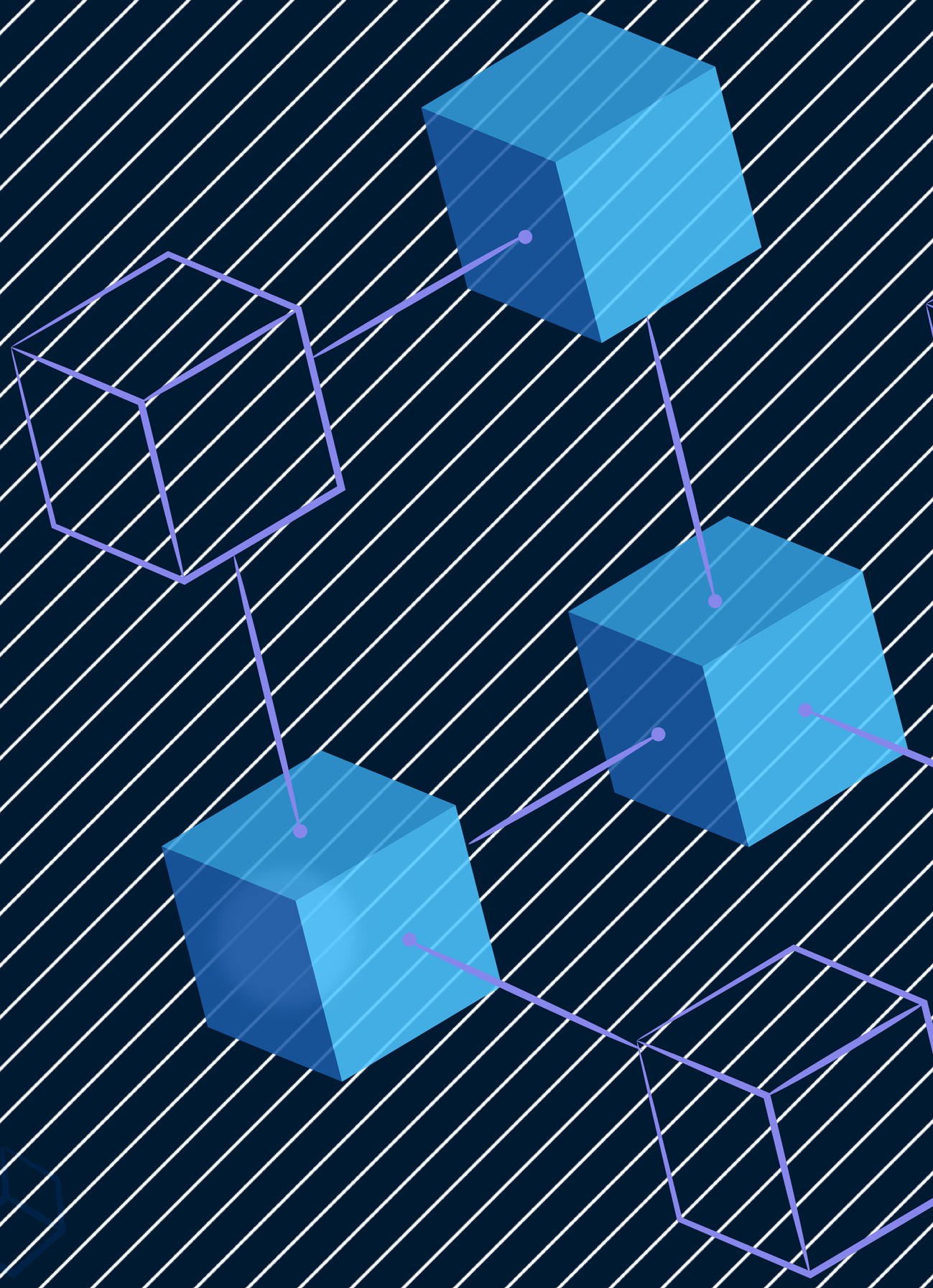
QuillAudits

Audit Report
January, 2022



For

Amplify

Contents

Contents

Scope of the Audit

The scope of this audit was to analyze and document the Amplify smart contracts codebase for quality, security, and correctness.

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis
In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis
Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis
Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption
In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis, Theo.

Issue Categories

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

Risk-level	Description
High	A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.
Medium	The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.
Low	Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.
Informational	These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	1	2	0
Closed	1	0	2	0

Introduction

During the period of **December 7, 2021, to December 15, 2021** - QuillAudits Team performed a security audit for **Amplify** smart contracts.

The code for the audit was taken from the following official repo of **Amplify**:

Note	Date	Commit hash
Version 1	December	21f17395ec22bd3758433d462f1cd024f95636f2
Version 2	December	39793a387584ef1191e453bbd67d7e98e8f132f6

Issues Found

A. Contract – Vesting

High severity issues

A.1 Usage Of transfer Instead Of safeTransfer:

```
Line 150:
    function withdraw(address destination) external onlyOwner
nonZeroAddress(destination) returns (bool) {
        return token.transfer(destination, this.balanceOf());
    }
```

Description

The ERC20 standard token implementation functions return the transaction status as a Boolean. It's good practice to check for the return status of the function call to ensure that the transaction was successful. It is the developer's responsibility to enclose these function calls with require() to ensure that, when the intended ERC20 function call returns false, the caller transaction also fails. However, it is mostly missed by developers when they carry out checks. In effect, the transaction would always succeed, even if the token transfer didn't.

Remediation

Use the safeTransfer function from the safeERC20 Implementation or put the transfer call inside an assert or require to verify that it returned true.

Status: Closed

Fixed

The Amplify team has fixed the issue by adding a require statements to make sure that the transaction passed successfully.

Medium severity issues

A.2 For Loop Over Dynamic Array:

```
Line 121:
    function createEntries(EntryVars[] calldata _entries) external onlyOwner nonReentrant
returns (bool){
        require(_entries.length  > 0, "empty data");
        require(_entries.length  <= 80, "exceed max length");

        for(uint8 i=0;i<_entries.length;i++){
            _createEntry(_entries[i]);
        }
        return true;
    }
```

```
Line 135:
    function claim() external nonReentrant  {
        uint256 totalAmount;
        for(uint8 i=0; i < entryIdsByRecipient[msg.sender].length; i++) {
            totalAmount += _claim(entryIdsByRecipient[msg.sender][i]);
        }
        if (totalAmount > 0) {
            emit Claimed(msg.sender, totalAmount);
            assert(token.transfer(msg.sender, totalAmount));
        }
    }
```

```
Line 177:
    function balanceOf(address account) external view returns (uint256 amount) {
        for(uint8 i=0; i < entryIdsByRecipient[account].length; i++) {
            amount += _balanceOf(entryIdsByRecipient[account][i]);
        }
        return amount;
    }
```

```
Line 202:
    function getSnapshot(address account) external nonZeroAddress(account) view
returns(Snapshot[] memory) {
        Snapshot[] memory snapshot = new Snapshot[]
(entryIdsByRecipient[account].length);

        for(uint8 i=0; i < entryIdsByRecipient[account].length; i++) {
            Entry memory entry = entries[entryIdsByRecipient[account][i]];
            snapshot[i] = Snapshot({
                entryId: entryIdsByRecipient[account][i],
                amount: entry.amount,
                start: entry.start,
                end: entry.end,
                cliff: entry.cliff,
                claimed: entry.claimed,
                available: _balanceOf(entryIdsByRecipient[account][i]),
                isFired: entry.isFired
            });
        }
        return snapshot;
    }
```

```
Line 235:
    function lockedOf(address account) external view returns (uint256 amount) {
        for(uint8 i=0; i < entryIdsByRecipient[account].length; i++) {
            amount += _lockedOf(entryIdsByRecipient[account][i]);
        }
        return amount;
    }
```

Description

When smart contracts are deployed or their associated functions are invoked, the execution of these operations always consumes a certain quantity of gas, according to the amount of computation required to accomplish them. Modifying an unknown-size array that grows in size over time can result in a Denial-of-Service attack. Simply by having an excessively huge array, users can exceed the gas limit, therefore preventing the transaction from ever succeeding.

Remediation

Avoid actions that involve looping across the entire data structure. If you really must loop over an array of unknown size, arrange for it to consume many blocks and thus multiple transactions.

Status: Acknowledged

Acknowledged

The Amplify team has acknowledged the issue.

Low severity issues

A.3 Missing address verification:

```
Line 67:
    function initialize(address owner_, IERC20 token_) external nonZeroAddress(owner_) {
        owner = owner_;
        token = token_;

        _entered = false;
    }
```

```
Line 80:
    function transferOwnership(address _newOwner) external onlyOwner
nonZeroAddress(_newOwner) {
        address currentOwner = owner;
        require(_newOwner != currentOwner, "New owner cannot be the current owner");
        owner = _newOwner;
        emit AdminChanged(currentOwner, _newOwner);
    }
```

Description

Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, the contract's functionality may become inaccessible or tokens may be burned in perpetuity.

Remediation

It’s recommended to undertake further validation prior to user-supplied data. The concerns can be resolved by utilizing a whitelist technique or a modifier.

Status: Closed

Fixed

The Amplify team has fixed the issue by verifying the addresses that are coming from the arguments using the nonZeroAddress modifier.

A.4 Usage Of block.timestamp:

```
Line 242:
    function getBlockTimestamp() public virtual view returns (uint256) {
        return block.timestamp;
    }
```

Description

Block.timestamp is used in the contract. The variable block is a set of variables. The timestamp does not always reflect the current time and may be inaccurate. The value of a block can be influenced by miners. Maximal Extractable Value attacks require a timestamp of up to 900 seconds. There is no guarantee that the value is right, all what is guaranteed is that it is higher than the timestamp of the previous block.

Remediation

You can use an Oracle to get the exact time or verify if a delay of 900 seconds won't destroy the logic of the staking contract.

Status: Acknowledged

Acknowledged

The Amplify team has acknowledged the issue.

B. Contract – VestingFactory

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

B.1 Missing address verification:

```
Line 55:
    function createVestingContract(IERC20 _token) external virtual {
        address _contract = createClone(libraryAddress);

        Vesting(_contract).initialize(msg.sender, _token);
        instances.push(Instance(_contract, msg.sender, address(_token)));

        emit InstanceCreated(_contract, msg.sender, address(_token));
    }
```

Description
Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, the contract's functionality may become inaccessible or tokens may be burned in perpetuity.

Remediation
It’s recommended to undertake further validation prior to user-supplied data. The concerns can be resolved by utilizing a whitelist technique or a modifier.

Status: Closed

Fixed
The Amplify team has fixed the issue by verifying the addresses that are coming from the arguments using the nonZeroAddress modifier.

B.2 Usage Of block.timestamp:

```
Line 242:
    function getBlockTimestamp() public virtual view returns (uint256) {
        return block.timestamp;
    }
```

Description

Block.timestamp is used in the contract. The variable block is a set of variables. The timestamp does not always reflect the current time and may be inaccurate. The value of a block can be influenced by miners. Maximal Extractable Value attacks require a timestamp of up to 900 seconds. There is no guarantee that the value is right, all what is guaranteed is that it is higher than the timestamp of the previous block.

Remediation

You can use an Oracle to get the exact time or verify if a delay of 900 seconds won't destroy the logic of the staking contract.

Status: Acknowledged

Acknowledged

The Amplify team has acknowledged the issue.

Functional Testing

constructor	PASS	PASS	PASS	Setting variable owner not needed as it is initialised in the Ownable.sol contract
setLibraryAddress	PASS	PASS	PASS	Missing event in critical state change
createVestingContract	PASS	PASS	PASS	
getBlockTimestamp	PASS	PASS	PASS	
Vesting.sol				

initialize	FAIL	FAIL	FAIL	Missing access controls leads to initialize function to be called again.
createEntry	PASS	PASS	PASS	return value of _createEntry ignored
createEntries	PASS	PASS	PASS	return value of _createEntry ignored
_createEntry				If entry.amount and entry.unlocked are equal only token transfer takes place and no entry is created Comments says that pre-approval is needed to bring tokens into vesting contract but direct token transfers are made from vesting contract
fireEntry	PASS	PASS	PASS	
_fireEntry	PASS	PASS	PASS	
withdraw	PASS	PASS	PASS	
balanceOf	PASS	PASS	PASS	
balanceOf(uint256)	PASS	PASS	PASS	
balanceOf(address)	PASS	PASS	PASS	
_balanceOf(uint256)	PASS	PASS	PASS	can be internal
lockedOf(uint256)	PASS	PASS	PASS	
lockedOf(address)	PASS	PASS	PASS	
_lockedOf(uint256)	PASS	PASS	PASS	can be internal
getBlockTimestamp	PASS	PASS	PASS	
withdraw	PASS	PASS	PASS	
claim	PASS	PASS	PASS	
_claim	PASS	PASS	PASS	

Automated Tests

Slither

```
Reentrancy in Vesting._createEntry(Vesting.EntryVars) (Vesting.sol#206-242):
    External calls:
    - assert(bool)(token.transfer(recipient,entry.unlocked)) (Vesting.sol#219)
    State variables written after the call(s):
    - entries[currentEntryId] = Entry(recipient,entry.amount -
    entry.unlocked,entry.start,entry.start,entry.end,entry.cliff,0,entry.isFireable,false)
    (Vesting.sol#224-234)
     - entryIdsByRecipient[recipient].push(currentEntryId) (Vesting.sol#235)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
```

```
Reentrancy in Vesting._createEntry(Vesting.EntryVars) (Vesting.sol#206-242):
    External calls:
    - assert(bool)(token.transfer(recipient,entry.unlocked)) (Vesting.sol#219)
    Event emitted after the call(s):
    - EntryCreated(currentEntryId,recipient,entry.amount -
    entry.unlocked,entry.start,entry.end,entry.cliff) (Vesting.sol#237)

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

```
Vesting._createEntry(Vesting.EntryVars) (Vesting.sol#206-242) uses timestamp for comparisons
    Dangerous comparisons:
    - require(bool,string)(entry.start >= currentTimestamp,Start time must be in the future)
    (Vesting.sol#216)
Vesting.claim() (Vesting.sol#248-257) uses timestamp for comparisons
    Dangerous comparisons:
    - totalAmount > 0 (Vesting.sol#253)
    - assert(bool)(token.transfer(msg.sender,totalAmount)) (Vesting.sol#255)
Vesting._claim(uint256) (Vesting.sol#259-274) uses timestamp for comparisons
    Dangerous comparisons:
    - amountToClaim > 0 (Vesting.sol#263)
    - require(bool,string)(entry.amount >= entry.claimed,claim exceed vested amount)
    (Vesting.sol#270)
Vesting._fireEntry(uint256) (Vesting.sol#285-295) uses timestamp for comparisons
    Dangerous comparisons:
    - require(bool,string)(entry.amount > 0,entry not exists) (Vesting.sol#287)
    - require(bool,string)(entry.isFireable,entry not fireable) (Vesting.sol#288)
Vesting._balanceOf(uint256) (Vesting.sol#339-356) uses timestamp for comparisons
    Dangerous comparisons:
    - currentTimestamp <= entry.start + entry.cliff (Vesting.sol#347)
    - currentTimestamp > entry.end || entry.isFired (Vesting.sol#351)

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

Pragma version0.8.4 (Vesting.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

Parameter Ownable.transferOwnership(address)._newOwner (Vesting.sol#112) is not in mixedCase
Parameter Vesting.createEntries(Vesting.EntryVars[])._entries (Vesting.sol#196) is not in mixedCase
Function Vesting._balanceOf(uint256) (Vesting.sol#339-356) is not in mixedCase
Parameter Vesting._balanceOf(uint256)._entryId (Vesting.sol#339) is not in mixedCase
Function Vesting._lockedOf(uint256) (Vesting.sol#379-385) is not in mixedCase
Parameter Vesting._lockedOf(uint256)._entryId (Vesting.sol#379) is not in mixedCase

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Reentrancy in Vesting._createEntry(Vesting.EntryVars) (VestingFactory.sol#206-242):
External calls:
- assert(bool)(token.transfer(recipient,entry.unlocked)) (VestingFactory.sol#219)
State variables written after the call(s):
- entries[currentEntryId] = Entry(recipient,entry.amount - entry.unlocked,entry.start,entry.start,entry.end,entry.cliff,0,entry.isFireable,false) (VestingFactory.sol#224-234)
- entryIdsByRecipient[recipient].push(currentEntryId) (VestingFactory.sol#235)
Reentrancy in VestingFactory.createVestingContract(IERC20) (VestingFactory.sol#505-512):
External calls:
- Vesting(_contract).initialize(msg.sender,_token) (VestingFactory.sol#508)
State variables written after the call(s):
- instances.push(Instance(_contract,msg.sender,address(_token))) (VestingFactory.sol#509)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in Vesting._createEntry(Vesting.EntryVars) (VestingFactory.sol#206-242):
External calls:
- assert(bool)(token.transfer(recipient,entry.unlocked)) (VestingFactory.sol#219)
Event emitted after the call(s):
- EntryCreated(currentEntryId,recipient,entry.amount - entry.unlocked,entry.start,entry.end,entry.cliff) (VestingFactory.sol#237)
Reentrancy in VestingFactory.createVestingContract(IERC20) (VestingFactory.sol#505-512):
External calls:
- Vesting(_contract).initialize(msg.sender,_token) (VestingFactory.sol#508)
Event emitted after the call(s):
- InstanceCreated(_contract,msg.sender,address(_token)) (VestingFactory.sol#511)

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

Vesting._createEntry(Vesting.EntryVars) (VestingFactory.sol#206-242) uses timestamp for comparisons
Dangerous comparisons:
- require(bool,string)(entry.start >= currentTimestamp,Start time must be in the future) (VestingFactory.sol#216)
Vesting.claim() (VestingFactory.sol#248-257) uses timestamp for comparisons
Dangerous comparisons:
- totalAmount > 0 (VestingFactory.sol#253)
- assert(bool)(token.transfer(msg.sender,totalAmount)) (VestingFactory.sol#255)
Vesting._claim(uint256) (VestingFactory.sol#259-274) uses timestamp for comparisons
Dangerous comparisons:
- amountToClaim > 0 (VestingFactory.sol#263)
- require(bool,string)(entry.amount >= entry.claimed,claim exceed vested amount) (VestingFactory.sol#270)
Vesting._fireEntry(uint256) (VestingFactory.sol#285-295) uses timestamp for comparisons
Dangerous comparisons:
- require(bool,string)(entry.amount > 0,entry not exists) (VestingFactory.sol#287)
- require(bool,string)(entry.isFireable,entry not fireable) (VestingFactory.sol#288)
Vesting._balanceOf(uint256) (VestingFactory.sol#339-356) uses timestamp for comparisons
Dangerous comparisons:
- currentTimestamp <= entry.start + entry.cliff (VestingFactory.sol#347)
- currentTimestamp > entry.end || entry.isFired (VestingFactory.sol#351)

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

Clones.createClone(address) (VestingFactory.sol#430-442) uses assembly
- INLINE ASM (VestingFactory.sol#433-439)
Clones.isClone(address,address) (VestingFactory.sol#444-459) uses assembly
- INLINE ASM (VestingFactory.sol#446-458)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Clones.isClone(address,address) (VestingFactory.sol#444-459) is never used and should be removed

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version0.8.4 (VestingFactory.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

Parameter Ownable.transferOwnership(address)._newOwner (VestingFactory.sol#112) is not in mixedCase
Parameter Vesting.createEntries(Vesting.EntryVars[])._entries (VestingFactory.sol#196) is not in mixedCase
Function Vesting._balanceOf(uint256) (VestingFactory.sol#339-356) is not in mixedCase
Parameter Vesting._balanceOf(uint256)._entryId (VestingFactory.sol#339) is not in mixedCase
Function Vesting._lockedOf(uint256) (VestingFactory.sol#379-385) is not in mixedCase
Parameter Vesting._lockedOf(uint256)._entryId (VestingFactory.sol#379) is not in mixedCase
Parameter VestingFactory.setLibraryAddress(address)._libraryAddress (VestingFactory.sol#494) is not in mixedCase
Parameter VestingFactory.createVestingContract(IERC20)._token (VestingFactory.sol#505) is not in mixedCase

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Clones.createClone(address) (VestingFactory.sol#430-442) uses literals with too many digits:
- mstore(uint256,uint256)(clone_createClone_asm_0,0x3d602d80600a3d3981f3363d3d373d3d3d363d730000000000000000000000000000000) (VestingFactory.sol#435)
Clones.createClone(address) (VestingFactory.sol#430-442) uses literals with too many digits:
- mstore(uint256,uint256)(clone_createClone_asm_0 + 0x28,0x5af43d82803e903d91602b57fd5bf3000000000000000000000000000000000000) (VestingFactory.sol#437)
Clones.isClone(address,address) (VestingFactory.sol#444-459) uses literals with too many digits:
- mstore(uint256,uint256)(clone_isClone_asm_0,0x363d3d373d3d3d363d730000000000000000000000000000000000000000) (VestingFactory.sol#448)
Clones.isClone(address,address) (VestingFactory.sol#444-459) uses literals with too many digits:
- mstore(uint256,uint256)(clone_isClone_asm_0 + 0x1e,0x5af43d82803e903d91602b57fd5bf3000000000000000000000000000000000000) (VestingFactory.sol#450)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

getBlockTimestamp() should be declared external:
- VestingFactory.getBlockTimestamp() (VestingFactory.sol#514-516)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external

Reentrancy in Vesting._createEntry(Vesting.EntryVars) (Vesting.sol#206-242):
External calls:
- assert(bool)(token.transfer(recipient,entry.unlocked)) (Vesting.sol#219)
State variables written after the call(s):
- entries[currentEntryId] = Entry(recipient,entry.amount - entry.unlocked,entry.start,entry.start,entry.end,entry.cliff,0,entry.isFireable,false) (Vesting.sol#224-234)
- entryIdsByRecipient[recipient].push(currentEntryId) (Vesting.sol#235)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in Vesting._createEntry(Vesting.EntryVars) (Vesting.sol#206-242):
External calls:
- assert(bool)(token.transfer(recipient,entry.unlocked)) (Vesting.sol#219)
Event emitted after the call(s):
- EntryCreated(currentEntryId,recipient,entry.amount - entry.unlocked,entry.start,entry.end,entry.cliff) (Vesting.sol#237)

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

Vesting._createEntry(Vesting.EntryVars) (Vesting.sol#206-242) uses timestamp for comparisons
Dangerous comparisons:
- require(bool,string)(entry.start >= currentTimestamp,Start time must be in the future) (Vesting.sol#216)
Vesting.claim() (Vesting.sol#248-257) uses timestamp for comparisons
Dangerous comparisons:
- totalAmount > 0 (Vesting.sol#253)
- assert(bool)(token.transfer(msg.sender,totalAmount)) (Vesting.sol#255)
Vesting._claim(uint256) (Vesting.sol#259-274) uses timestamp for comparisons
Dangerous comparisons:
- amountToClaim > 0 (Vesting.sol#263)
- require(bool,string)(entry.amount >= entry.claimed,claim exceed vested amount) (Vesting.sol#270)
Vesting._fireEntry(uint256) (Vesting.sol#285-295) uses timestamp for comparisons
Dangerous comparisons:
- require(bool,string)(entry.amount > 0,entry not exists) (Vesting.sol#287)
- require(bool,string)(entry.isFireable,entry not fireable) (Vesting.sol#288)
Vesting._balanceOf(uint256) (Vesting.sol#339-356) uses timestamp for comparisons
Dangerous comparisons:
- currentTimestamp <= entry.start + entry.cliff (Vesting.sol#347)
- currentTimestamp > entry.end || entry.isFired (Vesting.sol#351)

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

Pragma version0.8.4 (Vesting.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

Parameter Ownable.transferOwnership(address)._newOwner (Vesting.sol#112) is not in mixedCase
Parameter Vesting.createEntries(Vesting.EntryVars[])._entries (Vesting.sol#196) is not in mixedCase
Function Vesting._balanceOf(uint256) (Vesting.sol#339-356) is not in mixedCase
Parameter Vesting._balanceOf(uint256)._entryId (Vesting.sol#339) is not in mixedCase
Function Vesting._lockedOf(uint256) (Vesting.sol#379-385) is not in mixedCase
Parameter Vesting._lockedOf(uint256)._entryId (Vesting.sol#379) is not in mixedCase

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Vesting.sol analyzed (7 contracts with 75 detectors), 15 result(s) found

Reentrancy in Vesting._createEntry(Vesting.EntryVars) (VestingFactory.sol#206-242):
External calls:
- assert(bool)(token.transfer(recipient,entry.unlocked)) (VestingFactory.sol#219)
State variables written after the call(s):
- entries[currentEntryId] = Entry(recipient,entry.amount - entry.unlocked,entry.start,entry.start,entry.end,entry.cliff,0,entry.isFireable,false) (VestingFactory.sol#224-234)
- entryIdsByRecipient[recipient].push(currentEntryId) (VestingFactory.sol#235)
Reentrancy in VestingFactory.createVestingContract(IERC20) (VestingFactory.sol#505-512):
External calls:
- Vesting(_contract).initialize(msg.sender,_token) (VestingFactory.sol#508)
State variables written after the call(s):
- instances.push(Instance(_contract,msg.sender,address(_token))) (VestingFactory.sol#509)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in Vesting._createEntry(Vesting.EntryVars) (VestingFactory.sol#206-242):
External calls:
- assert(bool)(token.transfer(recipient,entry.unlocked)) (VestingFactory.sol#219)
Event emitted after the call(s):
- EntryCreated(currentEntryId,recipient,entry.amount - entry.unlocked,entry.start,entry.end,entry.cliff) (VestingFactory.sol#237)
Reentrancy in VestingFactory.createVestingContract(IERC20) (VestingFactory.sol#505-512):
External calls:
- Vesting(_contract).initialize(msg.sender,_token) (VestingFactory.sol#508)
Event emitted after the call(s):
- InstanceCreated(_contract,msg.sender,address(_token)) (VestingFactory.sol#511)

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

Vesting._createEntry(Vesting.EntryVars) (VestingFactory.sol#206-242) uses timestamp for comparisons
Dangerous comparisons:
- require(bool,string)(entry.start >= currentTimestamp,Start time must be in the future) (VestingFactory.sol#216)
Vesting.claim() (VestingFactory.sol#248-257) uses timestamp for comparisons
Dangerous comparisons:
- totalAmount > 0 (VestingFactory.sol#253)
- assert(bool)(token.transfer(msg.sender,totalAmount)) (VestingFactory.sol#255)
Vesting._claim(uint256) (VestingFactory.sol#259-274) uses timestamp for comparisons
Dangerous comparisons:
- amountToClaim > 0 (VestingFactory.sol#263)
- require(bool,string)(entry.amount >= entry.claimed,claim exceed vested amount) (VestingFactory.sol#270)
Vesting._fireEntry(uint256) (VestingFactory.sol#285-295) uses timestamp for comparisons
Dangerous comparisons:
- require(bool,string)(entry.amount > 0,entry not exists) (VestingFactory.sol#287)
- require(bool,string)(entry.isFireable,entry not fireable) (VestingFactory.sol#288)
Vesting._balanceOf(uint256) (VestingFactory.sol#339-356) uses timestamp for comparisons
Dangerous comparisons:
- currentTimestamp <= entry.start + entry.cliff (VestingFactory.sol#347)
- currentTimestamp > entry.end || entry.isFired (VestingFactory.sol#351)

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

Clones.createClone(address) (VestingFactory.sol#430-442) uses assembly
- INLINE ASM (VestingFactory.sol#433-439)
Clones.isClone(address,address) (VestingFactory.sol#444-459) uses assembly
- INLINE ASM (VestingFactory.sol#446-458)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Clones.isClone(address,address) (VestingFactory.sol#444-459) is never used and should be removed

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version0.8.4 (VestingFactory.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

Parameter Ownable.transferOwnership(address)._newOwner (VestingFactory.sol#112) is not in mixedCase
Parameter Vesting.createEntries(Vesting.EntryVars[])._entries (VestingFactory.sol#196) is not in mixedCase
Function Vesting._balanceOf(uint256) (VestingFactory.sol#339-356) is not in mixedCase
Parameter Vesting._balanceOf(uint256)._entryId (VestingFactory.sol#339) is not in mixedCase
Function Vesting._lockedOf(uint256) (VestingFactory.sol#379-385) is not in mixedCase
Parameter Vesting._lockedOf(uint256)._entryId (VestingFactory.sol#379) is not in mixedCase
Parameter VestingFactory.setLibraryAddress(address)._libraryAddress (VestingFactory.sol#494) is not in mixedCase
Parameter VestingFactory.createVestingContract(IERC20)._token (VestingFactory.sol#505) is not in mixedCase

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Clones.createClone(address) (VestingFactory.sol#430-442) uses literals with too many digits:
- mstore(uint256,uint256)(clone_createClone_asm_0,0x3d602d80600a3d3981f3363d3d373d3d3d363d730000000000000000000000000000000000) (VestingFactory.sol#435)
Clones.createClone(address) (VestingFactory.sol#430-442) uses literals with too many digits:
- mstore(uint256,uint256)(clone_createClone_asm_0 + 0x28,0x5af43d82803e903d91602b57fd5bf3000000000000000000000000000000000000) (VestingFactory.sol#437)
Clones.isClone(address,address) (VestingFactory.sol#444-459) uses literals with too many digits:
- mstore(uint256,uint256)(clone_isClone_asm_0,0x363d3d373d3d3d363d730000000000000000000000000000000000000000) (VestingFactory.sol#448)
Clones.isClone(address,address) (VestingFactory.sol#444-459) uses literals with too many digits:
- mstore(uint256,uint256)(clone_isClone_asm_0 + 0x1e,0x5af43d82803e903d91602b57fd5bf300000000000000000000000000000000000000) (VestingFactory.sol#450)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

getBlockTimestamp() should be declared external:
- VestingFactory.getBlockTimestamp() (VestingFactory.sol#514-516)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external

VestingFactory.sol analyzed (9 contracts with 75 detectors), 27 result(s) found

Results
No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

Closing Summary

Overall, smart contracts are very well written and adhere to guidelines. Many issues were discovered during the initial audit; the majority of them are fixed.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the Amplify Contracts. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Amplify Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report
January, 2022

For

Amplify

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com