Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	IronVest Token
Platform:	Ferrum Network
Language:	Solidity
Date:	November 8th, 2022

Table of contents

Email: audit@EtherAuthority.io

`

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 IronVest Token team to perform the Security audit of the IronVest Token smart contract 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 November 8th, 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

IronVest Token Contract is smart contract, having functions like claim, claimable, initialize, addCliffVesting, addVesting, emergencyWithdraw, etc. The IronVest Token contract inherits IERC20Upgradeable, AccessControlUpgradeable, SafeERC20Upgradeable, Initializable, ReentrancyGuardUpgradeable 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 IronVest Token Smart Contract**
Platform	**Ferrum Network / Solidity**
File	IronVest.sol
File MD5 Hash	C5C39D2245B77CE0786F31649DD34016
Online code link	https://github.com/ferrumnet/linear-release-engine/blob/main/contracts/IronVest.sol
Audit Date	November 8th, 2022

Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Other Specifications ● Open Zeppelin standard code is used. ● Vester owners can create a new vesting with a cliff. ● Owner can withdraw emergency tokens that are sent to the contract mistakenly. ● Admin can set signer addresses.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity based smart contracts are **"Secured"**. This token contract does contain owner control, which does not make it 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 0 critical, 0 high, 0 medium and 1 low and some very low level issues.

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	Moderated
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
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Code Quality

This audit scope has 1 smart contract. Smart contract contains Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in the IronVest Token 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 IronVest Token.

The IronVest Token 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 used, which is a good thing.

Documentation

We were given an IronVest Token smart contract code in the form of a github weblink. The hash of that code is mentioned above in the table.

As mentioned above, code parts are well commented on. So it is 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.

Use of Dependencies

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

Apart from libraries, its functions are not used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	initializer	modifier	Passed	No Issue
3	reinitializer	modifier	Passed	No Issue
4	onlyInitializing	modifier	Passed	No Issue
5	_disableInitializers	internal	Passed	No Issue
6	__ReentrancyGuard_init	internal	access only Initializing	No Issue
7	__ReentrancyGuard_init_unchained	internal	access only Initializing	No Issue
8	nonReentrant	modifier	Passed	No Issue
9	_nonReentrantBefore	write	Passed	No Issue
10	_nonReentrantAfter	write	Passed	No Issue
11	_reentrancyGuardEntered	internal	Passed	No Issue
12	__AccessControl_init	internal	access only Initializing	No Issue
13	__AccessControl_init_unchained	internal	access only Initializing	No Issue
14	onlyRole	modifier	Passed	No Issue
15	supportsInterface	read	Passed	No Issue
16	hasRole	read	Passed	No Issue
17	_checkRole	internal	Passed	No Issue
18	_checkRole	internal	Passed	No Issue
19	getRoleAdmin	read	Passed	No Issue
20	grantRole	write	access only Role	No Issue
21	revokeRole	write	access only Role	No Issue
22	renounceRole	write	Passed	No Issue
23	_setupRole	internal	Passed	No Issue
24	_setRoleAdmin	internal	Passed	No Issue
25	_grantRole	internal	Passed	No Issue
26	_revokeRole	internal	Passed	No Issue
27	onlyVester	modifier	Passed	No Issue
28	onlyOwner	modifier	Passed	No Issue
29	initialize	internal	access only Initializing	No Issue
30	addVesting	external	Infinite loops possibility	Refer Audit Findings
31	claim	external	Passed	No Issue
32	addCliffVesting	external	Infinite loops possibility	Refer Audit Findings
33	claimCliff	external	Passed	No Issue
34	claimNonCliff	external	Passed	No Issue
35	emergencyWithdraw	external	access only Owner	No Issue
36	setSigner	external	access only Owner	No Issue

37	poolInformation	external	Passed	No Issue
38	claimable	read	Compile time warnings	Refer Audit Findings
39	cliffClaimable	read	Compile time warnings	Refer Audit Findings
40	nonCliffClaimable	read	Compile time warnings	Refer Audit Findings
41	signatureVerification	read	Passed	No Issue
42	_splitSignature	internal	Passed	No Issue
43	_verifyMessage	internal	Passed	No Issue
44	_messageHash	internal	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.

Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

No critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No medium severity vulnerabilities were found.

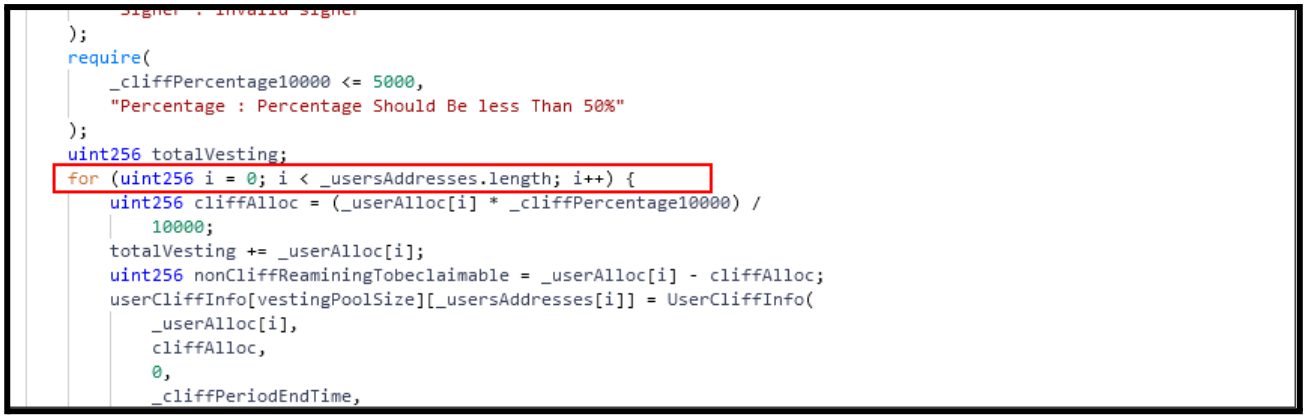
Low

(1) Infinite loops possibility:

addVesting()

```
/// @notice Create a new vesting.
function addVesting(
    string memory _poolName,
    uint256 _vestingEndTime,
    address _tokenAddress,
    address[] memory _usersAddresses,
    uint256[] memory _userAlloc,
    bytes memory _signature,
    bytes memory _keyHash
) external onlyVester nonReentrant {
    require(
        _usersAddresses.length == _userAlloc.length,
        "IIronVest Array : Length of _usersAddresses And _userAlloc Must Be Equal"
    );
    require(
        _vestingEndTime > block.timestamp,
        "IIronVest : Vesting End Time Should Be Greater Than Current Time"
    );
    require(
        signatureVerification(
            _signature,
            _poolName,
            _tokenAddress,
            _keyHash
        ) == signer,
        "Signer : Invalid signer"
    );
    uint256 totalVesting;
    for (uint256 i = 0; i < _usersAddresses.length; i++) {
        totalVesting += _userAlloc[i];
        userInfo[vestingPoolSize][_usersAddresses[i]] = UserInfo(
```

addCliffVesting()



```
        signer : invalid signer
    );
    require(
        _cliffPercentage10000 <= 5000,
        "Percentage : Percentage Should Be less Than 50%"
    );
    uint256 totalVesting;
    for (uint256 i = 0; i < _usersAddresses.length; i++) {
        uint256 cliffAlloc = (_userAlloc[i] * _cliffPercentage10000) /
            10000;
        totalVesting += _userAlloc[i];
        uint256 nonCliffReaminingTobeclaimable = _userAlloc[i] - cliffAlloc;
        userCliffInfo[vestingPoolSize][_usersAddresses[i]] = UserCliffInfo(
            _userAlloc[i],
            cliffAlloc,
            0,
            _cliffPeriodEndTime,
```

As array elements will increase, then 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 structure.

- addVesting() - _usersAddresses.length
- addCliffVesting() - _usersAddresses.length

Very Low / Informational / Best practices:

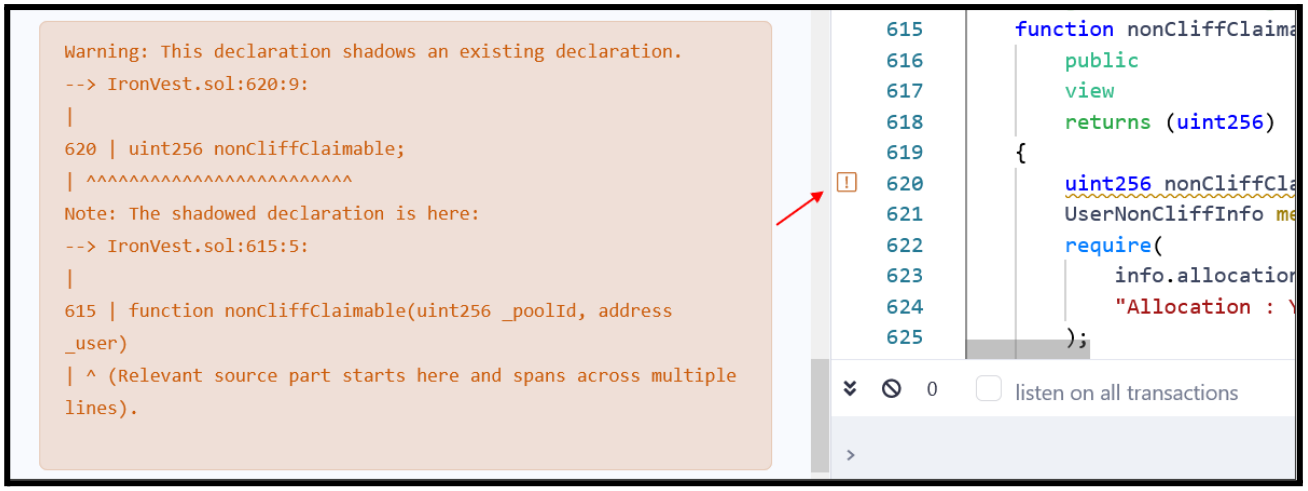
(1) Unlocked Compiler Version:

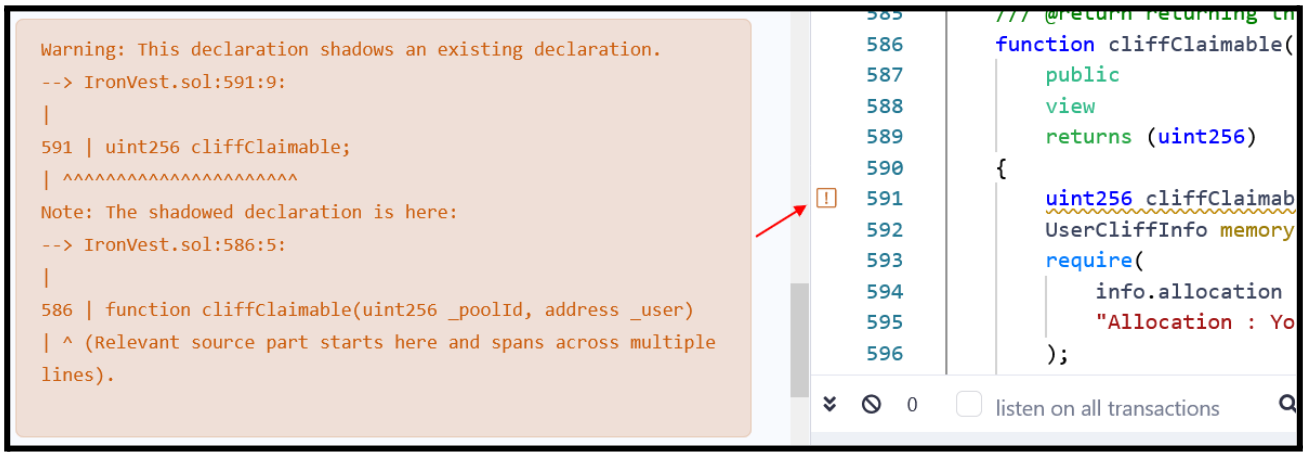
The contract uses the "^" prefix specifier, Use the Unlocked compiler version. Unlocked compiler version code of the smart contract, and that gives permission to the users to compile it one higher than a particular version.

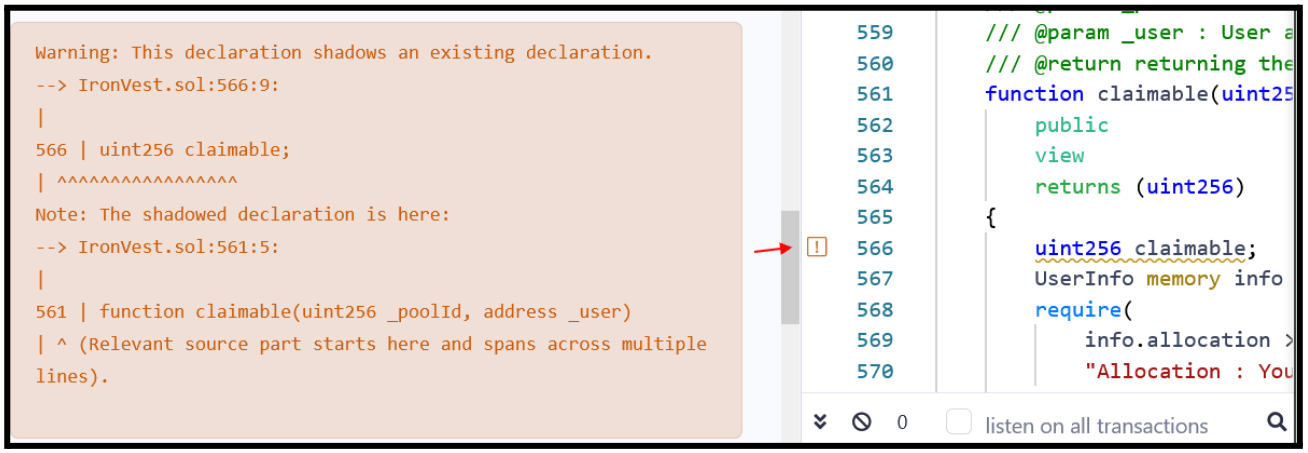
Resolution: We suggest using that the compiler version is unlocked instead of the locked compiler version. The following line of code can be added to the project:

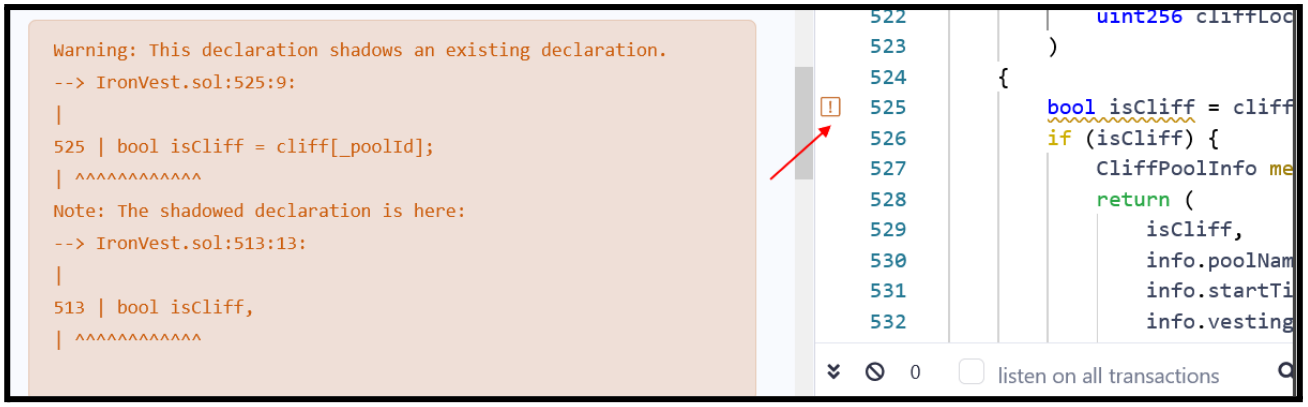
- pragma solidity 0.8.17;

(2) Compile time warnings:









Warning: This declaration shadows an existing declaration.

1)IronVest.sol:525:9: | 525 | bool isCliff = cliff[_poolId];

IronVest.sol:513:13: | 513 | bool isCliff, |

2.)IronVest.sol:566:9: | 566 | uint256 claimable; |

IronVest.sol:561:5: | 561 | function claimable(uint256 _poolId, address _user) |

3)IronVest.sol:591:9: | 591 | uint256 cliffClaimable;

IronVest.sol:586:5: | 586 | function cliffClaimable(uint256 _poolId, address _user) |

4)IronVest.sol:620:9: | 620 | uint256 nonCliffClaimable; |

IronVest.sol:615:5: | 615 | function nonCliffClaimable(uint256 _poolId, address _user) |

Resolution: We suggest if a variable is declared in the function then no need to declare it again in the same function. so we need to remove unwanted declarations.

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:

- addVesting: Vester can create a new vesting.
- addCliffVesting: Vester can create a new vesting with a cliff.
- emergencyWithdraw: Owner can use it to withdraw tokens that are sent to the contract mistakenly.
- setSigner: Owner can set a signer address.

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 github weblink. And we have used all possible tests based on given objects as files. We have observed 1 low severity issue and some Informational issues in the smart contract. But those are not critical ones. **So the smart contract is ready for 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.

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 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).

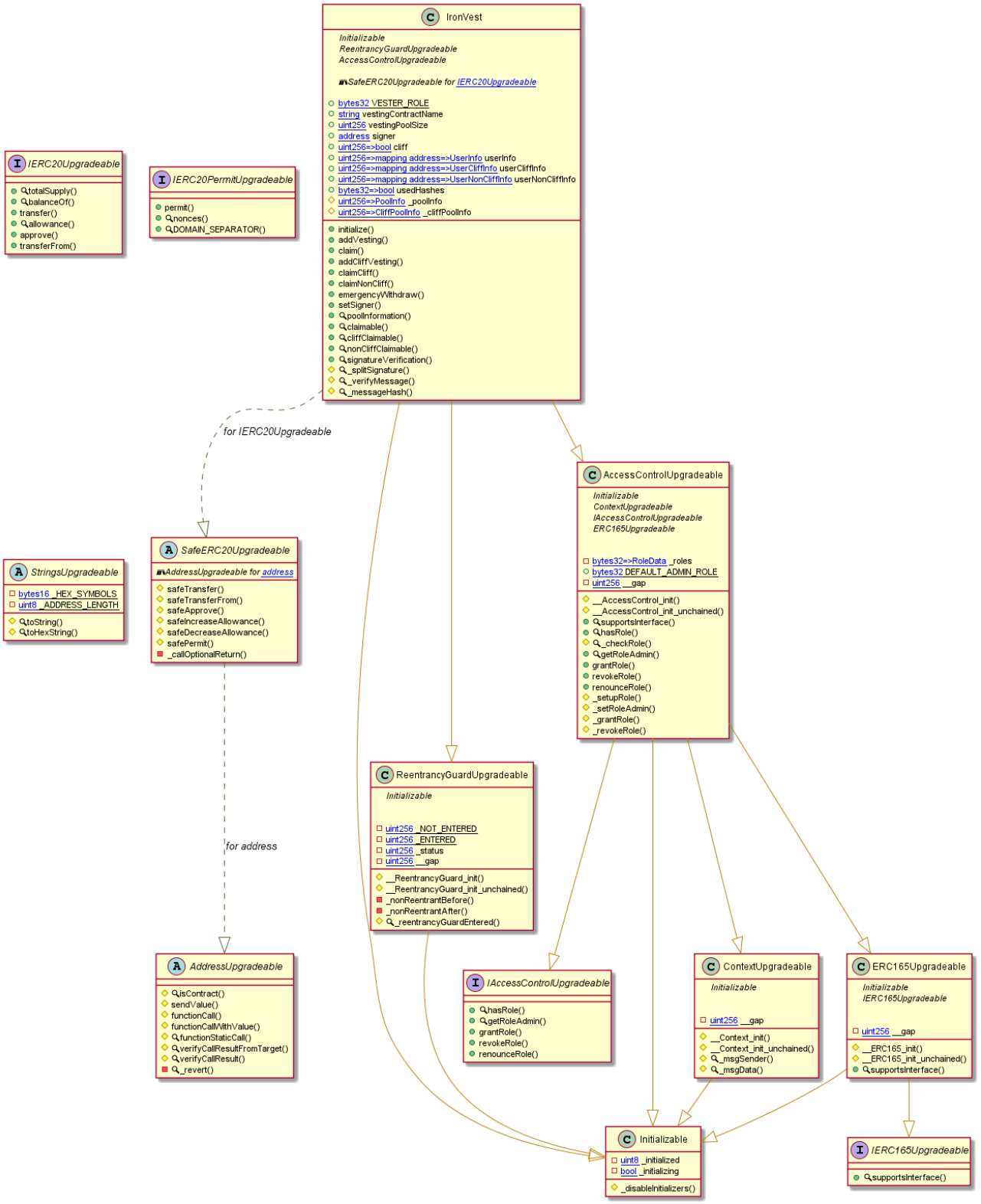
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we 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 - IronVest Token



Slither Results Log

Slither Log >> IronVest.sol

```
INFO:Detectors:
IronVest.claimable(uint256,address).claimable (IronVest.sol#1292) shadows:
        - IronVest.claimable(uint256,address) (IronVest.sol#1287-1306) (function)
IronVest.cliffClaimable(uint256,address).cliffClaimable (IronVest.sol#1317) shadows:
        - IronVest.cliffClaimable(uint256,address) (IronVest.sol#1312-1335) (function)
IronVest.nonCliffClaimable(uint256,address).nonCliffClaimable (IronVest.sol#1346) shadows:
        - IronVest.nonCliffClaimable(uint256,address) (IronVest.sol#1341-1362) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
IronVest.initialize(string,address)._signer (IronVest.sol#912) lacks a zero-check on :
                - signer = _signer (IronVest.sol#921)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in IronVest.addCliffVesting(string,uint256,uint256,uint256,address,uint256,address[],uint256[],bytes,bytes) (IronVes
t.sol#1032-1136):
        External calls:
        - IERC20Upgradeable(_tokenAddress).safeTransferFrom(_msgSender(),address(this),totalVesting) (IronVest.sol#1115-1119)
        State variables written after the call(s):
        - cliff[vestingPoolSize] = true (IronVest.sol#1120)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in IronVest.addCliffVesting(string,uint256,uint256,uint256,address,uint256,address[],uint256[],bytes,bytes) (IronVes
t.sol#1032-1136):
        External calls:
        - IERC20Upgradeable(_tokenAddress).safeTransferFrom(_msgSender(),address(this),totalVesting) (IronVest.sol#1115-1119)
        Event emitted after the call(s):
        - CliffAddVesting(_msgSender(),vestingPoolSize,_poolName,_vestingEndTime,_cliffVestingEndTime,nonCliffVestingPeriod,_cl
iffPeriodEndTime,_tokenAddress,totalVesting,_usersAddresses,_userAlloc) (IronVest.sol#1121-1133)
Reentrancy in IronVest.addVesting(string,uint256,address,address[],uint256[],bytes,bytes) (IronVest.sol#933-997):
        External calls:
        - IERC20Upgradeable(_tokenAddress).safeTransferFrom(_msgSender(),address(this),totalVesting) (IronVest.sol#979-983)
        Event emitted after the call(s):
        - AddVesting(_msgSender(),vestingPoolSize,_poolName,block.timestamp,_vestingEndTime,_tokenAddress,totalVesting,_usersAd
dresses,_userAlloc) (IronVest.sol#984-994)
Reentrancy in IronVest.claim(uint256) (IronVest.sol#1003-1018):
        External calls:
        - IERC20Upgradeable(_poolInfo[_poolId].tokenAddress).safeTransfer(_msgSender(),transferAble) (IronVest.sol#1006-1009)
```

```
        Event emitted after the call(s):
        - Claim(_poolId,transferAble,_msgSender(),remainingToBeClaimable) (IronVest.sol#1017)
Reentrancy in IronVest.claimCliff(uint256) (IronVest.sol#1143-1168):
        External calls:
        - IERC20Upgradeable(_cliffPoolInfo[_poolId].tokenAddress).safeTransfer(_msgSender(),transferAble) (IronVest.sol#1152-11
55)
        Event emitted after the call(s):
        - CliffClaim(_poolId,transferAble,_msgSender(),remainingTobeClaimable) (IronVest.sol#1162-1167)
Reentrancy in IronVest.claimNonCliff(uint256) (IronVest.sol#1175-1199):
        External calls:
        - IERC20Upgradeable(_cliffPoolInfo[_poolId].tokenAddress).safeTransfer(_msgSender(),transferAble) (IronVest.sol#1185-11
88)
        Event emitted after the call(s):
        - NonCliffClaim(_poolId,transferAble,_msgSender(),remainingTobeClaimable) (IronVest.sol#1193-1198)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
IronVest.addVesting(string,uint256,address,address[],uint256[],bytes,bytes) (IronVest.sol#933-997) uses timestamp for compariso
ns
        Dangerous comparisons:
        - require(bool,string)(_vestingEndTime > block.timestamp,IIronVest : Vesting End Time Should Be Greater Than Current Ti
me) (IronVest.sol#946-949)
IronVest.claim(uint256) (IronVest.sol#1003-1018) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(transferAble > 0,IIronVest : Invalid TransferAble) (IronVest.sol#1005)
IronVest.addCliffVesting(string,uint256,uint256,uint256,address,uint256,address[],uint256[],bytes,bytes) (IronVest.sol#1032-113
6) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(_cliffPeriodEndTime > block.timestamp,IIronVest : Cliff Vesting Time Must Be Lesser Than Vesting
 Time) (IronVest.sol#1056-1059)
IronVest.claimCliff(uint256) (IronVest.sol#1143-1168) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(_cliffPoolInfo[_poolId].cliffPeriodEndTime < block.timestamp,IIronVest : Cliff Period Is Not Ove
r Yet) (IronVest.sol#1145-1148)
        - require(bool,string)(transferAble > 0,IIronVest : Invalid TransferAble) (IronVest.sol#1151)
IronVest.claimNonCliff(uint256) (IronVest.sol#1175-1199) uses timestamp for comparisons
```

```
IronVest.claimNonCliff(uint256) (IronVest.sol#1175-1199) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(_cliffPoolInfo[_poolId].cliffPeriodEndTime < block.timestamp,IIronVest : Cliff Period Is Not Ove
r Yet) (IronVest.sol#1177-1180)
        - require(bool,string)(transferAble > 0,IIronVest : Invalid TransferAble) (IronVest.sol#1184)
IronVest.claimable(uint256,address) (IronVest.sol#1287-1306) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(info.allocation > 0,Allocation : You Don't have allocation in this pool) (IronVest.sol#1294-1297
)
        - _poolInfo[_poolId].vestingEndTime <= block.timestamp (IronVest.sol#1298)
IronVest.cliffClaimable(uint256,address) (IronVest.sol#1312-1335) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(info.allocation > 0,Allocation : You Don't have allocation in this pool) (IronVest.sol#1319-1322
)
        - _cliffPoolInfo[_poolId].cliffPeriodEndTime <= block.timestamp (IronVest.sol#1324)
        - _cliffPoolInfo[_poolId].cliffVestingEndTime >= block.timestamp (IronVest.sol#1326)
IronVest.nonCliffClaimable(uint256,address) (IronVest.sol#1341-1362) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(info.allocation > 0,Allocation : You Don't have allocation in this pool) (IronVest.sol#1348-1351
)
        - _cliffPoolInfo[_poolId].cliffPeriodEndTime <= block.timestamp (IronVest.sol#1353)
        - _cliffPoolInfo[_poolId].vestingEndTime >= block.timestamp (IronVest.sol#1354)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

```
INFO:Detectors:
AccessControlUpgradeable.__AccessControl_init_unchained() (IronVest.sol#536-537) is never used and should be removed
AccessControlUpgradeable._setRoleAdmin(bytes32,bytes32) (IronVest.sol#702-706) is never used and should be removed
AddressUpgradeable.functionCall(address,bytes) (IronVest.sol#40-42) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256) (IronVest.sol#52-58) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes) (IronVest.sol#71-73) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes,string) (IronVest.sol#75-82) is never used and should be removed
AddressUpgradeable.sendValue(address,uint256) (IronVest.sol#33-38) is never used and should be removed
AddressUpgradeable.verifyCallResult(bool,bytes,string) (IronVest.sol#100-110) is never used and should be removed
ContextUpgradeable.__Context_init() (IronVest.sol#427-428) is never used and should be removed
ContextUpgradeable.__Context_init_unchained() (IronVest.sol#430-431) is never used and should be removed
ContextUpgradeable._msgData() (IronVest.sol#436-438) is never used and should be removed
ERC165Upgradeable.__ERC165_init() (IronVest.sol#512-513) is never used and should be removed
ERC165Upgradeable.__ERC165_init_unchained() (IronVest.sol#515-516) is never used and should be removed
Initializable._disableInitializers() (IronVest.sol#417-423) is never used and should be removed
ReentrancyGuardUpgradeable._reentrancyGuardEntered() (IronVest.sol#499-501) is never used and should be removed
SafeERC20Upgradeable.safeApprove(IERC20Upgradeable,address,uint256) (IronVest.sol#159-169) is never used and should be removed
SafeERC20Upgradeable.safeDecreaseAllowance(IERC20Upgradeable,address,uint256) (IronVest.sol#180-191) is never used and should b
e removed
SafeERC20Upgradeable.safeIncreaseAllowance(IERC20Upgradeable,address,uint256) (IronVest.sol#171-178) is never used and should b
e removed
SafeERC20Upgradeable.safePermit(IERC20PermitUpgradeable,address,address,uint256,uint256,uint8,bytes32,bytes32) (IronVest.sol#19
3-207) is never used and should be removed
StringsUpgradeable.toHexString(uint256) (IronVest.sol#250-261) is never used and should be removed
StringsUpgradeable.toString(uint256) (IronVest.sol#225-245) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version0.8.4 (IronVest.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 AddressUpgradeable.sendValue(address,uint256) (IronVest.sol#33-38):
        - (success) = recipient.call{value: amount}() (IronVest.sol#36)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (IronVest.sol#60-69):
        - (success,returndata) = target.call{value: value}(data) (IronVest.sol#67)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (IronVest.sol#75-82):
        - (success,returndata) = target.staticcall(data) (IronVest.sol#80)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Function IERC20PermitUpgradeable.DOMAIN_SEPARATOR() (IronVest.sol#137) is not in mixedCase
Function ContextUpgradeable.__Context_init() (IronVest.sol#427-428) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (IronVest.sol#430-431) is not in mixedCase
Variable ContextUpgradeable.__gap (IronVest.sol#440) is not in mixedCase
Function ReentrancyGuardUpgradeable.__ReentrancyGuard_init() (IronVest.sol#460-462) is not in mixedCase
Function ReentrancyGuardUpgradeable.__ReentrancyGuard_init_unchained() (IronVest.sol#464-466) is not in mixedCase
Variable ReentrancyGuardUpgradeable.__gap (IronVest.sol#508) is not in mixedCase
Function ERC165Upgradeable.__ERC165_init() (IronVest.sol#512-513) is not in mixedCase
Function ERC165Upgradeable.__ERC165_init_unchained() (IronVest.sol#515-516) is not in mixedCase
Variable ERC165Upgradeable.__gap (IronVest.sol#529) is not in mixedCase
Function AccessControlUpgradeable.__AccessControl_init() (IronVest.sol#533-534) is not in mixedCase
Function AccessControlUpgradeable.__AccessControl_init_unchained() (IronVest.sol#536-537) is not in mixedCase
Variable AccessControlUpgradeable.__gap (IronVest.sol#741) is not in mixedCase
Parameter IronVest.initialize(string,address)._vestingName (IronVest.sol#912) is not in mixedCase
Parameter IronVest.initialize(string,address)._signer (IronVest.sol#912) is not in mixedCase
Parameter IronVest.addVesting(string,uint256,address,address[],uint256[],bytes,bytes)._poolName (IronVest.sol#934) is not in mi
xedCase
Parameter IronVest.addVesting(string,uint256,address,address[],uint256[],bytes,bytes)._vestingEndTime (IronVest.sol#935) is not
 in mixedCase
Parameter IronVest.addVesting(string,uint256,address,address[],uint256[],bytes,bytes)._tokenAddress (IronVest.sol#936) is not i
n mixedCase
Parameter IronVest.addVesting(string,uint256,address,address[],uint256[],bytes,bytes)._usersAddresses (IronVest.sol#937) is not
 in mixedCase
Parameter IronVest.addVesting(string,uint256,address,address[],uint256[],bytes,bytes)._userAlloc (IronVest.sol#938) is not in m
ixedCase
Parameter IronVest.addVesting(string,uint256,address,address[],uint256[],bytes,bytes)._signature (IronVest.sol#939) is not in m
ixedCase
Parameter IronVest.addVesting(string,uint256,address,address[],uint256[],bytes,bytes)._keyHash (IronVest.sol#940) is not in mix
edCase
Parameter IronVest.claim(uint256)._poolId (IronVest.sol#1003) is not in mixedCase
Parameter IronVest.addCliffVesting(string,uint256,uint256,uint256,address,uint256,address[],uint256[],bytes,bytes)._poolName (I
ronVest.sol#1033) is not in mixedCase
Parameter IronVest.addCliffVesting(string,uint256,uint256,uint256,address,uint256,address[],uint256[],bytes,bytes)._vestingEndT
ime (IronVest.sol#1034) is not in mixedCase
Parameter IronVest.addCliffVesting(string,uint256,uint256,uint256,address,uint256,address[],uint256[],bytes,bytes)._cliffVestin
gEndTime (IronVest.sol#1035) is not in mixedCase
Parameter IronVest.addCliffVesting(string,uint256,uint256,uint256,address,uint256,address[],uint256[],bytes,bytes)._cliffPeriod
EndTime (IronVest.sol#1036) is not in mixedCase
Parameter IronVest.addCliffVesting(string,uint256,uint256,uint256,address,uint256,address[],uint256[],bytes,bytes)._tokenAddres
s (IronVest.sol#1037) is not in mixedCase
Parameter IronVest.addCliffVesting(string,uint256,uint256,uint256,address,uint256,address[],uint256[],bytes,bytes)._cliffPercen
tage10000 (IronVest.sol#1038) is not in mixedCase
Parameter IronVest.addCliffVesting(string,uint256,uint256,uint256,address,uint256,address[],uint256[],bytes,bytes)._usersAddres
ses (IronVest.sol#1039) is not in mixedCase
Parameter IronVest.addCliffVesting(string,uint256,uint256,uint256,address,uint256,address[],uint256[],bytes,bytes)._userAlloc (
IronVest.sol#1040) is not in mixedCase
Parameter IronVest.addCliffVesting(string,uint256,uint256,uint256,address,uint256,address[],uint256[],bytes,bytes)._signature (
IronVest.sol#1041) is not in mixedCase
Parameter IronVest.addCliffVesting(string,uint256,uint256,uint256,address,uint256,address[],uint256[],bytes,bytes)._keyHash (Ir
onVest.sol#1042) is not in mixedCase
Parameter IronVest.claimCliff(uint256)._poolId (IronVest.sol#1143) is not in mixedCase
Parameter IronVest.claimNonCliff(uint256)._poolId (IronVest.sol#1175) is not in mixedCase
Parameter IronVest.emergencyWithdraw(IERC20Upgradeable,uint256)._token (IronVest.sol#1204) is not in mixedCase
Parameter IronVest.emergencyWithdraw(IERC20Upgradeable,uint256)._amount (IronVest.sol#1204) is not in mixedCase
Parameter IronVest.setSigner(address)._signer (IronVest.sol#1213) is not in mixedCase
Parameter IronVest.poolInformation(uint256)._poolId (IronVest.sol#1235) is not in mixedCase
Parameter IronVest.claimable(uint256,address)._poolId (IronVest.sol#1287) is not in mixedCase
Parameter IronVest.claimable(uint256,address)._user (IronVest.sol#1287) is not in mixedCase
Parameter IronVest.cliffClaimable(uint256,address)._poolId (IronVest.sol#1312) is not in mixedCase
Parameter IronVest.cliffClaimable(uint256,address)._user (IronVest.sol#1312) is not in mixedCase
Parameter IronVest.nonCliffClaimable(uint256,address)._poolId (IronVest.sol#1341) is not in mixedCase
Parameter IronVest.nonCliffClaimable(uint256,address)._user (IronVest.sol#1341) is not in mixedCase
Parameter IronVest.signatureVerification(bytes,string,address,bytes)._signature (IronVest.sol#1371) is not in mixedCase
Parameter IronVest.signatureVerification(bytes,string,address,bytes)._poolName (IronVest.sol#1372) is not in mixedCase
Parameter IronVest.signatureVerification(bytes,string,address,bytes)._tokenAddress (IronVest.sol#1373) is not in mixedCase
Parameter IronVest.signatureVerification(bytes,string,address,bytes)._keyHash (IronVest.sol#1374) is not in mixedCase
Variable IronVest._poolInfo (IronVest.sol#829) is not in mixedCase
Variable IronVest._cliffPoolInfo (IronVest.sol#831) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

```
INFO:Detectors:
AccessControlUpgradeable.__gap (IronVest.sol#741) is never used in IronVest (IronVest.sol#743-1458)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variables
INFO:Detectors:
grantRole(bytes32,address) should be declared external:
        - AccessControlUpgradeable.grantRole(bytes32,address) (IronVest.sol#632-634)
revokeRole(bytes32,address) should be declared external:
        - AccessControlUpgradeable.revokeRole(bytes32,address) (IronVest.sol#647-649)
renounceRole(bytes32,address) should be declared external:
        - AccessControlUpgradeable.renounceRole(bytes32,address) (IronVest.sol#667-671)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:IronVest.sol analyzed (13 contracts with 75 detectors), 109 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity Static Analysis

IronVest.sol

Security

Inline assembly:

The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.

more

Pos: 676:8:

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: 221:30:

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: 628:58:

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: 630:21:

Email: audit@EtherAuthority.io

Gas & Economy

Gas costs:

Gas requirement of function IronVest.vestingContractName 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: 85:4:

Gas costs:

Gas requirement of function IronVest.addVesting 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: 207:4:

Gas costs:

Gas requirement of function IronVest.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: 277:4:

Gas costs:

Gas requirement of function IronVest.addCliffVesting 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: 306:4:

Gas costs:

Gas requirement of function IronVest.signatureVerification 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: 644:4:

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: 234:8:

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.

more

Pos: 348:8:

Miscellaneous

Constant/View/Pure functions:

IronVest.emergencyWithdraw(contract IERC20Upgradeable,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 478:4:

Constant/View/Pure functions:

IronVest._splitSignature(bytes) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 665:4:

Similar variable names:

IronVest._verifyMessage(bytes32,uint8,bytes32,bytes32) : Variables have very similar names "_v" and "_r". Note: Modifiers are currently not considered by this static analysis.
Pos: 711:64:

Similar variable names:

IronVest._verifyMessage(bytes32,uint8,bytes32,bytes32) : Variables have very similar names "_v" and "_r". Note: Modifiers are currently not considered by this static analysis.
Pos: 711:68:

Similar variable names:

IronVest._verifyMessage(bytes32,uint8,bytes32,bytes32) : Variables have very similar names "_r" and "_s". Note: Modifiers are currently not considered by this static analysis.
Pos: 711:72:

No return:

IronVest._splitSignature(bytes): Defines a return type but never explicitly returns a value.
Pos: 665:4:

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: 674:8:

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: 241:16:

Solhint Linter

IronVest.sol

```
IronVest.sol:3:1: Error: Compiler version 0.8.17 does not satisfy the
r semver requirement
IronVest.sol:221:31: Error: Avoid to make time-based decisions in
your business logic
IronVest.sol:240:17: Error: Avoid to make time-based decisions in
your business logic
IronVest.sol:241:52: Error: Avoid to make time-based decisions in
your business logic
IronVest.sol:246:13: Error: Avoid to make time-based decisions in
your business logic
IronVest.sol:262:13: Error: Avoid to make time-based decisions in
your business logic
IronVest.sol:289:31: Error: Avoid to make time-based decisions in
your business logic
IronVest.sol:331:35: Error: Avoid to make time-based decisions in
your business logic
IronVest.sol:378:13: Error: Avoid to make time-based decisions in
your business logic
IronVest.sol:420:58: Error: Avoid to make time-based decisions in
your business logic
IronVest.sol:434:36: Error: Avoid to make time-based decisions in
your business logic
IronVest.sol:452:58: Error: Avoid to make time-based decisions in
your business logic
IronVest.sol:466:39: Error: Avoid to make time-based decisions in
your business logic
IronVest.sol:572:50: Error: Avoid to make time-based decisions in
your business logic
IronVest.sol:576:18: Error: Avoid to make time-based decisions in
your business logic
IronVest.sol:598:59: Error: Avoid to make time-based decisions in
your business logic
IronVest.sol:600:64: Error: Avoid to make time-based decisions in
your business logic
IronVest.sol:603:22: Error: Avoid to make time-based decisions in
your business logic
IronVest.sol:627:59: Error: Avoid to make time-based decisions in
your business logic
IronVest.sol:628:59: Error: Avoid to make time-based decisions in
your business logic
IronVest.sol:630:22: Error: Avoid to make time-based decisions in
your business logic
IronVest.sol:676:9: Error: Avoid using inline assembly. It is
acceptable only in rare cases
```

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

Email: audit@EtherAuthority.io