0x Guard

Smart contracts security assessment

Final report

Tariff: Standard

Fibon

March 2025

0xguard.com

hello@0xguard.com

Contents

Introduction

The report has been prepared for **Fibon**.

The audited project consists of ERC-20 token, multisig contract, ICO sale, and vesting contract. The FibonToken can be minted by the owner, has a blacklist for transfers, and has a flat rate transfer fee.

The code is available at the GitHub repository and was audited after the commit 09521af3d298974266987d0c61198381ae484e86.

Report update. The contracts were updated according to this report and rechecked after the commit 55251ec66653820d1d5d7485df7af2970a3743a2.

Report update #2. The contracts were updated according to this report and rechecked after the commit a91497ad44c8d654f5566f72c07d7bd22935e20e.

Name	Fibon
Audit date	2024-09-23 - 2025-03-13
Language	Solidity
Platform	Polygon Network

Contracts checked

Name	Address
FibonToken	
FibonMultiSig	
FibonICO	
FibonVesting	
All project contracts	

Procedure

We perform our audit according to the following procedure:

Automated analysis

- Scanning the project's smart contracts with several publicly available automated Solidity analysis tools
- Manual verification (reject or confirm) all the issues found by the tools

Manual audit

- Manually analyze smart contracts for security vulnerabilities
- Smart contracts' logic check

Known vulnerabilities checked

Title	Check result
Unencrypted Private Data On-Chain	passed
Code With No Effects	passed
Message call with hardcoded gas amount	passed
Typographical Error	passed
DoS With Block Gas Limit	passed
Presence of unused variables	passed
Incorrect Inheritance Order	passed
Requirement Violation	passed
Weak Sources of Randomness from Chain Attributes	passed
Shadowing State Variables	passed

Incorrect Constructor Name	passed
Block values as a proxy for time	passed
Authorization through tx.origin	passed
DoS with Failed Call	passed
Delegatecall to Untrusted Callee	passed
Use of Deprecated Solidity Functions	passed
Assert Violation	passed
State Variable Default Visibility	passed
Reentrancy	passed
Unprotected SELFDESTRUCT Instruction	passed
Unprotected Ether Withdrawal	passed
Unchecked Call Return Value	passed
Floating Pragma	passed
Outdated Compiler Version	passed
Integer Overflow and Underflow	passed
Function Default Visibility	passed

Classification of issue severity

High severity	High severity issues can cause a significant or full loss of funds, change of contract ownership, major interference with contract logic. Such issues require immediate attention.
Medium severity	Medium severity issues do not pose an immediate risk, but can be detrimental to the client's reputation if exploited. Medium severity issues may lead to a contract failure and can be fixed by modifying the contract state or redeployment. Such issues require attention.

Low severity Low severity issues do not cause significant destruction to the contract's functionality. Such issues are recommended to be taken into consideration.

Issues

High severity issues

1. Privileged functions (FibonToken)
Status: Partially fixed

The contract owner can mint an arbitrary number of tokens.

The owner can blacklist address from participation in transfers.

The owner can update transfer fee or disable it.

```
/**
 * @notice Allows the owner to mint new tokens.
 * @dev Only the contract owner can call this function.
 * @param to The address to receive the minted tokens.
 * @param amount The amount of tokens to mint.
 */
function mint(address to, uint256 amount) public onlyOwner {
    require(!isBlacklisted[to], "Recipient is blacklisted");
    _mint(to, amount);
}

/**
 * @notice Sets the fixed transfer fee
 * @param newFee The new fee amount in token units
 */
function setTransferFee(uint256 newFee) external onlyOwner {
    uint256 oldFee = transferFee;
    transferFee = newFee;
    emit TransferFeeUpdated(oldFee, newFee);
}
```

```
    /**
     * @notice Allows the owner to blacklist an address
     * @param _account The address to blacklist
     */
    function blacklistAddress(address _account) external onlyOwner {
        require(_account != address(0), "Invalid address");
        require(!isBlacklisted[_account], "Address already blacklisted");

        isBlacklisted[_account] = true;
        emit AddressBlacklisted(_account);
    }
```

Recommendation: Secure owner's account, limit the total supply with a hard cap, limit max transfer fee.

Update: The total supply has been capped to 5.882e27.

The maximum transfer fee has been limited to 10%.

The owner's blacklisting ability remains.

We recommend securing owner's account by using a smart account with distributed access.

2. Owner can change token address (FibonVesting)

Status: Fixed

The contract owner can update address of token contract, meaning that beneficiaries may unlock different tokens than expected.

```
    function setToken(IERC20 _newToken) external onlyOwner {
        token = _newToken;
    }
```

Recommendation: Remove the `setToken` function.

3. Owner can drain vested tokens (FibonVesting)

Status: Open

The contract owner can transfer out all vested tokens by adding a beneficiary with immediate release schedule.

```
    /**
     * @notice Allows the owner to add a new vesting type.
     * @param typeId The ID of the vesting type.
     * @param phases The array of VestingPhase structs.
     */
    function addVestingType(
        uint8 typeId,
        VestingPhase[] memory phases
    ) external onlyOwner {
        require(vestingTypes[typeId].length == 0, "Vesting type already exists");

        _validatePhases(phases);

        uint256 totalPercentage;
        for (uint256 i = 0; i < phases.length; i++) {
            require(phases[i].end >= phases[i].start, "Phase end must be after or equal
to start");
            totalPercentage += phases[i].percentage;
        }
        require(totalPercentage == 100, "Total percentages must equal 100");

        for (uint256 i = 0; i < phases.length; i++) {
            vestingTypes[typeId].push(phases[i]);
        }

        emit VestingTypeAdded(typeId);
    }

    /**
     * @notice Creates a new vesting schedule for a beneficiary using a predefined
vesting type.
     * @dev Only the owner can call this function.
     * @param _beneficiary The address of the beneficiary.
     * @param typeId The ID of the predefined vesting type.
     * @param _amount The total amount of tokens allocated to the beneficiary.
     */
    function createVestingSchedule(
        address _beneficiary,
        uint8 typeId,
        uint256 _amount
    ) external onlyOwner {
        require(_beneficiary != address(0), "Invalid beneficiary address");
        require(_amount > 0, "Amount must be greater than 0");
        require(vestingSchedules[_beneficiary].startTime == 0, "Vesting schedule already
exists");

        VestingPhase[] storage vtPhases = vestingTypes[typeId];
        require(vtPhases.length > 0, "Invalid vesting type");

        require(token.balanceOf(address(this)) >= _amount, "Insufficient contract
balance");

        vestingSchedules[_beneficiary].startTime = block.timestamp;
        vestingSchedules[_beneficiary].releasedAmount = 0;
        vestingSchedules[_beneficiary].isDisabled = false;

        for (uint256 i = 0; i < vtPhases.length; i++) {
            vestingSchedules[_beneficiary].phases.push(vtPhases[i]);
        }

        totalAllocation[_beneficiary] = _amount;
        totalAllocated += _amount;

        emit VestingScheduleCreated(_beneficiary, _amount);
    }
```

Recommendation: Adding a beneficiary could be done with incoming transfer of tokens to be vested.

We recommend transferring contract's ownership to multisig smart account to mitigate risk.

Update: The contract owner can transfer out vested tokens by calling the `disableVestingSchedule` function that transfers all not ready to be released tokens to the owner.

4. Locked tokens (FibonVesting)

Status: Fixed

The owner can lock all unclaimed user's tokens by calling the `disableVestingSchedule` function, removing all existing schedule phases and creating a fixed schedule, which can't be released.

```
    /**
     * @notice Allows the owner to disable a vesting schedule and create a new one with
remaining time.
     * @dev Only the owner can call this function.
     * @param _beneficiary The address of the beneficiary whose schedule is to be
disabled.
     */
    function disableVestingSchedule(address _beneficiary, bool _disabled) external
onlyOwner {
        VestingSchedule storage schedule = vestingSchedules[_beneficiary];
        require(schedule.startTime != 0, "No vesting schedule for beneficiary");
        require(!schedule.isDisabled, "Schedule already disabled");
        require(_disabled, "Can only disable schedules");

        (uint256 totalVested, uint256 totalReleased, ) = getVestedAmount(_beneficiary);

        uint256 remainingTokens = totalAllocation[_beneficiary] - totalVested;
        uint256 totalAmount = totalAllocation[_beneficiary];

        uint256 originalEndTime = schedule.startTime +
schedule.phases[schedule.phases.length - 1].end;
        uint256 currentTime = block.timestamp;
        uint256 remainingTime = originalEndTime > currentTime ? originalEndTime -
currentTime : 0;

        if (remainingTime > 0 && remainingTokens > 0) {...}

        schedule.isDisabled = true;
        emit VestingScheduleDisabled(_beneficiary);
    }

    /**
     * @notice Allows a beneficiary to release their vested tokens.
     * @dev The function will revert if no tokens are available for release.
     */
    function release() external {
        VestingSchedule storage schedule = vestingSchedules[msg.sender];
        require(schedule.startTime != 0, "No vesting schedule for caller");
        require(!schedule.isDisabled, "Vesting schedule is disabled");

        uint256 releasableAmount = calculateReleasableAmount(msg.sender);
        require(releasableAmount > 0, "No tokens available for release");

        schedule.releasedAmount += releasableAmount;
        token.safeTransfer(msg.sender, releasableAmount);

        emit TokensReleased(msg.sender, releasableAmount);
    }
```

Recommendation: Fix the logic.

Update: The function has been reworked to transfer out releasable amount for beneficiary and transfer remaining amount to the owner of the contract.

Medium severity issues

1. Non-standard behavior (FibonToken)

Status: Fixed

The transfer amount is required to be not less than the current `transferFee`. A zero-amount transfer mail fail in case of positive `transferFee`.

The required allowance must be greater than the transferred amount for paying the `transferFee`.

Non-standard behavior may result in unexpected consequences such as incompatibility with other protocols.

```
    /**
     * @dev Override transferFrom to add fee collection
     */
    function transferFrom(address from, address to, uint256 amount) public virtual
```

```
override returns (bool) {
        require(!isBlacklisted[from], "Sender is blacklisted");
        require(!isBlacklisted[to], "Recipient is blacklisted");

        if (transferFee > 0 && from != feeCollector) {
            require(amount >= transferFee, "Amount less than fee");
            uint256 totalAmount = amount + transferFee;
            require(balanceOf(from) >= totalAmount, "Insufficient balance for transfer
with fee");

            uint256 currentAllowance = allowance(from, msg.sender);
            require(currentAllowance >= totalAmount, "Insufficient allowance for
transfer with fee");

            _spendAllowance(from, msg.sender, totalAmount);

            _transfer(from, feeCollector, transferFee);

            _transfer(from, to, amount);
            return true;
        }

        return super.transferFrom(from, to, amount);
    }
```

Recommendation: Common transfer fee implementation is to reduce receiver's amount by the transfer fee.

2. Inefficient ETH receiving (FibonMultiSig)

Status: Fixed

The `submitTransaction` function registers submitted transaction, which may contain a `msg.value`. In order to execute such pending transaction, one must send ETH to the contract directly in separate transaction. The `receive` and `fallback` functions are empty and duplicates each other.

```
    function submitTransaction(address destination, uint value, bytes memory data)
public onlyOwner returns (uint transactionId) {
        transactionId = transactionCount;
        transactions[transactionId] = Transaction({
            destination: destination,
            value: value,
            data: data,
            executed: false,
            confirmations: 0
        });
        transactionCount += 1;
        emit Submission(transactionId);
        confirmTransaction(transactionId);
    }
```

Recommendation: The `submitTransaction` should be marked as payable and checked for `msg.value == value`.

3. Claim of pre-launch phase is non-linear (FibonICO)

Status: Fixed

The `claimPreLaunchTokens` function is non-linear, it calculates the claimable amount for remaining tokens, but not for the vested ones, e.g., first claim of 100 vested tokens at 50% of duration is 50 tokens, but the second claim at 75% is not 25, but 0.75*50=37.5 tokens.

```
    function claimPreLaunchTokens() external nonReentrant {
        require(block.timestamp >= preLaunchClaimTime[msg.sender], "Cliff period not 
over");
        uint256 claimableAmount = calculateClaimableAmount(msg.sender);
        require(claimableAmount > 0, "No tokens to claim");

        preLaunchPurchases[msg.sender] -= claimableAmount;
        token.safeTransfer(msg.sender, claimableAmount);

        emit TokensClaimed(msg.sender, claimableAmount);
    }

    function calculateClaimableAmount(address _buyer) public view returns (uint256) {
        if (block.timestamp < preLaunchClaimTime[_buyer]) {
            return 0;
        }
        uint256 totalVestingTime = PRELAUNCH_VESTING;
        uint256 elapsedTime = block.timestamp - preLaunchClaimTime[_buyer];
        if (elapsedTime >= totalVestingTime) {
            return preLaunchPurchases[_buyer];
        } else {
            return (preLaunchPurchases[_buyer] * elapsedTime) / totalVestingTime;
        }
    }
```

Recommendation: Fix the logic or add documentation.

4. Lack of testing and documentation (All project contracts)

Status: Fixed

The project doesn't contain any kind of tests, the documentation is a single file for deployment guide.

Recommendation: Add tests with reasonable coverage, include documentation in the NatSpec format.

Update: Testing coverage is above 90% of statements and functions for all of project contracts.

Low severity issues

1. Duplicated event (FibonToken)

Status: Fixed

The event `Minted(to, amount)` duplicates the `Transfer(from,to,value)` event and can be safely removed.

```
    function mint(address to, uint256 amount) public onlyAdmin {
        _mint(to, amount);
        emit Minted(to, amount);
    }
```

2. Inefficient search (FibonMultiSig)

Status: Fixed

The `isOwner` function should be optimized to O(1) by using mapping for owners.

```
    function isOwner(address addr) public view returns (bool) {
        for (uint i = 0; i < owners.length; i++) {
            if (owners[i] == addr) {
                return true;
            }
        }
        return false;
    }
```

3. Immutable owners (FibonMultiSig)
Status: Acknowledged

The list of FibonMultiSig owners and decision threshold are set up in the contract's constructor and can't be updated. If there is a suspicion that the owner's account has been compromised, the only reasonable solution is to forfeit established multisig and move all funds and governance roles to another instance.

4. Inconsistent version of imports (FibonICO)
Status: Fixed

The FibonICO requires v4.x versions of OpenZeppelin to be imported, while the FibonToken is compatible with v5.x versions only.

5. Constructor parameters are not validated (FibonVesting)
Status: Fixed

The `cliff` parameter of the constructor can be set greater than total duration of the vesting, resulting in 100% unlock after the cliff period, ignoring the `duration` parameter. The `updateVestingSchedule` function includes this safety check.

```
    constructor(
        address initialOwner,
        IERC20 _token,
        uint256 _cliff,
        uint256 _duration
    ) Ownable(initialOwner) {
        token = _token;
        cliff = _cliff;
        duration = _duration;
        startTime = block.timestamp;
    }
```

Recommendation: Include `require(_duration > _cliff, "Invalid vesting schedule");` line.

Conclusion

Fibon FibonToken, FibonMultiSig, FibonICO, FibonVesting, All project contracts contracts were audited. 4 high, 4 medium, 5 low severity issues were found.
2 high, 4 medium, 4 low severity issues have been fixed in the update.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without 0xGuard prior written consent.

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts 0xGuard to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.
OxGuard retains exclusive publishing rights for the results of this audit on its website and social networks.

Slither output

```
INFO:Detectors:
Reentrancy in FibonMultiSig.executeTransaction(uint256) (contracts/
Multisig.sol#170-182):
        External calls:
        - (success,None) = transaction.destination.call{value: transaction.value}
(transaction.data) (contracts/Multisig.sol#174)
        State variables written after the call(s):
        - transaction.executed = false (contracts/Multisig.sol#179)
        FibonMultiSig.transactions (contracts/Multisig.sol#34) can be used in cross
function reentrancies:
        - FibonMultiSig.confirmTransaction(uint256) (contracts/Multisig.sol#132-140)
        - FibonMultiSig.executeTransaction(uint256) (contracts/Multisig.sol#170-182)
        - FibonMultiSig.notExecuted(uint256) (contracts/Multisig.sol#72-75)
        - FibonMultiSig.submitTransaction(address,uint256,bytes) (contracts/
Multisig.sol#111-125)
        - FibonMultiSig.submitWithdrawal(address,uint256) (contracts/
Multisig.sol#149-163)
        - FibonMultiSig.transactionExists(uint256) (contracts/Multisig.sol#63-66)
        - FibonMultiSig.transactions (contracts/Multisig.sol#34)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-
vulnerabilities
INFO:Detectors:
FibonVesting.calculateReleasableAmount(address) (contracts/Vesting.sol#279-312)
performs a multiplication on the result of a division:
        - progress = (timeInPhase * 1e18) / phaseDuration (contracts/Vesting.sol#304)
        - vestedInPhase = (allocation * phase.percentage * progress) / (100 * 1e18)
(contracts/Vesting.sol#305)
FibonVesting.getVestedAmount(address) (contracts/Vesting.sol#333-375) performs a
multiplication on the result of a division:
        - progress = (timeInPhase * 1e18) / phaseDuration (contracts/Vesting.sol#364)
        - vestedInPhase = (allocation * phase.percentage * progress) / (100 * 1e18)
(contracts/Vesting.sol#365)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-
multiply
INFO:Detectors:
FibonVesting._createVestingSchedule(address,uint256,uint8,uint256) (contracts/
Vesting.sol#437-462) uses a dangerous strict equality:
        - require(bool,string)(vestingSchedules[_beneficiary].startTime == 0,Vesting
schedule already exists) (contracts/Vesting.sol#446)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-
strict-equalities
INFO:Detectors:
FibonVesting.calculateReleasableAmount(address).totalVested (contracts/Vesting.sol#285)
is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-
local-variables
INFO:Detectors:
FibonToken.permit(address,address,uint256,uint256,uint8,bytes32,bytes32).owner
(contracts/FibonToken.sol#162) shadows:
        - Ownable.owner() (node_modules/@openzeppelin/contracts/access/
Ownable.sol#56-58) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-
shadowing
INFO:Detectors:
FibonICO.updateCliffPeriod(uint256) (contracts/ICO.sol#232-235) should emit an event
for:
        - preLaunchCliff = newCliff (contracts/ICO.sol#234)
FibonICO.updateVestingPeriod(uint256) (contracts/ICO.sol#241-244) should emit an event
for:
        - preLaunchVesting = newVesting (contracts/ICO.sol#243)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-
arithmetic
INFO:Detectors:
Reentrancy in FibonMultiSig.executeTransaction(uint256) (contracts/
Multisig.sol#170-182):
        External calls:
        - (success,None) = transaction.destination.call{value: transaction.value}
(transaction.data) (contracts/Multisig.sol#174)
        Event emitted after the call(s):
        - Execution(transactionId) (contracts/Multisig.sol#176)
        - ExecutionFailure(transactionId) (contracts/Multisig.sol#178)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-
vulnerabilities-3
INFO:Detectors:
FibonICO.buyTokens(uint8) (contracts/ICO.sol#120-146) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= startTime && block.timestamp <=
endTime,ICO is not active) (contracts/ICO.sol#121)
```

```
        - require(bool,string)(block.timestamp >= phase.startTime && block.timestamp <=
phase.endTime,Phase is not active) (contracts/ICO.sol#126)
FibonICO.claimPreLaunchTokens() (contracts/ICO.sol#151-160) uses timestamp for
comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= preLaunchClaimTime[msg.sender],Cliff
period not over) (contracts/ICO.sol#152)
        - require(bool,string)(claimableAmount > 0,No tokens to claim) (contracts/
ICO.sol#154)
FibonICO.calculateClaimableAmount(address) (contracts/ICO.sol#167-183) uses timestamp
for comparisons
        Dangerous comparisons:
        - block.timestamp < preLaunchClaimTime[_buyer] (contracts/ICO.sol#168)
        - elapsedTime >= totalVestingTime (contracts/ICO.sol#176)
FibonICO.withdrawFunds(address) (contracts/ICO.sol#202-207) uses timestamp for
comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp > endTime,ICO has not ended yet)
(contracts/ICO.sol#203)
FibonVesting.calculateReleasableAmount(address) (contracts/Vesting.sol#279-312) uses
timestamp for comparisons
        Dangerous comparisons:
        - currentTime < phaseStartTime (contracts/Vesting.sol#295)
        - currentTime >= phaseEndTime (contracts/Vesting.sol#297)
FibonVesting.getVestedAmount(address) (contracts/Vesting.sol#333-375) uses timestamp
for comparisons
        Dangerous comparisons:
        - currentTime < phaseStartTime (contracts/Vesting.sol#356)
        - currentTime >= phaseEndTime (contracts/Vesting.sol#358)
FibonVesting._createVestingSchedule(address,uint256,uint8,uint256) (contracts/
Vesting.sol#437-462) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(vestingSchedules[_beneficiary].startTime == 0,Vesting
schedule already exists) (contracts/Vesting.sol#446)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-
timestamp
INFO:Detectors:
FibonVesting._createVestingSchedule(address,uint256,uint8,uint256) (contracts/
Vesting.sol#437-462) has costly operations inside a loop:
        - totalAllocated += _amount (contracts/Vesting.sol#459)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-
```

```
operations-inside-a-loop
INFO:Detectors:
Low level call in FibonMultiSig.executeTransaction(uint256) (contracts/
Multisig.sol#170-182):
        - (success,None) = transaction.destination.call{value: transaction.value}
(transaction.data) (contracts/Multisig.sol#174)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-
calls
INFO:Detectors:
Parameter FibonToken.blacklistAddress(address)._account (contracts/FibonToken.sol#77)
is not in mixedCase
Parameter FibonToken.unblacklistAddress(address)._account (contracts/FibonToken.sol#89)
is not in mixedCase
Parameter FibonICO.buyTokens(uint8)._phase (contracts/ICO.sol#120) is not in mixedCase
Parameter FibonICO.calculateClaimableAmount(address)._buyer (contracts/ICO.sol#167) is
not in mixedCase
Parameter FibonICO.getPhase(uint8)._phase (contracts/ICO.sol#190) is not in mixedCase
Parameter FibonICO.withdrawFunds(address)._recipient (contracts/ICO.sol#202) is not in
mixedCase
Parameter FibonVesting.calculateReleasableAmount(address)._beneficiary (contracts/
Vesting.sol#279) is not in mixedCase
Parameter FibonVesting.recoverERC20(address,uint256)._token (contracts/Vesting.sol#321)
is not in mixedCase
Parameter FibonVesting.recoverERC20(address,uint256)._amount (contracts/
Vesting.sol#321) is not in mixedCase
Parameter FibonVesting.getVestedAmount(address)._beneficiary (contracts/Vesting.sol#333)
is not in mixedCase
Parameter FibonVesting.getVestedPercentage(address)._beneficiary (contracts/
Vesting.sol#382) is not in mixedCase
Parameter FibonVesting.disableVestingSchedule(address)._beneficiary (contracts/
Vesting.sol#405) is not in mixedCase
Parameter FibonVesting.initializeVestingSchedules(address[],uint256[],uint8[],uint256)._
beneficiaries (contracts/Vesting.sol#473) is not in mixedCase
Parameter
FibonVesting.initializeVestingSchedules(address[],uint256[],uint8[],uint256)._amounts
(contracts/Vesting.sol#474) is not in mixedCase
Parameter
FibonVesting.initializeVestingSchedules(address[],uint256[],uint8[],uint256)._typeIds
(contracts/Vesting.sol#475) is not in mixedCase
Parameter
FibonVesting.initializeVestingSchedules(address[],uint256[],uint8[],uint256)._startTime
```

```
(contracts/Vesting.sol#476) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-
solidity-naming-conventions
INFO:Detectors:
FibonICO.constructor(IERC20,uint256,uint256,uint256,uint256) (contracts/ICO.sol#92-114)
uses literals with too many digits:
        - ico2 = Phase(44000000 * 10 ** 18,0,_startTime + 5184000,_startTime + 7776000)
(contracts/ICO.sol#112)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-
digits
INFO:Detectors:
FibonICO.hardCap (contracts/ICO.sol#29) should be immutable
FibonICO.startTime (contracts/ICO.sol#23) should be immutable
FibonICO.token (contracts/ICO.sol#17) should be immutable
FibonVesting.token (contracts/Vesting.sol#16) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-
variables-that-could-be-declared-immutable
INFO:Slither:. analyzed (31 contracts with 93 detectors), 39 result(s) found
```

0x Guard