Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Athame Finance Protocol
Website:	athame.finance
Platform:	Avalanche Network
Language:	Solidity
Date:	April 24th, 2022

Table of contents

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.

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contracted by the Athame Finance team to perform the Security audit of the Depository, Treasury and ATHF Token smart contracts codes. 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 April 20th, 2022.

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

- Athame Finance is a DAAS or DeFi As A Service protocol running on the Avalanche Network. DAAS is defined as paying someone else to invest your capital. Through governance we invest your capital in various DeFi protocols, tokens, coins and NFT's across multiple blockchains.
- Athame Finance Governance is the heart and brain of the Athame Protocol.
- Anyone can be part of the Athame Governance Process, and anyone with ATHF voting power can vote on proposals. Governance makes decisions about new features and directions of where the Athame Protocol should go. Governance is the ultimate decision body for the Athame Protocol; no one can override vote results.
- The Athame Finance contract inherits the ERC20, Ownable, SafeMath, SafeERC20, Pausable, ERC20Burnable, AccessControl 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 Athame Finance Protocol Smart Contracts**
Platform	**Avalanche / Solidity**
File 1	AthameDepository.sol
File 1 Git Commit	28ca49dc785c9b511ee1515dd2a1c1bdbd74c56f

File 2	AthameTreasury.sol
File 2 Git Commit	b5fdd955a742d2f9fe8a82d02bdb9c51c00f72be
File 3	AthameToken.sol
File 3 Git Commit	cf94f0ba1a985858b6a103d4648879305e75ffdf
Audit Date	April 20th, 2022
Revision Date	April 24th, 2022

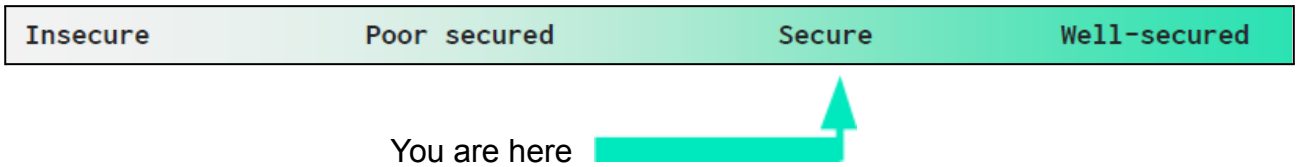
Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
File 1 AthameDepository.sol • Default vesting periods: 7 days • Default fee: 10% • The owner can pause/unpause buying shares. • The owner can set the share price and also can change the deposit token.	**YES, This is valid.**
File 2 AthameTreasury.sol • The owner can deposit / withdraw the tokens.	**YES, This is valid.**
File 3 AthameToken.sol • Name: Athame Finance Token • Symbol: ATHF • Decimals: 18 • Token supply: unlimited (but controlled by AthameDepository contract after ownership of ATHF contract renounced)	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Secured"**. Also, these contracts do 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 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. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 2 critical, 1 high, 0 medium and 1 low and some very low level issues. These issues are fixed / acknowledged in the revised code.

Investors Advice: 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	Solidity version not specified	Passed
	Solidity version too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of 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	Moderated
	High consumption ‘storage’ storage	Passed
	SathMath unnecessary usage	Passed
Business Risk	The maximum limit for mintage not set	Acknowledged
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Code Quality

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

The libraries in the Athame Finance Protocol 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 Athame Finance Protocol.

The Athame Finance team has provided unit test scripts, which have helped to determine the integrity of the code in an automated way.

Code parts are **not** well commented on smart contracts.

Documentation

We were given an Athame Finance Protocol smart contract code in the form of the github links. The git commits of that code are mentioned above in the table.

As mentioned above, code parts are not well commented. So it is not easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Another source of information was its official website https://athame.finance which provided rich information about the project architecture and tokenomics.

Use of Dependencies

As per our observation, the libraries are used in this smart contracts infrastructure that 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

AthameDepository.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	_transferOwnership	internal	Passed	No Issue
7	paused	read	Passed	No Issue
8	whenNotPaused	modifier	Passed	No Issue
9	whenPaused	modifier	Passed	No Issue
10	_pause	internal	Passed	No Issue
11	_unpause	internal	Passed	No Issue
12	pause	external	access only Owner	No Issue
13	unpause	external	access only Owner	No Issue
14	setDepositToken	external	access only Owner	No Issue
15	setSharePrice	external	access only Owner	No Issue
16	deposit	external	access only Owner	No Issue
17	withdraw	external	access only Owner	No Issue
18	buyShares	external	Passed	No Issue
19	claim	external	Passed	No Issue
20	getBalance	read	Passed	No Issue
21	updateInvestors	write	Passed	No Issue
22	getDaysPassed	read	Passed	No Issue

AthameToken.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	_transferOwnership	internal	Passed	No Issue
7	name	read	Passed	No Issue
8	symbol	read	Passed	No Issue
9	decimals	read	Passed	No Issue
10	totalSupply	read	Passed	No Issue
11	balanceOf	read	Passed	No Issue
12	transfer	write	Passed	No Issue
13	allowance	read	Passed	No Issue

14	approve	write	Passed	No Issue
15	transferFrom	write	Passed	No Issue
16	increaseAllowance	write	Passed	No Issue
17	decreaseAllowance	write	Passed	No Issue
18	_transfer	internal	Passed	No Issue
19	_mint	internal	Passed	No Issue
20	_burn	internal	Passed	No Issue
21	_approve	internal	Passed	No Issue
22	_spendAllowance	internal	Passed	No Issue
23	_beforeTokenTransfer	internal	Passed	No Issue
24	_afterTokenTransfer	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	getRoleAdmin	read	Passed	No Issue
30	grantRole	write	access only Role	No Issue
31	revokeRole	write	access only Role	No Issue
32	renounceRole	write	Passed	No Issue
33	_setupRole	internal	Passed	No Issue
34	_setRoleAdmin	internal	Passed	No Issue
35	_grantRole	internal	Passed	No Issue
36	_revokeRole	internal	Passed	No Issue
37	burn	write	Passed	No Issue
38	burnFrom	write	Passed	No Issue
39	mint	write	Unlimited minting	Ownership will be renounced to control minting by Depository contract
40	grantMinterRole	external	access only Role	No Issue
41	revokeMinterRole	external	access only Role	No Issue

AthameTreasury.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	_transferOwnership	internal	Passed	No Issue
7	onlyRole	modifier	Passed	No Issue
8	supportsInterface	read	Passed	No Issue
9	hasRole	read	Passed	No Issue
10	_checkRole	internal	Passed	No Issue
11	getRoleAdmin	read	Passed	No Issue

12	grantRole	write	access only Role	No Issue
13	revokeRole	write	access only Role	No Issue
14	renounceRole	write	Passed	No Issue
15	_setupRole	internal	Passed	No Issue
16	_setRoleAdmin	internal	Passed	No Issue
17	_grantRole	internal	Passed	No Issue
18	_revokeRole	internal	Passed	No Issue
19	withdraw	external	Passed	No Issue
20	deposit	external	Passed	No Issue
21	getBalance	read	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 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 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

(1) Use of unbounded loops - AthameDepository.sol

```
125         // get total vested shares
126         for (uint32 i = 0; i < accounts.length; i++) {
127             address currentHolder = accounts[i];
128             Investor memory investor = investors[currentHolder];
129             vestedShares = vestedShares.add(investor.totalShareCount);
130         }
```

In deposit function (line number #125), unbounded loops are used for calculating and transferring reward. "accounts" array's length can reach a very big number. This will break the function and contract as the loop will hit the "block gas limit".

Resolution: We suggest optimizing the loop. Execute loop logic in batches. e.g- process the first 20 elements of the “accounts” array at a time, then next 20 elements.. until the whole array is processed.

Status: Refactored and Fixed the issue

(2) Rugpull situation - AthameDepository.sol

```
154     function withdraw(uint256 _amount) external onlyOwner {
155         IERC20(depositToken).safeTransfer(msg.sender, _amount);
156
157         totalUnclaimed = totalUnclaimed.sub(_amount);
158
159         emit Withdrawal(depositToken, _amount);
160     }
```

In withdraw function (line number #154), there is a situation of the owner transfering all unclaimed funds of all users to his wallet. This is a possible rugpull situation.

Resolution: We suggest removing this function if not needed.

Status: Removed this function and Fixed the issue

High Severity

(1) Max minting limit is not set - ATHF token contract

```
function mint(address to, uint256 amount) public onlyRole(MINTER_ROLE) {
    _mint(to, amount);
}
```

The owner (via minter role wallet) can mint unlimited ATHF tokens. Unlimited token minting is not good for healthy tokenomics.

Resolution: We suggest renouncing the ownership of the ATHF token contract after setting the minting role to the AthameDepository contract. This will remove the "human influence" and make token minting fully decentralized by the AthameDepository contract.

Status: Acknowledged

Medium

No medium severity vulnerabilities found.

Low

(1) Use of SafeMath is unnecessary - All three contracts

```
    using SafeMath for uint256;
```

Solidity version over 0.8.0 comes with in-built integer overflow / underflow protection. Therefore the use of SafeMath is not necessary. On the other hand, this does not raise any security issues, but It saves some gas if that is removed.

Resolution: We advise removing it.

Status: Removed SafeMath and thus this issue is Fixed

Very Low / Informational / Best practices:

(1) Redundant condition - AthameDepository.sol

```
address public feeCollector;
```

The "fee" variable (line number #45) does not have a "constant" keyword as it is meant to be a constant.

Resolution: We suggest using "constant" in the variable definition.

Status: Fixed

(2) Redundant condition - AthameDepository.sol

```
        // Transfer the fee
        if (fee != 0) {
            IERC20(depositToken).safeTransferFrom(
                msg.sender,
                feeCollector,
                totalFee
            );
```

Because the “fee” is a constant (line number #112), the fee's value is going to be more than 0. So here the checking condition is of no use. It's redundant.

Resolution: We suggest removing this condition.

Status: Condition removed and fixed the issue

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:

- withdraw: Depository and Treasury owners can withdraw the amount.
- deposit: Depository and Treasury owners can deposit an amount.
- setSharePrice: AthameDepository owner can set share price.
- setDepositToken: AthameDepository owner can set deposit tokens.
- unpause: AthameDepository owner can return to normal state.
- pause: AthameDepository owner can trigger stopped state.
- mint: Minter role wallet can mint tokens.

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 github links. And we have used all possible tests based on given objects as files. We have observed some issues in the smart contracts and they are resolved / acknowledged. **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 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.

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

Security state of the reviewed 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 whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. 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, 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 are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing 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.

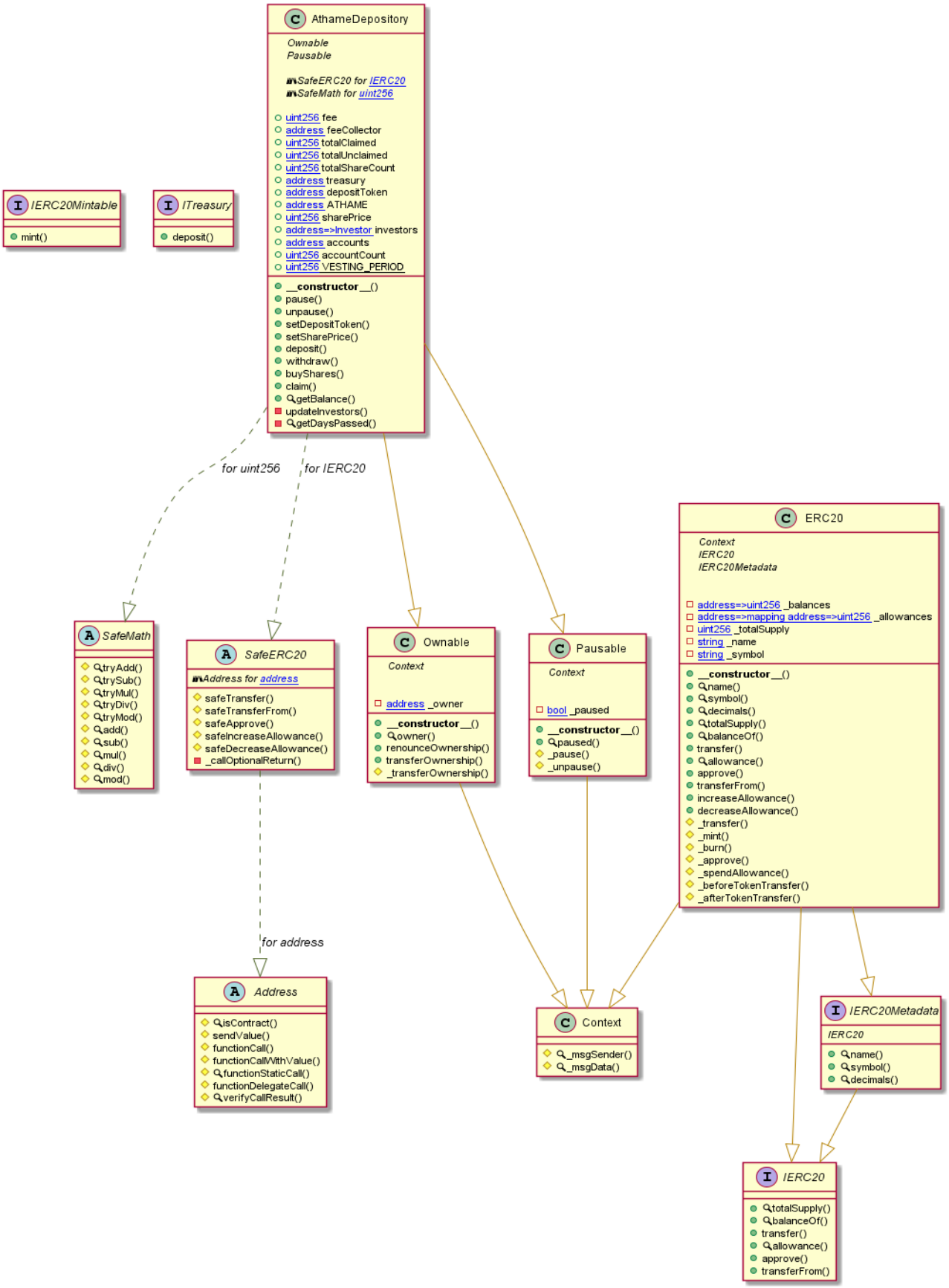
Technical Disclaimer

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 explicit security of the audited smart contracts.

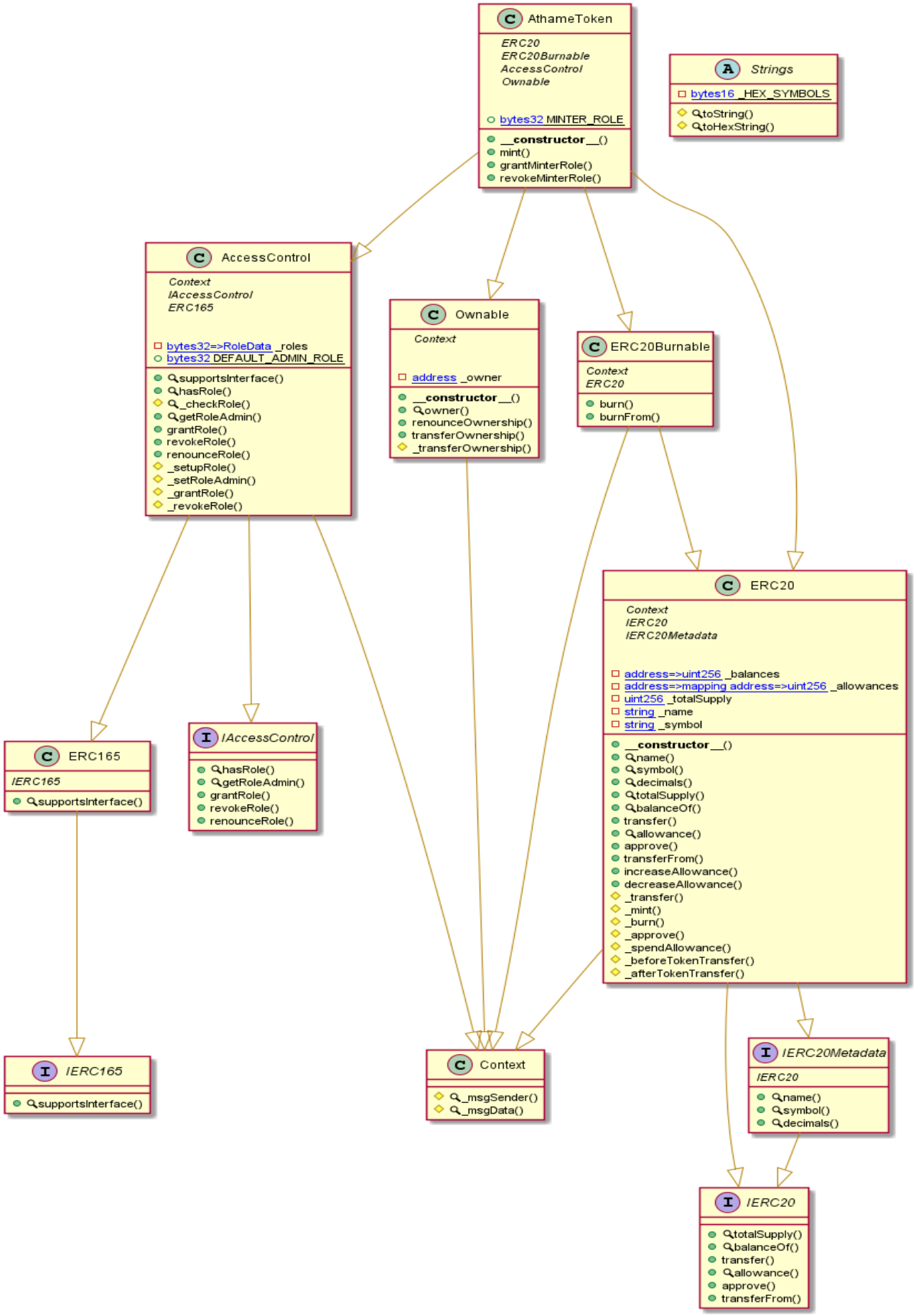
Appendix

Code Flow Diagram - Athame Finance Protocol

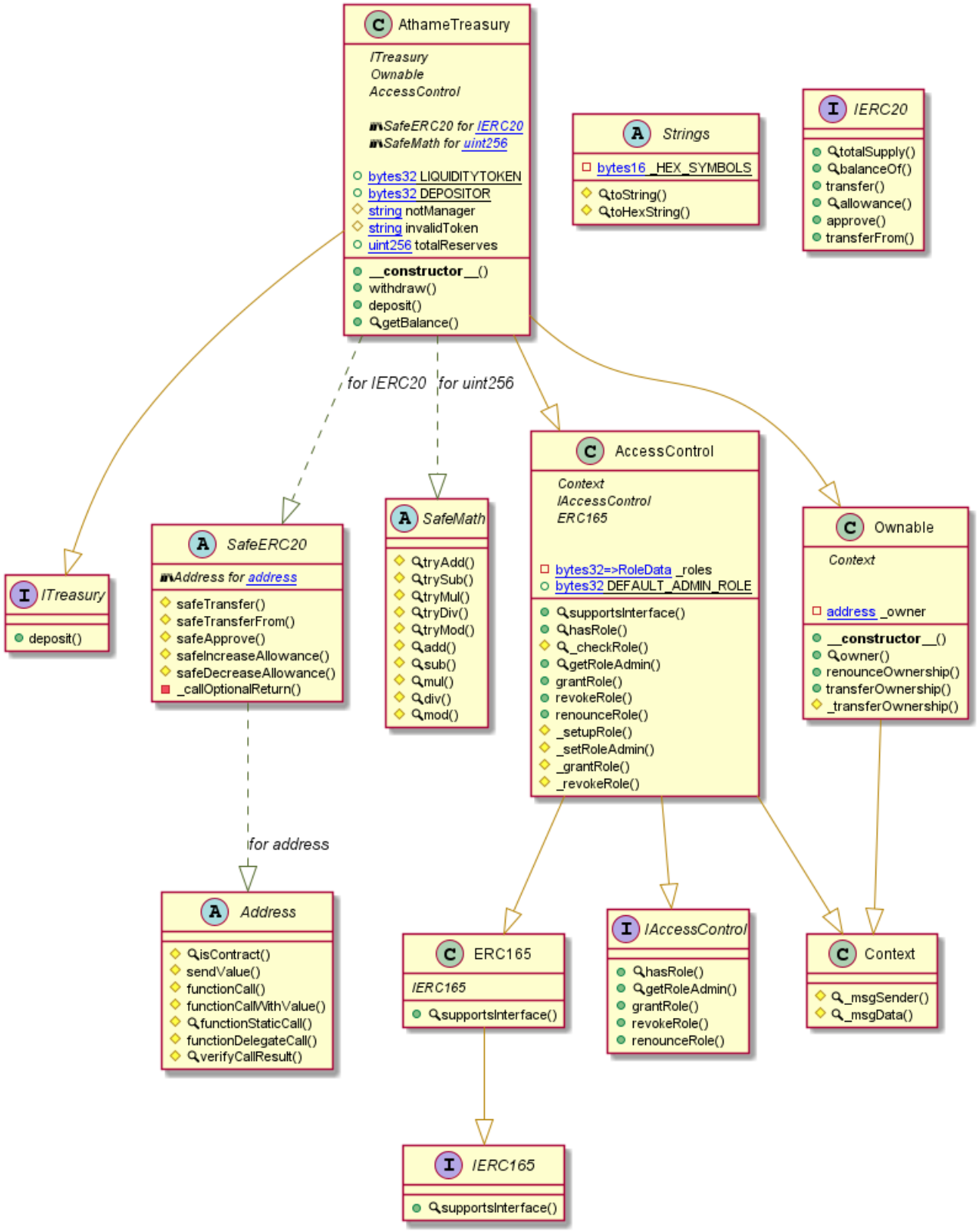
AthameDepository Diagram



AthameToken Diagram



AthameTreasury Diagram



Slither Results Log

Slither log >> AthameDepository.sol

```
INFO:Detectors:
AthameDepository.constructor(address,address,address,address,uint256)._feeCollector (AthameDepository.sol#1161) lacks a zero-c
ck on :
                - feeCollector = _feeCollector (AthameDepository.sol#1166)
AthameDepository.constructor(address,address,address,address,uint256)._treasury (AthameDepository.sol#1158) lacks a zero-check
n :
                - treasury = _treasury (AthameDepository.sol#1167)
AthameDepository.constructor(address,address,address,address,uint256)._athame (AthameDepository.sol#1159) lacks a zero-check o
:
                - ATHAME = _athame (AthameDepository.sol#1168)
AthameDepository.constructor(address,address,address,address,uint256)._depositToken (AthameDepository.sol#1160) lacks a zero-c
ck on :
                - depositToken = _depositToken (AthameDepository.sol#1169)
AthameDepository.setDepositToken(address)._token (AthameDepository.sol#1185) lacks a zero-check on :
                - depositToken = _token (AthameDepository.sol#1186)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in AthameDepository.buyShares(uint256) (AthameDepository.sol#1259-1301):
        External calls:
        - IERC20(depositToken).safeTransferFrom(msg.sender,address(this),_totalAmount) (AthameDepository.sol#1267-1271)
        - IERC20(depositToken).approve(address(treasury),_totalAmount) (AthameDepository.sol#1272)
        - ITreasury(treasury).deposit(_totalAmount,depositToken) (AthameDepository.sol#1273)
        State variables written after the call(s):
        - accountCount += 1 (AthameDepository.sol#1282)
        - accounts.push(msg.sender) (AthameDepository.sol#1281)
        - investors[msg.sender].account = msg.sender (AthameDepository.sol#1280)
        - totalShareCount = totalShareCount.add(_shareCount) (AthameDepository.sol#1276)
Reentrancy in AthameDepository.claim() (AthameDepository.sol#1303-1323):
        External calls:
        - IERC20(depositToken).safeTransfer(msg.sender,investor.unclaimedDividends) (AthameDepository.sol#1313-1316)
        State variables written after the call(s):
        - totalClaimed = totalClaimed.add(investor.unclaimedDividends) (AthameDepository.sol#1318)
        - totalUnclaimed = totalUnclaimed.sub(investor.unclaimedDividends) (AthameDepository.sol#1317)
Reentrancy in AthameDepository.deposit(uint256) (AthameDepository.sol#1194-1247):
        External calls:
        - IERC20(depositToken).safeTransferFrom(msg.sender,address(this),finalAmount) (AthameDepository.sol#1200-1204)
        - IERC20(depositToken).safeTransferFrom(msg.sender,feeCollector,totalFee) (AthameDepository.sol#1208-1212)
```

```
        - IERC20(depositToken).safeTransferFrom(msg.sender,feeCollector,totalFee) (AthameDepository.sol#1208-1212)
        State variables written after the call(s):
        - updateInvestors() (AthameDepository.sol#1216)
                - investor.investments[x].vested = true (AthameDepository.sol#1351)
                - investor.totalShareCount = investor.totalShareCount.add(investor.investments[x].shareCount) (AthameDepositor
sol#1354-1356)
        - investor_scope_2.unclaimedDividends = investor_scope_2.unclaimedDividends.add(rewardToBeDistributed) (AthameDeposito
.sol#1238-1240)
        - totalUnclaimed = totalUnclaimed.add(finalAmount) (AthameDepository.sol#1244)
Reentrancy in AthameDepository.withdraw(uint256) (AthameDepository.sol#1249-1255):
        External calls:
        - IERC20(depositToken).safeTransfer(msg.sender,_amount) (AthameDepository.sol#1250)
        State variables written after the call(s):
        - totalUnclaimed = totalUnclaimed.sub(_amount) (AthameDepository.sol#1252)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in AthameDepository.buyShares(uint256) (AthameDepository.sol#1259-1301):
        External calls:
        - IERC20(depositToken).safeTransferFrom(msg.sender,address(this),_totalAmount) (AthameDepository.sol#1267-1271)
        - IERC20(depositToken).approve(address(treasury),_totalAmount) (AthameDepository.sol#1272)
        - ITreasury(treasury).deposit(_totalAmount,depositToken) (AthameDepository.sol#1273)
        - IERC20Mintable(ATHAME).mint(msg.sender,_shareCount * 10 ** ERC20(ATHAME).decimals()) (AthameDepository.sol#1286-1289
        Event emitted after the call(s):
        - OnInvestment(_shareCount,sharePrice.mul(_shareCount),msg.sender) (AthameDepository.sol#1300)
Reentrancy in AthameDepository.claim() (AthameDepository.sol#1303-1323):
        External calls:
        - IERC20(depositToken).safeTransfer(msg.sender,investor.unclaimedDividends) (AthameDepository.sol#1313-1316)
        Event emitted after the call(s):
        - Claim(msg.sender,investor.unclaimedDividends) (AthameDepository.sol#1320)
Reentrancy in AthameDepository.deposit(uint256) (AthameDepository.sol#1194-1247):
        External calls:
        - IERC20(depositToken).safeTransferFrom(msg.sender,address(this),finalAmount) (AthameDepository.sol#1200-1204)
        - IERC20(depositToken).safeTransferFrom(msg.sender,feeCollector,totalFee) (AthameDepository.sol#1208-1212)
        Event emitted after the call(s):
        - Deposit(finalAmount) (AthameDepository.sol#1246)
Reentrancy in AthameDepository.withdraw(uint256) (AthameDepository.sol#1249-1255):
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
AthameDepository.updateInvestors() (AthameDepository.sol#1340-1360) uses timestamp for comparisons
        Dangerous comparisons:
        - ! investor.investments[x].vested || getDaysPassed(investor.investments[x].created) >= VESTING_PERIOD (AthameDeposito
.sol#1347-1349)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.verifyCallResult(bool,bytes,string) (AthameDepository.sol#493-513) uses assembly
        - INLINE ASM (AthameDepository.sol#505-508)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Address.functionCall(address,bytes) (AthameDepository.sol#377-379) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (AthameDepository.sol#406-412) is never used and should be removed
Address.functionDelegateCall(address,bytes) (AthameDepository.sol#466-468) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (AthameDepository.sol#476-485) is never used and should be removed
Address.functionStaticCall(address,bytes) (AthameDepository.sol#439-441) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (AthameDepository.sol#449-458) is never used and should be removed
Address.sendValue(address,uint256) (AthameDepository.sol#352-357) is never used and should be removed
Context._msgData() (AthameDepository.sol#619-621) is never used and should be removed
ERC20._burn(address,uint256) (AthameDepository.sol#1003-1018) is never used and should be removed
ERC20._mint(address,uint256) (AthameDepository.sol#980-990) is never used and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (AthameDepository.sol#542-555) is never used and should be removed
```


Email: audit@EtherAuthority.io

```
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (AthameDepository.sol#658-660)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (AthameDepository.sol#666-669)
name() should be declared external:
        - ERC20.name() (AthameDepository.sol#785-787)
symbol() should be declared external:
        - ERC20.symbol() (AthameDepository.sol#793-795)
decimals() should be declared external:
        - ERC20.decimals() (AthameDepository.sol#810-812)
totalSupply() should be declared external:
        - ERC20.totalSupply() (AthameDepository.sol#817-819)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (AthameDepository.sol#824-826)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (AthameDepository.sol#836-840)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (AthameDepository.sol#859-863)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (AthameDepository.sol#881-890)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (AthameDepository.sol#904-908)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (AthameDepository.sol#924-933)
getBalance() should be declared external:
        - AthameDepository.getBalance() (AthameDepository.sol#1330-1332)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:AthameDepository.sol analyzed (12 contracts with 75 detectors), 66 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Slither log >> AthameToken.sol

```
INFO:Detectors:
AccessControl._setRoleAdmin(bytes32,bytes32) (AthameToken.sol#830-834) is never used and should be removed
AccessControl._setupRole(bytes32,address) (AthameToken.sol#821-823) is never used and should be removed
Context._msgData() (AthameToken.sol#259-261) is never used and should be removed
Strings.toHexString(uint256) (AthameToken.sol#55-66) is never used and should be removed
Strings.toString(uint256) (AthameToken.sol#30-50) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.4 (AthameToken.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (AthameToken.sol#299-301)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (AthameToken.sol#307-310)
name() should be declared external:
        - ERC20.name() (AthameToken.sol#350-352)
symbol() should be declared external:
        - ERC20.symbol() (AthameToken.sol#358-360)
decimals() should be declared external:
        - ERC20.decimals() (AthameToken.sol#375-377)
totalSupply() should be declared external:
        - ERC20.totalSupply() (AthameToken.sol#382-384)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (AthameToken.sol#389-391)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (AthameToken.sol#401-405)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (AthameToken.sol#424-428)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (AthameToken.sol#446-455)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (AthameToken.sol#469-473)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (AthameToken.sol#489-498)
```

```
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (AthameToken.sol#489-498)
grantRole(bytes32,address) should be declared external:
        - AccessControl.grantRole(bytes32,address) (AthameToken.sol#766-768)
revokeRole(bytes32,address) should be declared external:
        - AccessControl.revokeRole(bytes32,address) (AthameToken.sol#779-781)
renounceRole(bytes32,address) should be declared external:
        - AccessControl.renounceRole(bytes32,address) (AthameToken.sol#797-801)
burn(uint256) should be declared external:
        - ERC20Burnable.burn(uint256) (AthameToken.sol#867-869)
burnFrom(address,uint256) should be declared external:
        - ERC20Burnable.burnFrom(address,uint256) (AthameToken.sol#882-885)
mint(address,uint256) should be declared external:
        - AthameToken.mint(address,uint256) (AthameToken.sol#895-897)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:AthameToken.sol analyzed (12 contracts with 75 detectors), 25 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Slither log >> AthameTreasury.sol

```
INFO:Detectors:
Reentrancy in AthameTreasury.deposit(uint256,address) (AthameTreasury.sol#1044-1059):
        External calls:
        - IERC20(_token).safeTransferFrom(msg.sender,address(this),_amount) (AthameTreasury.sol#1052)
        State variables written after the call(s):
        - totalReserves = totalReserves.add(_amount) (AthameTreasury.sol#1054)
Reentrancy in AthameTreasury.withdraw(uint256,address) (AthameTreasury.sol#1033-1042):
        External calls:
        - IERC20(_token).safeTransfer(msg.sender,_amount) (AthameTreasury.sol#1037)
        State variables written after the call(s):
        - totalReserves = totalReserves.sub(_amount) (AthameTreasury.sol#1039)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in AthameTreasury.deposit(uint256,address) (AthameTreasury.sol#1044-1059):
        External calls:
        - IERC20(_token).safeTransferFrom(msg.sender,address(this),_amount) (AthameTreasury.sol#1052)
        Event emitted after the call(s):
        - Deposit(_token,_amount) (AthameTreasury.sol#1056)
Reentrancy in AthameTreasury.withdraw(uint256,address) (AthameTreasury.sol#1033-1042):
        External calls:
        - IERC20(_token).safeTransfer(msg.sender,_amount) (AthameTreasury.sol#1037)
        Event emitted after the call(s):
        - Withdrawal(_token,_amount) (AthameTreasury.sol#1041)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Address.verifyCallResult(bool,bytes,string) (AthameTreasury.sol#434-454) uses assembly
        - INLINE ASM (AthameTreasury.sol#446-449)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
AccessControl._setRoleAdmin(bytes32,bytes32) (AthameTreasury.sol#919-923) is never used and should be removed
AccessControl._setupRole(bytes32,address) (AthameTreasury.sol#910-912) is never used and should be removed
Address.functionCall(address,bytes) (AthameTreasury.sol#318-320) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (AthameTreasury.sol#347-353) is never used and should be removed
Address.functionDelegateCall(address,bytes) (AthameTreasury.sol#407-409) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (AthameTreasury.sol#417-426) is never used and should be removed
```

```
SafeMath.sub(uint256,uint256,string) (AthameTreasury.sol#691-700) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (AthameTreasury.sol#545-551) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (AthameTreasury.sol#587-592) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (AthameTreasury.sol#599-604) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (AthameTreasury.sol#570-580) is never used and should be removed
SafeMath.trySub(uint256,uint256) (AthameTreasury.sol#558-563) is never used and should be removed
Strings.toHexString(uint256) (AthameTreasury.sol#58-69) is never used and should be removed
Strings.toString(uint256) (AthameTreasury.sol#33-53) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.4 (AthameTreasury.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.
7.6
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (AthameTreasury.sol#293-298):
        - (success) = recipient.call{value: amount}() (AthameTreasury.sol#296)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (AthameTreasury.sol#361-372):
        - (success,returndata) = target.call{value: value}(data) (AthameTreasury.sol#370)
Low level call in Address.functionStaticCall(address,bytes,string) (AthameTreasury.sol#390-399):
        - (success,returndata) = target.staticcall(data) (AthameTreasury.sol#397)
Low level call in Address.functionDelegateCall(address,bytes,string) (AthameTreasury.sol#417-426):
        - (success,returndata) = target.delegatecall(data) (AthameTreasury.sol#424)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter AthameTreasury.withdraw(uint256,address)._amount (AthameTreasury.sol#1033) is not in mixedCase
Parameter AthameTreasury.withdraw(uint256,address)._token (AthameTreasury.sol#1033) is not in mixedCase
```

```
INFO:Detectors:
Parameter AthameTreasury.withdraw(uint256,address)._amount (AthameTreasury.sol#1033) is not in mixedCase
Parameter AthameTreasury.withdraw(uint256,address)._token (AthameTreasury.sol#1033) is not in mixedCase
Parameter AthameTreasury.deposit(uint256,address)._amount (AthameTreasury.sol#1044) is not in mixedCase
Parameter AthameTreasury.deposit(uint256,address)._token (AthameTreasury.sol#1044) is not in mixedCase
Parameter AthameTreasury.getBalance(address)._token (AthameTreasury.sol#1066) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
AthameTreasury.invalidToken (AthameTreasury.sol#1022) should be constant
AthameTreasury.notManager (AthameTreasury.sol#1021) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
grantRole(bytes32,address) should be declared external:
        - AccessControl.grantRole(bytes32,address) (AthameTreasury.sol#855-857)
revokeRole(bytes32,address) should be declared external:
        - AccessControl.revokeRole(bytes32,address) (AthameTreasury.sol#868-870)
renounceRole(bytes32,address) should be declared external:
        - AccessControl.renounceRole(bytes32,address) (AthameTreasury.sol#886-890)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (AthameTreasury.sol#984-986)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (AthameTreasury.sol#992-995)
getBalance(address) should be declared external:
        - AthameTreasury.getBalance(address) (AthameTreasury.sol#1066-1069)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:AthameTreasury.sol analyzed (13 contracts with 75 detectors), 50 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity Static Analysis

AthameDepository.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Address.functionCallWithValue(address,bytes,uint256,string): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 420:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in AthameDepository.claim(): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 1303:4:

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.
more
Pos: 1366:28:

Gas & Economy

Gas costs:

Gas requirement of function AthameDepository.claim 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: 1303:4:

Gas costs:

Gas requirement of function AthameDepository.getBalance 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: 1330:4:

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.
more
Pos: 1231:12:

Email: audit@EtherAuthority.io

Miscellaneous

Constant/View/Pure functions:

IERC20Mintable.mint(address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 7:4:

Constant/View/Pure functions:

AthameDepository.updateInvestors() : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 1340:4:

Similar variable names:

AthameDepository.claim() : Variables have very similar names "investor" and "investors". Note: Modifiers are currently not considered by this static analysis.

Pos: 1322:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

more

Pos: 1307:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

more

Pos: 1308:8:

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: 1229:37:

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: 1367:15:

AthameToken.sol

Gas & Economy

Gas costs:

Gas requirement of function AthameToken.name 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: 350:4:

Gas costs:

Gas requirement of function AthameToken.grantMinterRole 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: 899:4:

Gas costs:

Gas requirement of function AthameToken.revokeMinterRole 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: 906:4:

Miscellaneous

Constant/View/Pure functions:

ERC165.supportsInterface(bytes4) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 20:4:

Similar variable names:

ERC20Burnable.burnFrom(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 884:23:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 798:8:

No return:

IERC20.totalSupply(): Defines a return type but never explicitly returns a value.
Pos: 182:4:

AthameTreasury.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in SafeERC20.safeDecreaseAllowance(contract IERC20,address,uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 507:4:

Low level calls:

Use of "delegatecall": should be avoided whenever possible. External code, that is called can change the state of the calling contract and send ether from the caller's balance. If this is wanted behaviour, use the Solidity library feature if possible.

more

Pos: 424:50:

Gas & Economy

Gas costs:

Gas requirement of function AthameTreasury.deposit 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: 1044:4:

Gas costs:

Gas requirement of function AthameTreasury.getBalance 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: 1066:4:

Miscellaneous

Constant/View/Pure functions:

AccessControl._checkRole(bytes32,address) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 820:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

more

Pos: 1046:8:

AthameDepository.sol

```
AthameDepository.sol:571:18: Error: Parse error: missing ';' at '{'
AthameDepository.sol:928:18: Error: Parse error: missing ';' at '{'
AthameDepository.sol:961:18: Error: Parse error: missing ';' at '{'
AthameDepository.sol:1010:18: Error: Parse error: missing ';' at '{'
AthameDepository.sol:1061:22: Error: Parse error: missing ';' at '{'
```

AthameToken.sol

```
AthameToken.sol:493:18: Error: Parse error: missing ';' at '{'
AthameToken.sol:526:18: Error: Parse error: missing ';' at '{'
AthameToken.sol:575:18: Error: Parse error: missing ';' at '{'
AthameToken.sol:626:22: Error: Parse error: missing ';' at '{'
```

AthameTreasury.sol

```
AthameTreasury.sol:600:18: Error: Parse error: missing ';' at '{'
AthameTreasury.sol:696:18: Error: Parse error: missing ';' at '{'
AthameTreasury.sol:719:18: Error: Parse error: missing ';' at '{'
AthameTreasury.sol:745:18: Error: Parse error: missing ';' at '{'
```

Software analysis result:

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

Email: audit@EtherAuthority.io