons:

```
- currentTime < _vestingContractStartingDate (enfineoVesting.sol#716)
- tempTime + vestingDefinitions.vestingPeriods[i] <= currentTime (enfineoVesting.sol#737)
```

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp>

INFO:Detectors:

ENFVesting.claimAndStakeTokens(bytes32) (enfineoVesting.sol#640-705) has a high cyclomatic complexity (13).

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#cyclomatic-complexity>
INFO:Detectors:
Pragma version0.8.25 (enfineoVesting.sol#2) necessitates a version too recent to be trusted.
Consider deploying with 0.8.18.
solc-0.8.25 is not recommended for deployment
Reference:
<https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity>
INFO:Detectors:
Redundant expression "i (enfineoVesting.sol#586)" inENFVesting (enfineoVesting.sol#459-934)
Redundant expression "i (enfineoVesting.sol#825)" inENFVesting (enfineoVesting.sol#459-934)
Redundant expression "i (enfineoVesting.sol#902)" inENFVesting (enfineoVesting.sol#459-934)
Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements>
INFO:Detectors:
ENFVesting.constructor() (enfineoVesting.sol#474-527) uses literals with too many digits:
- _vestingTokens[MARKETING] = VestingTokensStruct(6000000 * 10 ** 18,0)
(enfineoVesting.sol#526)
Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits>
INFO:Slither:enfineoVesting.sol analyzed (9 contracts with 93 detectors), 28 result(s) found

Solidity Static Analysis

enfineoToken.sol

Gas costs:

Gas requirement of function ENF.mint is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 22:4:

Gas costs:

Gas requirement of function ENF.burn is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 31:4:

Similar variable names:

ENF.burn(address,uint256) : Variables have very similar names "_account" and "_amount". Note: Modifiers are currently not considered by this static analysis.

Pos: 32:14:

enfineoStaking.sol

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in ENFStaking.removeRewardFromPool(address,uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

Pos: 201:4:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

Pos: 143:48:

Gas costs:

Gas requirement of function ENFStaking.updateDepositsType is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 248:4:

Constant/View/Pure functions:

ENFStaking.getRewardOnADepositType(uint256,uint256) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

Pos: 360:4:

No return:

IENF.balanceOf(address): Defines a return type but never explicitly returns a value.

Pos: 9:4:

Delete from dynamic array:

Using "delete" on an array leaves a gap. The length of the array remains the same. If you want to remove the empty position you need to shift items manually and update the "length" property.

Pos: 157:11:

Data truncated:

Division of integer values yields an integer value again. That means e.g. $10 / 100 = 0$ instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.

Pos: 365:15:

enfineoToken.sol

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in ENFVesting.claimAndStakeTokens(bytes32): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

Pos: 217:5:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

Pos: 292:30:

Gas costs:

Gas requirement of function ENFVesting.setVestingContractClaimAndStakeStartingDate is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 348:4:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.

Pos: 160:8:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.

Pos: 479:8:

Constant/View/Pure functions:

ENFVesting.computeVestingIdForAddressAndVestingName(address,string) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

Pos: 388:4:

Similar variable names:

ENFVesting.getReleaseAmount(bytes32) : Variables have very similar names "vestingSchedule" and "vestingScheduleId". Note: Modifiers are currently not considered by this static analysis.

Pos: 315:37:

Data truncated:

Division of integer values yields an integer value again. That means e.g. $10 / 100 = 0$ instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.

Pos: 320:37:

Solhint Linter

enfineoToken.sol

```
Compiler version ^0.8.25 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
global import of path @openzeppelin/contracts/token/ERC20/ERC20.sol
is not allowed. Specify names to import individually or bind all
exports of the module into a name (import "path" as Name)
Pos: 1:3
global import of path
@openzeppelin/contracts/access/AccessControl.sol is not allowed.
Specify names to import individually or bind all exports of the
module into a name (import "path" as Name)
Pos: 1:4
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:15
```

enfineoToken.sol

```
Compiler version ^0.8.20 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
global import of path
@openzeppelin/contracts/access/AccessControl.sol is not allowed.
Specify names to import individually or bind all exports of the
module into a name (import "path" as Name)
Pos: 1:3
global import of path
@openzeppelin/contracts/utils/ReentrancyGuard.sol is not allowed.
Specify names to import individually or bind all exports of the
module into a name (import "path" as Name)
Pos: 1:4
global import of path @openzeppelin/contracts/utils/Pausable.sol is
not allowed. Specify names to import individually or bind all exports
of the module into a name (import "path" as Name)
Pos: 1:5
global import of path ./IENF.sol is not allowed. Specify names to
import individually or bind all exports of the module into a name
(import "path" as Name)
Pos: 1:7
global import of path StakingStructs.sol is not allowed. Specify
names to import individually or bind all exports of the module into a
name (import "path" as Name)
Pos: 1:8
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
```

```
Pos: 5:32
Avoid making time-based decisions in your business logic
Pos: 13:86
Avoid making time-based decisions in your business logic
Pos: 36:97
Avoid making time-based decisions in your business logic
Pos: 39:98
Avoid making time-based decisions in your business logic
Pos: 20:121
Avoid making time-based decisions in your business logic
Pos: 20:122
Avoid making time-based decisions in your business logic
Pos: 49:142
```

enfineoToken.sol

```
Compiler version 0.8.25 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1
global import of path @openzeppelin/contracts/token/ERC20/IERC20.sol
is not allowed. Specify names to import individually or bind all
exports of the module into a name (import "path" as Name)
Pos: 1:3
global import of path
@openzeppelin/contracts/access/AccessControl.sol is not allowed.
Specify names to import individually or bind all exports of the
module into a name (import "path" as Name)
Pos: 1:4
global import of path
@openzeppelin/contracts/utils/ReentrancyGuard.sol is not allowed.
Specify names to import individually or bind all exports of the
module into a name (import "path" as Name)
Pos: 1:5
global import of path VestingStructs.sol is not allowed. Specify
names to import individually or bind all exports of the module into a
name (import "path" as Name)
Pos: 1:6
Explicitly mark visibility of state
Pos: 5:41
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:50
Avoid making time-based decisions in your business logic
Pos: 31:234
Avoid making time-based decisions in your business logic
Pos: 31:291
```

Software analysis result:

This software reported many false positive results and some are informational issues. So, those issues can be safely ignored.

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io



This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io